Programming Paradigms

Type Systems (Part 4)

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
Type Equivalence

Prerequisite for type checking:
Clarify whether two types are equivalent

Two approaches

- Structural equivalence
  - Same structure means same type

- Name equivalence
  - Same type name means same type
Structural Equivalence

- Given two types, **compare** their structure recursively

- **Example:** Any class with
  - an int field called “age”,
  - a boolean field called “isRegistered”, and
  - a method called “printRecord”
Variation Across Languages

■ Do names matter?
  □ Same memory representation, but differently named
  □ E.g., different field names in a record

■ Does order matter?
  □ Different memory representation, but lossless reordering possible
  □ E.g., same fields but in different order
Examples (Pascal-like syntax)

\[ T_1 = \text{record}\ a: \text{integer}, \ b: \text{real} \ \text{end}; \]
\[ T_2 = \text{record}\ c: \text{integer}, \ d: \text{real} \ \text{end}; \]
\[ T_3 = \text{record}\ b: \text{real}, \ a: \text{integer} \ \text{end}; \]
\[ T = \text{record}\ \text{info}: \text{integer}, \ \text{next}: \ ^T \ \text{end}; \]
\[ U = \text{record}\ \text{info}: \text{integer}, \ \text{next}: \ ^V \ \text{end}; \]
\[ V = \text{record}\ \text{info}: \text{integer}, \ \text{next}: \ ^U \ \text{end}; \]
Limitation of Structural Equivalence

- Cannot distinguish different concepts that happen to be represented the same way
- Example:

  ```plaintext
type student = record
    name, address : string;
    age: integer
  end;
```

  vs.

  ```plaintext
type school = record
    name, address : string;
    age: integer
  end;
```
Limitation of Structural Equivalence

- Cannot distinguish different concepts that happen to be represented the same way

- Example:

```pascal
type student = record
  name, address : string;
  age: integer
end;

type school = record
  name, address : string;
  age: integer
end;

{ This is allowed: }
x : student; y : school;
x := y;
```
Name Equivalence

- Types with **different names** are different
- Assumption: Programmer wants it that way
- Used in many modern languages, e.g., Java
Limitations of Name Equivalence

- **Alias types** cause difficulties
- **Example:**

```
{ Here, we want both types to be the same }
type stack_element = integer;

{ Here, we want distinct types, to prevent mixed computations }
type celsius = real;
type fahrenheit = real;
```
Strict vs. Loose Name Equivalence

- Aliases are distinct types
  - `type A = B;` is a definition

- Aliases are equivalent types
  - `type A = B;` is a declaration
Type Conversion

- **Explicit conversion (cast)** of a value from one type to another
- Three cases
  - Types are **structurally equivalent**: Conversion is only conceptual, no code generated
  - 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

-- integer range 0...100
t : test-score

-- type alias for integer
c : celsius-temp

-- runtime, semantic check needed
t := test-score(n)

-- no check needed
n := integer(t)

-- runtime conversion
r := long-float(n)

-- runtime conversion & check
n := integer(r)

-- purely conceptual
n := integer(c)

-- purely conceptual
c := celsius-temp(n)
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.