CS 461

Programming Language Concepts

Gang Tan
Computer Science and Engineering
Penn State University

Where do we stand?

- We have been comparing FP with non-FP?
  - Higher-order functions
  - Imperative vs functional programming
- Next
  - Static vs dynamic typing
  - Memory management

Ch7 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)
- 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'

Uses 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
- Interpret low-level data
- Identify and prevent errors
  - Compile-time or run-time checking can prevent meaningless computations such as 3 + true - "Bill"

Types Essential for Interpreting Low-Level Data

- Machine data carries no type information.
  - Basically, just a sequence of bits.
- E.g., 0100 0000 0101 1000 0000 0000 0000 0000
  - The 32-bit integer 1,079,508,992
  - Two 16-bit integers 16472 and 0
  - Four ASCII characters: @ X NUL NUL
  - The floating point number 3.375
  - Or a pointer (e.g., a function address)
  - Or even interpreted as one or a few machine instructions

Type Error Examples

- Cause hardware errors
  - Jump to code address that does not contain a legal opcode
  - Call x() where x is not a function
- Unintended semantics
  - int_add(3, 3.375)
    - Does not cause a hardware error, since the bit pattern of float 3.375 can be interpreted as an integer
    - Silent error, even more harmful that those that cause hardware errors

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
  - In Java, upcasts always allowed; downcasts checked during runtime; wild casts always disallowed

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

Type Safety (Strong Typing)

- 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.

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
  - No language with explicit deallocation of memory is fully type-safe
- 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)?

Benefits of Dynamic Typing

- Heterogeneous lists
  - '(a, 7, b)
  - Statically typed languages have to give a type to such a list