CS4022D Principles of Programming Languages
Lecture #5: Type System

Saleena N
NIT Calicut

August 06, 2021

Type Systems

- Types
- Type System
- Type Checking
- Why Type Systems?

Types

```
int a, b, c;

int f ( int x);
```

- Types of the identifiers ?

Types ?

Types

- Set of values ?
  - type `int`: set of integers
  - type `real`: set of real numbers
  - type `boolean`: {`true`, `false`}

Classification of values

Types

Set of values

In a = b + c, what does + denote?

Types

- set of operations on the values
  - type `int`: integer arithmetic operations `+` `-` `*` `%`
  - type `real`: real number arithmetic operations
  - type `boolean`: `and`, `or`, `not` ...

Types

- set of values
- set of operations on the values
  - type `int`: set of integers (finite?) and the integer arithmetic operations `+` `-` `*` `%`
  - type `boolean`: values `true`, `false` with operations `and`, `or`, `not` ...

History

Type Systems for Efficiency: The first type systems were introduced in languages like FORTRAN to improve the efficiency of numerical calculations by distinguishing between integer valued and real valued arithmetic expressions. The compiler can then use different representations and generate appropriate machine instructions for primitive machine operations [1]

[1] Benjamin C Pierce *Types and Programming Languages*, PHI, 2004

Types

- Basic Types - int, char
- Composite Types - arrays, records
- Programmer Defined types

Type Checks

```
a = b+c;

y = f(a);
```

- Type checks during compilation ?

Type Checks

Type of expression `b+c`?

Type of `f(a)`?

Type of `f`?

Type Checks

Type of expression $e_1 + e_2$?

Based on the typing rules of the language.

Type of

- Variable?
- Expression?
- Function?
- Statement?

Type checking

- $x = f(y)$
  - Type of f ?
  - Type of $f(y)$?
  - Type of $x = f(y)$?

Types

- Types of constructs - calculated compositionally, with the type of an expression depending only on the types of its sub expressions
- type of $y + z$ depends on the type of y and z.

Type System

- Set of types in the language
- Set of rules for assigning types to various constructs

Typing Rules

- The type of an expression e of the form $e_1 + e_2$
  - if both e_1 and e_2 are integers, then type of e is integer
  - if both e_1 and e_2 are real, then type of e is real
  - if e_1 is integer and e_2 is real ?
- Rules for type conversion

Type System

- Rules for type conversion
- if e_1 is integer and e_2 is real, then type of e is real
- Type Conversion - implicit / explicit
- Casting?
- Compatible type

Type checking

- ensures that a program obeys the language's type compatibility rules.
  - the components of an expression are of appropriate type
  - an operator is applied to operands of compatible types
  - prevents addition of a record and an integer
  - prevents passing a file as argument to a function that computes the square root of a number
- Static / Dynamic checks

Type Checker

- Type checkers are built into compilers
- Static (compile time)checks - Statically typed.
- Uses the annotation provided by the programmer (explicitly typed)
- No type annotations - implicitly typed as in ML

Subtyping

- T_1 is a subtype of T_2 ?
- Class hierarchy in OO languages
- **ElectiveCourse** is a subclass of **Course** ?

Type Systems for[2]

- Early detection of certain programming errors
- Abstraction - class, interface of class
- Documentation
- Language Safety - static and dynamic checks
- Efficiency

[2] Benjamin C Pierce *Types and Programming Languages*, PHI, 2004