Programming Paradigms

Type Systems (Part 3)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2023

Overview

- **Introduction**
- **Types in Programming Languages**
- **Type Equivalence**
- **Type Compatibility and Conversions**
- **Formally Defined Type Systems**
  - Arithmetic Expressions
  - Lambda Calculus

Lambda Calculus

- **Core language that captures the essence of most PLs**
- **Serves both as**
  - A simple PL one could (in principle) develop in
  - A mathematical object for formally reasoning about PLs

Functional Abstraction

- **Key feature: Procedural (or functional) abstraction**
- **Everything is a function, e.g.,**
  - Arguments accepted by functions
  - Values returned by functions
- **Notation:** $\lambda n \,.\, \langle result \rangle$
  - Means "The function that, for each n, yields $\langle result \rangle$"

Examples

$\lambda x.\ x$ $\longrightarrow$ function that takes argument x and returns it

$\lambda x.$ if x then false else true $\longrightarrow$ negation of x

$(\lambda x.$ if x then false else true$)$ true $\longrightarrow$ apply above function to true, which yields false

Grammar of Untyped λ-Calculus

$t ::=$		terms
	$x \mid$	variables
	$\lambda x . t \mid$	abstraction
	$t \; t$	application

Parenthesis may be added for clarity, otherwise the following holds:

- Application is left-associative
  - ↳ E.g. $s\ t\ u$ means $(s\ t)\ u$
- Bodies of abstractions extend as far to the right as possible
  - ↳ E.g., $\lambda x . \lambda y . x\ y\ x$ means $\lambda x . (\lambda y . ((x\ y)\ x))$

Scope

- Occurrence of variable x is "bound" if it occurs in the body t of an abstraction $\lambda x.t$
- Otherwise, occurrence of x is "free"

Examples:

$x \; y$ → both x and y free

$\lambda y. x\, y$ → x is free, y is bound

$\lambda z. \lambda x. \lambda y. x\,(y\, z)$ → all variables are bound

$(\lambda x. x)\, x$ → first occurrence x is bound, second is free

Semantics

Each step of computation: Apply one function to one argument

$$(\lambda x.\ t_{12})\ t_2 \longrightarrow \underbrace{[x \mapsto t_2]\, t_{12}}.$$

Means "evaluates to"   Means "replace all x in t_{12} by t_2"

Examples:

$(\lambda x. x)\ y \longrightarrow y$

$(\lambda x. x\ (\lambda x. x))\ (u\ r) \longrightarrow (u\ r)\ (\lambda x. x)$

$(\lambda x. \lambda y. (\text{if iszero } x \text{ then } y \text{ else succ } y)\ 0)\ 0$

$\rightarrow \lambda y. (\text{if iszero } 0 \text{ then } y \text{ else succ } y)\ 0$

$\rightarrow \text{if iszero } 0 \text{ then } 0 \text{ else succ } 0 \rightarrow \ldots \rightarrow 0$

Quiz: Lambda-Calculus

What do the following terms evaluate to?

(a) $(\lambda z\ .\ succ\ z)\ 0$

(b) $\lambda x\ .\ true$

(c) $((\lambda a\ .\ \lambda b\ .\ if\ a\ then\ b\ else\ a)\ false)\ true$

Quiz: Lambda-Calculus

What do the following terms evaluate to?

(a) $(\lambda z\ .\ succ\ z)\ 0$

Function that computes the successor of a z, applied to 0, yields 1

(b) $\lambda x\ .\ true$

(c) $((\lambda a\ .\ \lambda b\ .\ if\ a\ then\ b\ else\ a)\ false)\ true$

Quiz: Lambda-Calculus

What do the following terms evaluate to?

(a) $(\lambda z\ .\ succ\ z)\ 0$ — **Function that computes the successor of a z, applied to 0, yields 1**

(b) $\lambda x\ .\ true$ — **Function that always returns true**

(c) $((\lambda a\ .\ \lambda b\ .\ if\ a\ then\ b\ else\ a)\ false)\ true$

Quiz: Lambda-Calculus

What do the following terms evaluate to?

(a) $(\lambda z\ .\ succ\ z)\ 0$ **Function that computes the successor of a z, applied to 0, yields 1**

(b) $\lambda x\ .\ true$ **Function that always returns true**

(c) $((\lambda a\ .\ \lambda b\ .\ if\ a\ then\ b\ else\ a)\ false)\ true$

Applying $\lambda a \ldots$ to false yields another function $\lambda b \ldots$. Applying that function to true yields false.

