Programming Paradigms

Lecture 9: Types (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2019/2020

Wake-up Exercise

What does the following C++ code print?

```
#include<iostream>
using namespace std;
int N = 5;

int main()
{
  static int x = 1;
  if (cout << x << " " && x++ < N && main())
  { }
  return 0;
}
```

https://ilias3.uni-stuttgart.de/vote/0ZT9

Wake-up Exercise

What does the following C++ code print?

```
#include<iostream>
using namespace std;
int N = 5;

int main()
{
  static int x = 1;
  if (cout << x << " " && x++ < N && main())
  { }
  return 0;
}
```

Result: 1, 2, 3, 4, 5

https://ilias3.uni-stuttgart.de/vote/0ZT9

Wake-up Exercise

What does the following C++ code print?

```
#include<iostream>
using namespace std;
int N = 5;

int main()
{
  static int x = 1;
  if (cout << x << " " && x++ < N && main())
  { }
  return 0;
}
```

Recursive call of `main`

Result: 1, 2, 3, 4, 5

https://ilias3.uni-stuttgart.de/vote/0ZT9

Wake-up Exercise

What does the following C++ code print?

```
#include<iostream>
using namespace std;
int N = 5;

int main()
{
  static int x = 1;
  if (cout << x << " " && x++ < N && main())
  { }
  return 0;
}
```

Variable keeps its value across invocations of the function

Result: 1, 2, 3, 4, 5

https://ilias3.uni-stuttgart.de/vote/0ZT9

Wake-up Exercise

What does the following C++ code print?

```
#include<iostream>
using namespace std;
int N = 5;

int main()
{
  static int x = 1;
  if (cout << x << " " && x++ < N && main())
  { }
  return 0;
}
```

Print current value of `x`

Result: 1, 2, 3, 4, 5

https://ilias3.uni-stuttgart.de/vote/0ZT9

Wake-up Exercise

What does the following C++ code print?

```
#include<iostream>
using namespace std;
int N = 5;

int main()
{
  static int x = 1;
  if (cout << x << " " && x++ < N && main())
  { }
  return 0;
}
```

Increment `x` until it reaches 5

Result: 1, 2, 3, 4, 5

https://ilias3.uni-stuttgart.de/vote/0ZT9

Overview

- **Introduction to Types and Type Systems**
- **Type Checking**
  - Type Equivalence
  - Type Compatibility 
  - Formal Definition of Type Systems
  - Type Inference
- **Equality Testing and Assignments**

Type Compatibility

- **Check whether combining two values is valid according to their types**
- **"Combining" may mean**
  - Assignment: Are left-hand side and right-hand side compatible?
  - Operators: Are operands compatible with the operator and with each other?
  - Function calls: Are actual arguments and formal parameters compatible?

Compatible ≠ Equal

Most PLs: Types may be compatible

even when not the same

Example (C):

```
double d = 2.3;
float f = d * 2;
int i = f;
printf("%d\n", i);
```

Compatible ≠ Equal (2)

- **Rules of PL define which types are compatible**
- **Examples of rules**
  - Can assign subtype to supertype:

    ```
    lhs = rhs;
    ```

  - Different number types are compatible with each other
  - Collections of same type are compatible, even if length differs

Type Conversions

When types aren't equal, they must be converted

- Option 1: Cast = explicit type conversion
  - Programmer changes value's type from T1 to T2
- Option 2: Coercion = implicit type conversion
  - PL allows values of type T1 in situation where type T2 expected
- Both options: Actual conversion happens at runtime

Runtime Behavior of Conversions

What happens during conversion?

Three cases:

- Types are structurally equivalent: Conversion is only conceptual, no instructions executed
- Types have different sets of values, but are represented in the same way in memory: May need check that value is in target type
- Different low-level representations: Need special instructions for conversion

Examples (Ada)

