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 ::= true |
      false |
      if t then t else t |
      0 |
      succ t |
      pred t |
      iszero t
```

(semantics: not formally defined)

Examples

succ 0 (= 1)

```
if (iszero (pred (succ 0)))
   then 0
   else (succ 0)          (= 0)
```

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))

} expression that don't make sense

if true then false else true : Bool

pred (succ (succ 0)) : Nat

Type Rules

Background: $\dfrac{A}{B}$ .. rule

if A is true, then B is true

$\dfrac{}{B}$ .. axiom

B is always true

Bool:

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

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

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

Nat:

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

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

$$\frac{t_1 : \text{Nat}}{\text{pred } t_1 : \text{Nat}} \quad (\text{T-Pred})$$

$$\frac{t_1 : \text{Nat}}{\text{iszero } t_1 : \text{Bool}} \quad (\text{T-IsZero})$$

Type Checking Expressions

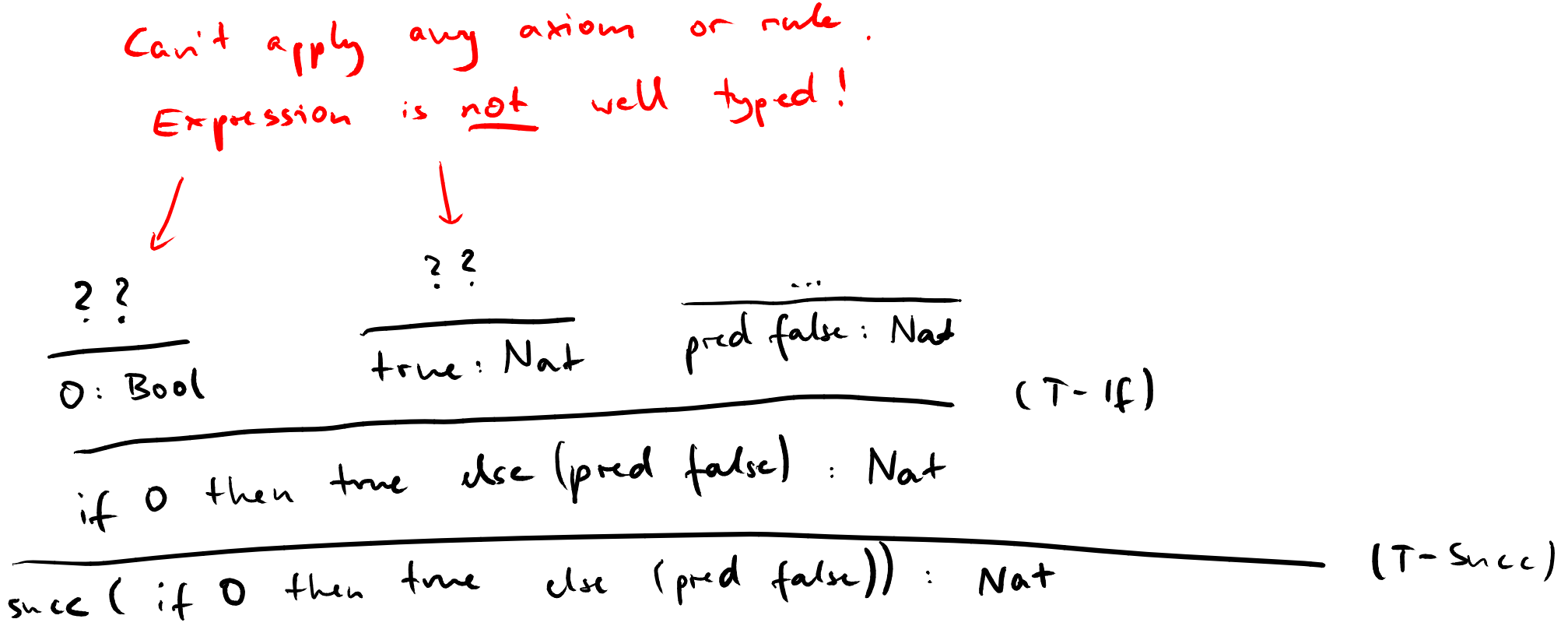
- **Typing relation: Smallest binary relation between terms and types that satisfies all instances of the rules**
- **Term t is typable (or well typed) if there is some T such that $t : T$**
- **Type derivation: Tree of instances of the typing rules that shows $t : T$**

Type Derivation : Example 1

$$\dfrac{\dfrac{}{\texttt{true} : \texttt{Bool}}\,(\text{T-True}) \quad \dfrac{}{\texttt{false} : \texttt{Bool}}\,(\text{T-False}) \quad \dfrac{}{\texttt{true} : \texttt{Bool}}\,(\text{T-True})}{\texttt{if true then false else true} : \texttt{Bool}}\,(\text{T-If})$$

✓

Example 2 :


Can't apply any axiom or rule.
Expression is not well typed!
? ?
0 : Bool
? ?
true : Nat
...
pred false : Nat
(T-If)
if 0 then true else (pred false) : Nat
(T-Succ)
succ (if 0 then true else (pred false)) : Nat

Quiz: Typing Derivation

Find the typing derivation for the following expression:

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

How many axioms and rules do you need?

Please vote in Ilias.

Quiz:

3 axioms, 4 rules

$$\dfrac{\dfrac{}{\text{false} : \text{Bool}}\,(\text{T-False}) \quad \dfrac{\dfrac{\dfrac{}{0 : \text{Nat}}\,(\text{T-Zero})}{\text{pred } 0 : \text{Nat}}\,(\text{T-Pred})}{\text{pred (pred (0)) : Nat}}\,(\text{T-Pred}) \quad \dfrac{\dfrac{}{0 : \text{Nat}}\,(\text{T-Zero})}{\text{succ } 0 : \text{Nat}}}{\text{if false then (pred (pred (0)) else (succ 0) : Nat}}\,(\text{T-If})$$

⋮

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

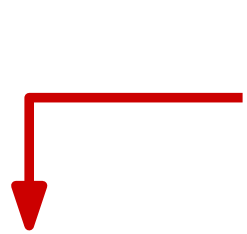
```
// Scala
var businessName = "Montreux Jazz Cafe"

def squareOf(x: Int) = x * x



businessName = squareOf(23)
```

Example



Inferred to be a `String`

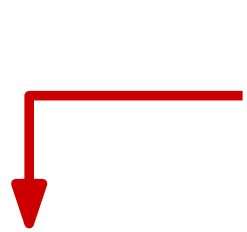
```
// Scala
var businessName = "Montreux Jazz Cafe"

def squareOf(x: Int) = x * x
```

Inferred to return an `Int`

```
businessName = squareOf(23)
```

Example



Inferred to be a `String`

```
// Scala
var businessName = "Montreux Jazz Cafe"

def squareOf(x: Int) = x * x
```

Inferred to return an `Int`

```
businessName = squareOf(23)
```

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.