Let's Add Types

- **As for arithmetic expressions, syntax allows both**
  - Meaningful programs
    - $\lambda x \, . \, x \; true$
  - Meaningless programs
    - $true \; \lambda x \, . \, x$
    - $\lambda x \, . \, y$

Let's Add Types

- **As for arithmetic expressions, syntax allows both**
  - Meaningful programs
    - $\lambda x \,.\, x \; true$
  - Meaningless programs
    - $true \; \lambda x \,.\, x$
    - $\lambda x \,.\, y$

Cannot apply $true$ because it's not a function

y is not bound

Syntax of Types

Let's focus on boolean expressions & λ-calculus
(i.e., no natural numbers)

T ::=	types
Bool |	boolean (as before)
$T \to T$	function type, mapping one type to another

Type constructor is right-associative, i.e., $T_1 \to T_2 \to T_3$ means $T_1 \to (T_2 \to T_3)$

Example: $Bool \to Bool \to Bool$
is type of function that takes a Bool and returns another function that takes a Bool & then returns a Bool

Type Annotations

Instead of $\lambda x. t$

we write $\lambda x : T_1 . t$ ⟵ Means a function that takes an argument of type T_1

or

$\lambda x : T_1 . t : T_1 \to T_2$ ⟵ (as above) and returns a value of type T_2

Example: $\lambda x : \text{Bool} .\ \text{if } x \text{ then true else } x : \text{Bool} \to \text{Bool}$

Typing Context

- **Extend typing relation**
  - So far, binary relation $t : T$
  - Now, ternary relation $\Gamma \vdash t : T$
    - Means “Term t has type T under the assumptions in Γ”
  - Γ is the typing context (or type environment)
    - Set of assumptions about types of free variables
    - If no assumptions, we write $\vdash t : T$

Quiz: Typed Lambda-Calculus

Under which context Γ does the following hold: $\Gamma \vdash f\ x : Bool$

Quiz: Typed Lambda-Calculus

Under which context Γ does the following hold: $\Gamma \vdash f\ x : Bool$

Answer:

$f : Bool \rightarrow Bool, x : Bool$

Type Rules

New rules for typed λ-calculus:

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

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

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\, t_2 : T_2} \quad (\text{T-App})$$

Adapted rules f. boolean expressions

$$\frac{}{\Gamma \vdash \text{true} : \text{Bool}} \quad (\text{T-True})$$

$$\frac{}{\Gamma \vdash \text{false} : \text{Bool}} \quad (\text{T-False})$$

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

Can be Bool or any function type

Example of Type Derivation

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

Quiz: Type Derivation (2)

Show (by drawing the derivation tree) that the following term has the indicated type:

$$f : Bool \to Bool \vdash$$
$$f\ (if\ false\ then\ true\ else\ false) : Bool$$

How many times do you have to write "Bool"?

Quiz

B means Bool

$\# B = 23$

$$\dfrac{f:B\to B \in f:B\to B}{f:B\to B \vdash f:B\to B}\,(\text{T-Var}) \qquad \dfrac{\dfrac{}{f:B\to B \vdash \text{false}:B}\,(\text{T-False}) \quad \dfrac{}{f:B\to B \vdash \text{true}:B}\,(\text{T-True}) \quad \dfrac{}{f:B\to B \vdash \text{false}:B}\,(\text{T-False})}{f:B\to B \vdash \text{if false then true else false}:B}\,(\text{T-If})$$

$$\overline{f:B\to B \vdash f\,(\text{if false then true else false}):B}\;(\text{T-App})$$

Another Example

if true then $(\lambda x{:}Bool.\ true)$ <--------- Yields a Bool

else $(\lambda x{:}Bool.\ \lambda y{:}Bool.\ y)$ <---- Yields a function

Not allowed because T-If rule requires same type

Outlook

- **Many extension to the simple, typed λ-calculus**
  - E.g., Tuples, records, exceptions, subtyping
- **See book *Types and Programming Languages* by Benjamin Pierce**

Overview

- **Introduction**
- **Types in Programming Languages**
- **Type Equivalence**
- **Type Compatibility and Conversions**
- **Formally Defined Type Systems**
  - Arithmetic Expressions
  - Lambda Calculus ✔