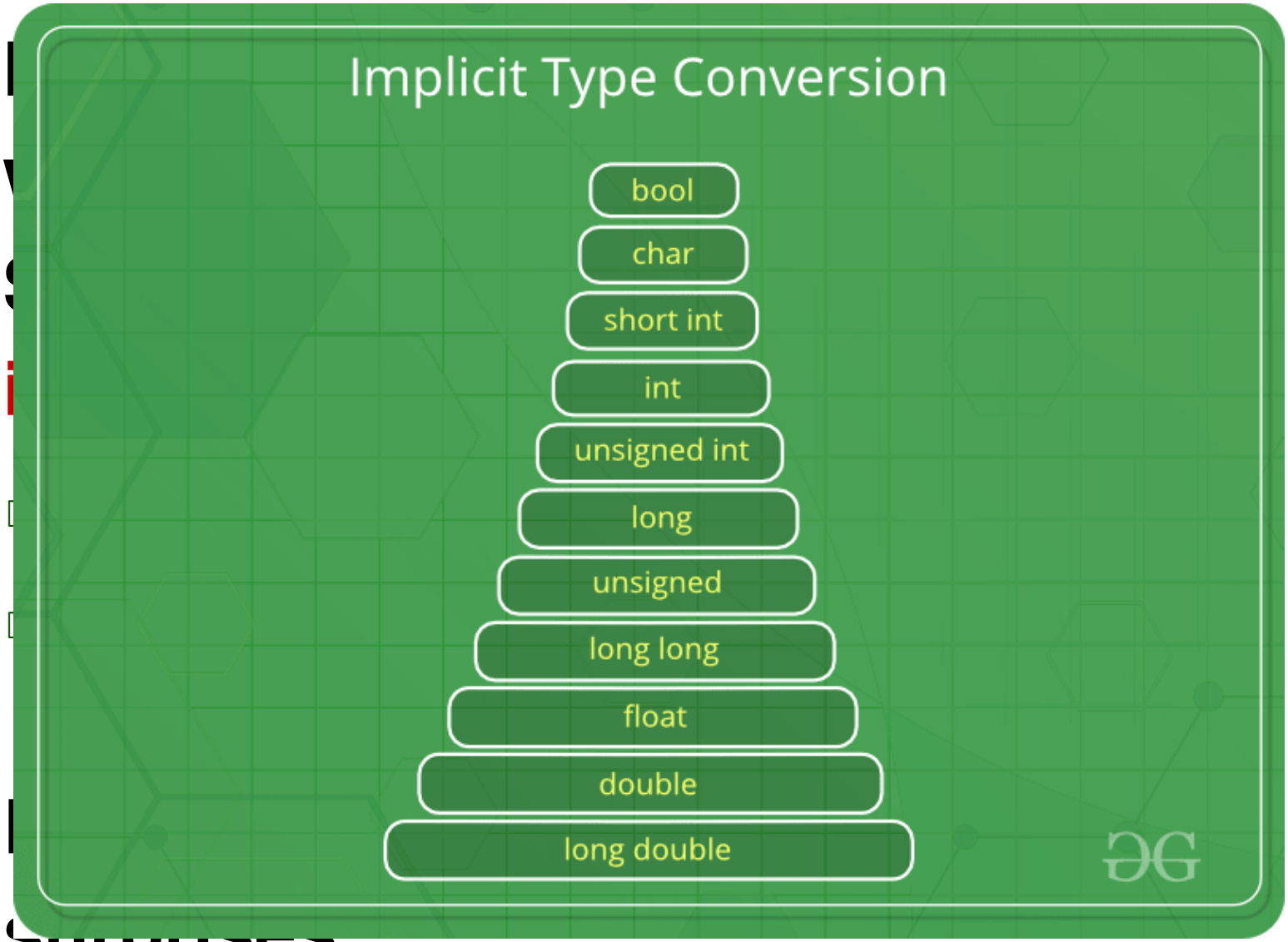
```
n: integer;
r: long_float;
t: integer range 0..100;          -- has alias: test_score
c: celsius_temp;                  -- type alias for integer

t := test_score (n);              -- runtime semantic check
n := integer (t);                 -- no check needed
r := long_float (n);              -- runtime conversion
n := integer (r);                 -- runtime conversion and check
n := integer (c);                 -- purely conceptual
c := celsius_temp (n);            -- purely conceptual
```

Coercions in C

- **Most primitive types are coerced whenever needed**
- **Some coercions may loose information**
  - `float` to `int`: Loose fraction
  - `int` to `char`: Causes `char` to overflow (and will give unexpected result)
- **Enable compiler warnings to avoid surprises**

Coercions in C

- I
- S
- I



Source: *geeksforgeeks.org*

Coercions in C: Demo

Demo: coercions.c

compile with gcc -Wconversion

Coercions in JavaScript

- **Almost all types are coerced when needed**
  - Rationale: Websites shouldn't crash
- **Some coercions make sense:**
  - `"number:" + 3` yields `"number:3"`
- **Many others are far from intuitive:**
  - `[1, 2] << "2"` yields `0`

More details and examples:
***The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript*. Pradel and Sen. ECOOP 2015**

Overview

- **Introduction to Types and Type Systems**
- **Type Checking**
  - Type Equivalence
  - Type Compatibility
  - Formal Definition of Type Systems 
  - Type Inference
- **Equality Testing and Assignments**

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 Expression : 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))

} expressions 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:

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

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

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

Nat:

