Introduction to Types

Professor: Suman Saha

Types

- A type is a collection of values that share some structural property and operations on those values.
- Examples
  - Integer type has values ..., -2, -1, 0, 1, 2, ... and operations +, -, *, /, <, ...
  - Boolean type has values true and false and operations $\wedge$, $\vee$, $\neg$.
  - int $\rightarrow$ bool is a set of functions that takes ints and returns booleans; operations: function invocation (application)

Types

- Non-examples
  - {3, true, 3.5}
- Distinction between sets that are types and sets that are not types is language dependent
  - e.g., Pascal allows range types: 1..100, ‘0’..’9’

Use for Types

- Program organization
  - Separate types for separate concepts
    - E.g., one class for courses; another class for students; …
- Formal documentation
  - Indicate intended use of declared identifiers
  - Types are checked by compilers, unlike program comments
- Support optimization
  - Example: short integers require fewer bits
  - Access record components by known offsets
- Identify and prevent errors
  - Compile-time or run-time checking can prevent meaningless computations such as 3 + true - “Bill”

Def. of Type Errors

- A type error is any error that arises because an operation is attempted on values of a data type for which it is undefined
  - e.g., adding an integer to a floating number
  - e.g., 3 + ‘hello’
  - e.g., invoke a function that needs two arguments with just one argument
  - e.g., accessing an array out of bound
- High level languages reduce the number of type errors via a type system

Static vs. Dynamic Typing

- A type system imposes constraints on programs to rule out type errors
- Static typing
  - Types of names (variables, functions, ...) are declared statically
  - Perform type checking at compile time
  - Example PLs: C, most of Java
  - E.g., In Java, o.f(x)
    - o must have some class C
    - C.f must exist
    - C.f must have type A->B
    - x is of type A

Static vs. Dynamic Typing

- Dynamic typing
  - Types of names (variables) can change during runtime, depending on the values assigned
    - Python: x = 3; x = [1, 4, 5]
  - Perform type checking at run time
    - Values need to carry type tags for type checking
  - E.g., Scheme, (car x) checks that x is a list and x has at least one element
  - Example PLs: Lisp, Scheme, Python, Perl, Ruby, PHP
- Still others (e.g., Java) do both
  - Upcasts always allowed; downcasts checked during runtime

Static vs. Dynamic Typing

- Basic tradeoff
  - Both prevent type errors
  - Dynamic typing slows down execution
    - Need more memory for representing type tags
    - Errors are identified at a later time
  - Static typing restricts program flexibility
    - Lisp (dynamically typed) lists: elements can have different types; ML (statically typed) lists: all elements must have the same type

Strongly vs Weakly Typed

- A language is *strongly typed* if its type system allows all type errors in a program to be detected either at compile time or at run time
- A strongly typed language can be either statically or dynamically typed.
- Type checking may miss type errors in weakly typed languages

Relative Type-Safety of Languages

- Not safe: BCPL family, including C and C++
  - Unsafe features: type casts, pointer arithmetic, union types, …
- Almost safe: Algol family, Pascal, Ada.
  - Unsafe feature: dangling pointers
    - Allocate a pointer p to a mem region, deallocate the memory referenced by p, then later use the value pointed to by p
- Safe: Lisp, ML, Smalltalk, and Java
  - Lisp, Smalltalk: dynamically typed
  - ML: statically typed
  - They use garbage collection

Lisp/Scheme is Dynamically Typed

- Lisp/Scheme:
  - No declaration of types (e.g., in the square function)
  - Check types dynamically
    - run the program first, without type checking
    - only if there is a type error during evaluation, an error will be reported
- Adding an integer to a boolean
  - (define f (lambda (x) (+ 2 #t))
  - (define (f x) (if (< x 10) 3 (square #t)))
    - syntactically correct
    - but will cause a dynamic error
  - (define (f x y) (if (> x 10) x (+ x y)))
    - (f 12 #t)
      - a static type system would reject this program
    - what about (f 9 #t)?

A Type System Has Rules For

- Type equivalence
  - when are the two types the same?
  - This determines when assignment can happen in statically typed languages
- Type compatibility
  - when can a value of type A be used in a context that expects type B?
- Type inference
  - what is the type of an expression, given the types of the operands?
- Type Safety (Strongly typed)
  - Absence of type error

Type Equivalence (In C)

```
struct complex {
    float re, im;
};
struct polar {
    float x, y;
};
struct {
    float re, im;
} a, b;
```

```
struct complex c, d;
struct polar e;

c.re = 1.0; c.im=2.0;
d = c

// what are equivalent types?
// C uses name equivalence for
   structs; c and d are of the
   same type;  a and c are not;
   d and e are not
```

Two notions of type equivalence

- Name equivalence
  - two types are the same if they have the same name
  - c,d are of the same type; a and c are not; d and e are not
  - Java uses name equivalence for classes
- Structural equivalence
  - two types are the same if they have the same structure (that is, they have the same components)
  - a,b,c,d would have the same type; d and e would have different types

C's Type Equivalence

- C uses name equivalence for structs and unions and structural equivalence for everything else (arrays, pointers, ...)
- example:

```
typedef float METERS;
typedef float FEET;        // same "structure" as above
METERS area;
FEET length;
area = length * length;   // this is legal in C!
```

Built-in Types vs. User-Defined Types

- In early languages, Fortran, Algol, Cobol, all of the types are built in.
- If needed a type color, could use integers; but what does it mean to multiply two colors.

- Purpose of types in programming languages is to provide ways of effectively *modeling* a problem solution.