Programming Paradigms

Type Systems (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2022

Quiz: Type Equivalence

Which of the following statements is true?

- Types are compatible if and only if they are equal.
- A type conversion always preserves the information stored in a value.
- Two types can be equivalent only if they share the same name and structure.
- With loose name equivalence, alias types are distinct types.

Quiz: Type Equivalence

Which of the following statements is true?

- ~~Types are compatible if and only if they are equal.~~
- ~~A type conversion always preserves the information stored in a value.~~
- ~~Two types can be equivalent only if they share the same name and structure.~~
- ~~With loose name equivalence, alias types are distinct types~~.

Overview

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

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
- For both options:

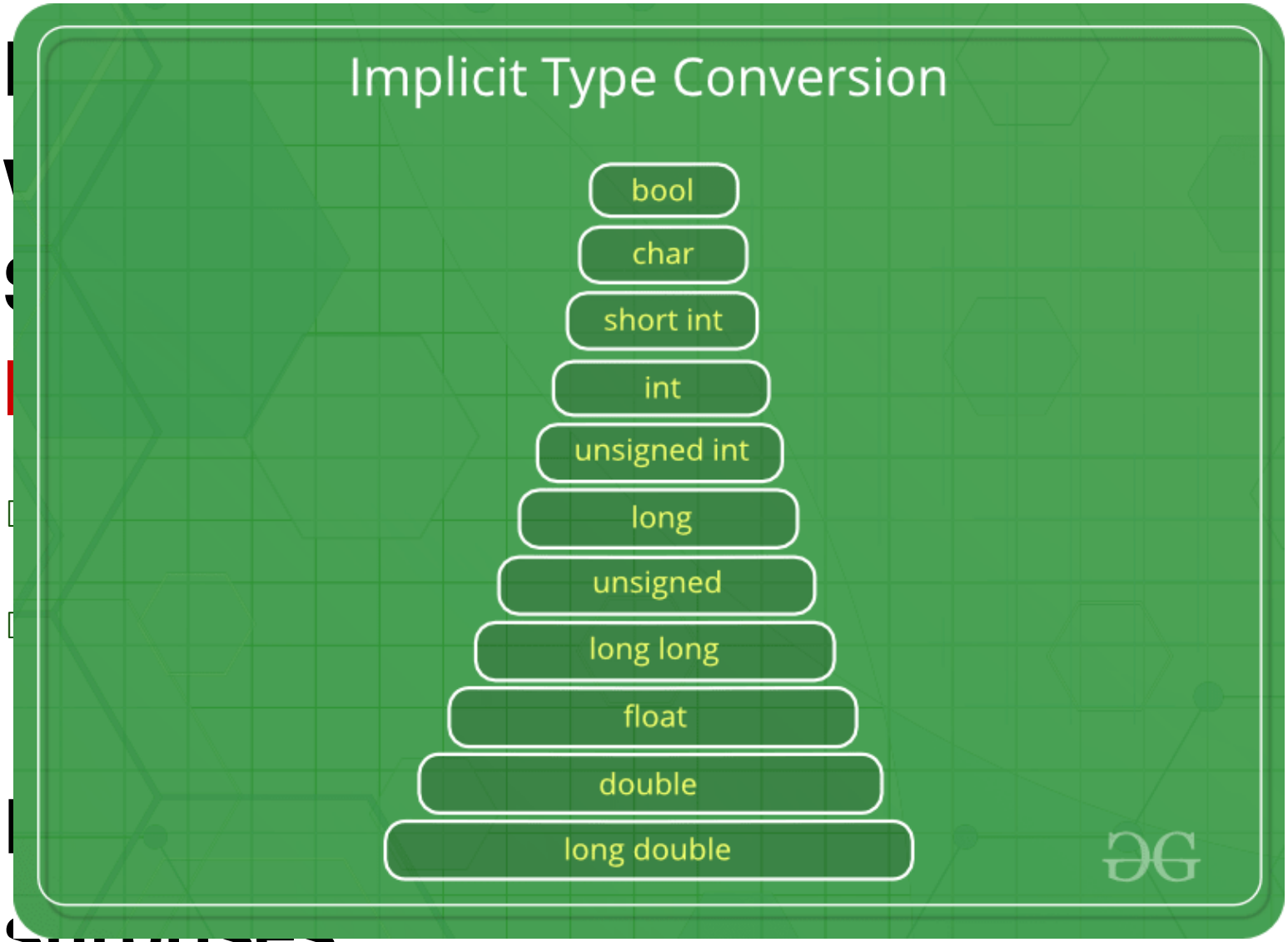
  Actual conversion happens at runtime

Coercions in C

- **Most primitive types are coerced whenever needed**
- **Some coercions cause information loss**
  - `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



Surprises

Source: *geeksforgeeks.org*

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**

Quiz: Coercions in C

What does the following C code print?

```
float d = 3.1416;
int l = d;
d = l;
char c = d;
bool b = c;

printf("d=%f, ", d);
printf("l=%d, ", l);
printf("c=%d, ", c);
printf("b=%d\n", b);
```

Quiz: Coercions in C

What does the following C code print?

```
float d = 3.141592;
int l = d;    // coercion to integer 3
d = l;        // fraction lost, d is 3.0
char c = d;   // fits into 8 bytes of a char
bool b = c;   // coercion to true

printf("d=%f, ", d);  // 3.000000
printf("l=%d, ", l);  // 3
printf("c=%d, ", c);  // 3
printf("b=%d\n", b);  // 1
```

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 $t : T$

"has type"

- **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 true,
then B is true

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

B is always true

Bool:

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

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

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

Nat:

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

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

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

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

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!

$$\dfrac{\dfrac{\dfrac{??}{0 : \text{Bool}} \quad \dfrac{??}{\text{true} : \text{Nat}} \quad \dfrac{..}{\text{pred false} : \text{Nat}}}{\text{if } 0 \text{ then true else (pred false)} : \text{Nat}}\ (\text{T-If})}{\text{succ (if } 0 \text{ then true else (pred false))} : \text{Nat}}\ (\text{T-Succ})$$

Quiz: Typing Derivation

Find the typing derivation for the following expression:

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

How many axioms and rules do you need to apply?

3 axioms, 4 rules

$$\dfrac{\dfrac{}{0 : \mathsf{Nat}}\,(\text{T-Zero})}{\text{pred } 0 : \mathsf{Nat}}\,(\text{T-Pred}) \qquad \dfrac{}{0 : \mathsf{Nat}}\,(\text{T-Zero})}{\dfrac{\dfrac{}{\text{true} : \mathsf{Bool}}\,(\text{T-True}) \qquad \dfrac{\text{pred } 0 : \mathsf{Nat}}{\text{succ (pred } 0) : \mathsf{Nat}}\,(\text{T-Succ}) \qquad \dfrac{0 : \mathsf{Nat}}{\text{succ } 0 : \mathsf{Nat}}\,(\text{T-Succ})}{\text{if true then (succ (pred 0)) else (succ 0)} : \mathsf{Nat}}\,(\text{T-If})}$$