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

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

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

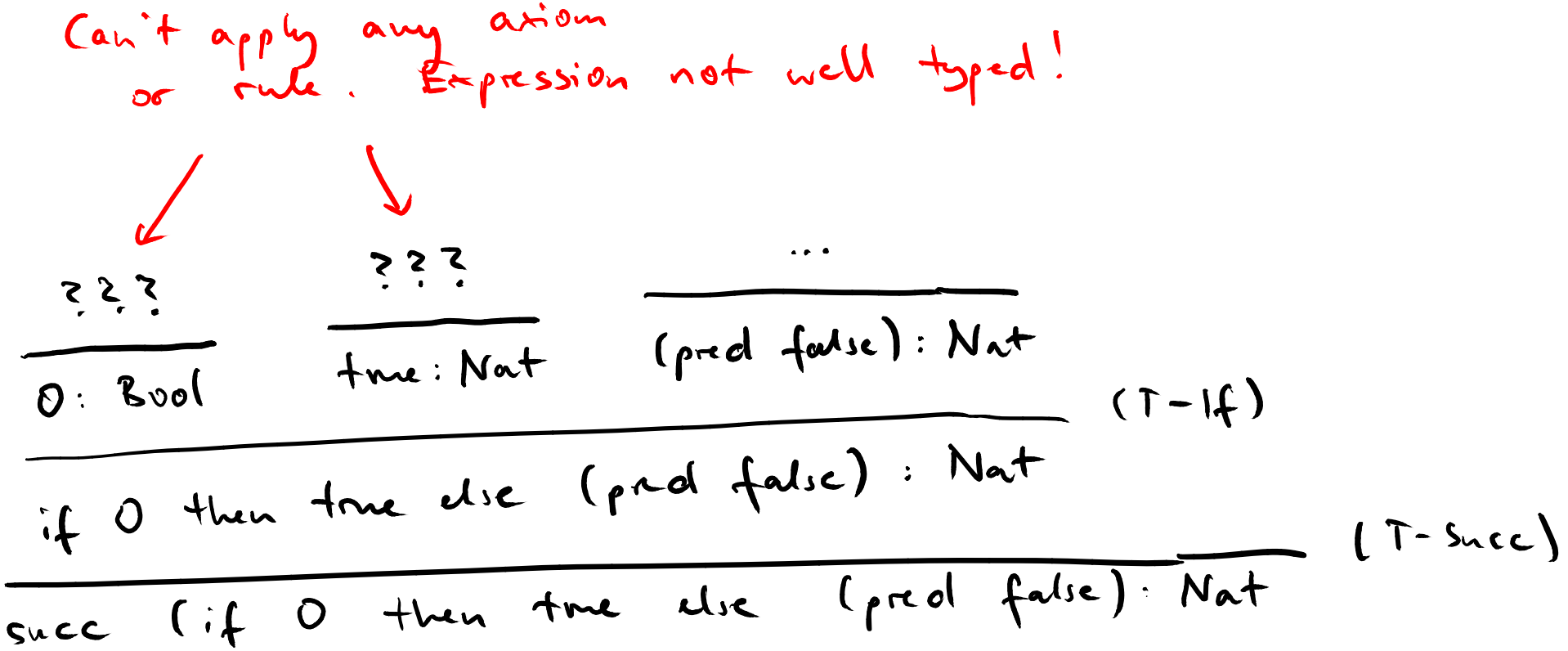
$$\frac{t_1 : \text{Nat}}{\text{iszero } t_1 : \text{Bool}} \quad (T\text{-IsZero})$$

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 Derivations: Example

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


Can't apply any axiom or rule. Expression 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?

https://ilias3.uni-stuttgart.de/vote/0ZT9

Quiz:

3 axioms, 4 rules

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

Overview

- **Introduction to Types and Type Systems**
- **Type Checking**
  - Type Equivalence
  - Type Compatibility
  - Formal Definition of Type Systems
  - Type Inference 
- **Equality Testing and Assignments**

Type Inference

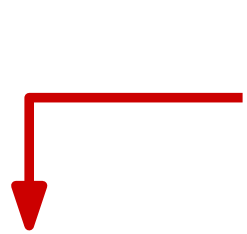
Some PLs are statically typed but allow programmers to omit some type annotations

- 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



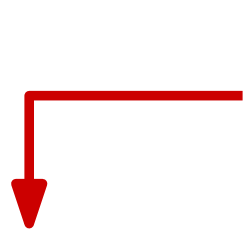
Inferred to be a `String`

```
// 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

Unifying Partial Type Information

- **Type inference algorithm:**
  Unifies partial type information for two values whenever type rules expect them to be the same
- **Type checking:**
  Find unique type for each value with no contradictions and no ambiguous occurrences of overloaded names

Overview

- **Introduction to Types and Type Systems**
- **Type Checking**
  - Type Equivalence
  - Type Compatibility
  - Formal Definition of Type Systems
  - Type Inference
- **Equality Testing and Assignments** 

Equality Testing

- **Equality operator: Many possible meanings**
  - E.g., for two strings
    - Are the strings aliases?
    - Are they bitwise identical?
    - Do they have the same sequence of chars?
    - Do they look the same if printed?
- **Meaning depends on PL and types of values**

Assignments

- **Assignment operator: Multiple possible meanings**
- **Deep vs. shallow**

- (Deep) copy of right-hand side to left-hand side's location
- Mostly used for primitive types
- Rare for objects, but useful, e.g., for remote procedure calls

- Copy reference to right-hand side to left-hand side
- Mostly used for objects

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

https://ilias3.uni-stuttgart.de/vote/0ZT9

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

https://ilias3.uni-stuttgart.de/vote/0ZT9

Overview

- **Introduction to Types and Type Systems**
- **Type Checking**
  - Type Equivalence
  - Type Compatibility
  - Formal Definition of Type Systems
  - Type Inference
- **Equality Testing and Assignments**

Overview

- **Introduction to Types and Type Systems**
- **Type Checking**
  - Type Equivalence
  - Type Compatibility
  - Formal Definition of Type Systems
  - Type Inference
- **Equality Testing and Assignments** ✔