Programming Paradigms

Type Systems (Part 6)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart
Summer 2020
Overview

- Introduction
- Types in Programming Languages
- Polymorphism
- Type Equivalence
- Type Compatibility
- Formally Defined Type Systems
Formally Defined Type Systems

- **Type systems are**
  - implemented in a compiler
  - formally described
  - and sometimes both

- **Active research area with dozens of papers each year**
  - Focus: New languages and strong type guarantees

- **Example here:** Typed expressions
Typed Expressions: Syntax

\[ t ::= \text{true} \mid \text{false} \mid \text{if } t \text{ then } t \text{ else } t \mid 0 \mid \text{succ } t \mid \text{pred } t \mid \text{iszero } t \]

Examples

\[ \text{succ } 0 \quad (= 1) \]
\[ \text{if } (\text{iszero } (\text{pred } (\text{succ } 0))) \text{ then } 0 \text{ else } (\text{succ } 0) \quad (= 0) \]

(semantics: not formally defined)
Not All Expressions Make Sense

- Only **some expressions** can be evaluated
  - Other don’t make sense
  - Implementation of the language would get **stuck** or throw a **runtime error**
Types to the Rescue

- **Use types to check** whether an expression is meaningful
  - If term $t$ has a type $T$, then its evaluation won’t get stuck
  - Written as $t : T$

- **Two types**
  - $Nat$ .. natural numbers
  - $Bool$ .. Boolean values
Examples

if (iszero 0) then true else 0

succ (if 0 then true else (pred false))

if true then false else true : Bool

pred (succ (succ 0)) : Nat

equation that don't make sense
Type Rules

Background: \[ \frac{A}{B} \quad \text{ rule} \]

if \( A \) is true,
then \( B \) is true

\[ \text{Bool: } \]

\[ \frac{}{\text{true: Bool}} \quad (T-\text{True}) \]

\[ \frac{}{\text{false: Bool}} \quad (T-\text{False}) \]

\[ \frac{t_1: \text{Bool} \, \quad t_2: T \, \quad t_3: T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3: T} \quad (T-\text{If}) \]

\[ \frac{B}{\quad \text{ axiom} \quad} \]

\( B \) is always true

\[ \frac{}{\text{Nat: } \quad \frac{}{0: \text{Nat}} \quad (T-\text{Zero})} \]

\[ \frac{t_1: \text{Nat}}{\text{succ } t_1: \text{Nat}} \quad (T-\text{Succ}) \]

\[ \frac{t_1: \text{Nat}}{\text{pred } t_1: \text{Nat}^+} \quad (T-\text{Pred}) \]

\[ \frac{t_1: \text{Nat}^+}{\text{iszero } t_1: \text{Bool}} \quad (T-\text{IsZero}) \]
Type Checking Expressions

- **Typing relation**: Smallest binary relation between terms and types that satisfies all instances of the rules

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

- **Type derivation**: Tree of instances of the typing rules that shows $t : T$
Type Derivation: Example 1

true: Bool       false: Bool       true: Bool
________________   ____________________   ____________________
                   (T-True)               (T-False)             (T-True)

if true then false else true : Bool

(T-If)
Example 2:

Can't apply any axiom or rule.
Expression is not well typed!

\[
\begin{align*}
\text{if } 0 \text{ then true else (pred false)} : \text{Nat} \\
\text{succ (if } 0 \text{ then true else (pred false))} : \text{Nat}
\end{align*}
\]
Quiz: Typing Derivation

Find the typing derivation for the following expression:

\[ \text{if false then (pred(pred 0)) else (succ 0)} \]

How many axioms and rules do you need?
3 axioms, 4 rules

\[ \text{false : Bool} \]
\[ \text{if false then } (\text{pred (pred (0))}) \text{ else } (\text{succ (0)) : Nat} \]
Type Inference

Some PLs are **statically typed** but allow programmers to **omit some type annotations**

- Get **guarantees** of static type checking
- Without paying the cost of full type annotations
- Different from gradual typing, where programmer decides when and where to annotate types
Example

// Scala
var businessName = "Montreux Jazz Cafe"

def squareOf(x: Int) = x * x

businessName = squareOf(23)
// Scala
var businessName = "Montreux Jazz Cafe"

def squareOf(x: Int) = x * x

businessName = squareOf(23)

Inferred to be a String

Inferred to return an Int
Example

// Scala
var businessName = "Montreux Jazz Cafe"

def squareOf(x: Int) = x * x

businessName = squareOf(23)

Inferred to
be a String

Inferred to
return an Int

Compile-time type error:
Can’t assign Int to String variable
Quiz: Types

Which of the following statements is true?

- Types are compatible if and only if they are equal
- Coercions mean that a programmer casts a value from one type to another type
- Type conversions are guaranteed to preserve the meaning of a value
- PLs with type inference may provide static type guarantees

Please vote in Ilias.
Quiz: Types

Which of the following statements is true?

- Types are compatible if and only if they are equal
- Coercions mean that a programmer casts a value from one type to another type
- Type conversions are guaranteed to preserve the meaning of a value
- PLs with type inference may provide static type guarantees

Please vote in Ilias.