Types

Lecture 13

Types

- What is a type?
  - The notion varies from language to language
- Consensus
  - A set of values
  - A set of operations on those values

- Classes in an object-oriented language are an elaboration of this notion to include relations among types and operations (class inheritance, method inheritance)

Types

- Often broken down into
  - Elementary
  - Compound

  - Properties
    - Set of possible values
    - Set of operations (arithmetic? Assignment?, display?)
    - Relations with other types
    - Representation (location, stack, heap..)
    - Constraints on use
    - Scope of definition
    - Coercions to other types

Typical built-in types

- Boolean
- Fixed precision integer (bit, byte, word; signed or not)
- Single Float IEEE 754 standard 32-bit
- Double Float IEEE 754 standard 64-bit
- Complex (Single, Double) Float
- Character (8 bit, maybe 16 bit Unicode)
- String
- Reference (address, pointer)
- File pointer
- Function (?) as a function address (but see, closures)
- Lisps usually have rationals, arbitrary prec. nums (bignums)
- Extended floats (usually extended-double 64+16)
- Decimal arithmetic IEEE 854? std

Typical compound construction

- Arrays (indexed sequence, uniform types)
- Record, Struct, defstruct (not uniform)
- Union (discriminated or not)
- Class or Object (CLOS in common lisp)
- Encapsulated function or closure (well, maybe not typical, but available in functional languages like Lisp, Scheme)

MJ types are rather impoverished, in case you hadn't noticed

- Integer, boolean are the only primitive types
- Array, class constructors
- How hard would it be to add strings, floats to MJ?
  - Data type formats are pretty easy
  - Remember, associated OPERATIONS matter

Why Do We Need Type Systems?

Consider the assembly language fragment

add $r1, $r2, $r3

Meaning c(r2)+c(r3) → r1 perhaps.

What are the types of $r1, $r2, $r3?

Types and Operations

- Certain operations are legal for values of each type
  - It doesn't make sense to add a function pointer and an integer in C
  - It does make sense to add two integers
  - It might make sense to add an integer and a pointer (this is what an array reference might be in some languages)
  - But all have the same assembly language implementation!

Type Systems

- A language's type system specifies which operations are valid for which types
- The goal of type checking is to ensure that operations are used with the correct types
  - Enforces intended interpretation of values, because nothing that comes later, in a conventional compiled language, will check!
  - Note that in some architectures the op-code "add" might look at the run-time tags of the arguments and do add integer, add double, add single. Not common (though there are existing machines that do this)
- Type systems provide a concise formalization of the semantic checking rules

What Can Types do For Us?

- Can detect certain kinds of errors
- Memory errors:
  - Reading from an invalid pointer, etc.
- Violation of abstraction boundaries, trying to alter an internal variable in an object that is not public.
- Improve quality of compiled code

Type Checking : where to find them..

- Scope of types follows scope of language:
  - *Statically scoped/typed*: All or almost all checking of types is done, or could be done, as part of compilation (C, Java, compiled CL)
  - *Dynamically scoped/typed*: Almost all checking of types is done as part of program execution. Wasteful to repeat checking if the type of a datum does not change. A good idea if programs operate over many types. (interpreted or compiled but undeclared CL, Scheme)
  - *Advisory typed*: (essentially only in CL.)
  - *Untyped*: No type checking (machine code)

Named vs structure type agreement

- Let function foo(x:array of int):array of int=x
  --no names, same type ..,
- Let type t = array of int
  s= array of int
  function foo(x:s):t=x

--different names, same type

Compare to (eq '(a b) '(a b)) vs (equal '(a b) '(a b)) in lisp

Types and subtypes in Common Lisp

- number, real, integer, float, single, double, complex, byte,
- type expressions (integer 0 255), (or integer single) (type1 (type2 type3)) {function spec}
- Much of this replaced by CLOS
- In its object-oriented system, a type specification may be of supertype; the actual parameter may be of a subtype.
-

Types and subtypes in Common Lisp

- new types defined by defstruct and by CLOS
- Defgeneric defines new functions by merging name + function signature (types of arguments) + inheritance

Defgeneric in Common Lisp

- defgeneric foo ((x fixnum) (y fixnum) ….),
- defgeneric foo ((x fixnum) (y double) ….),
- defgeneric foo ((x double) (y fixnum) ….),
- defgeneric foo ((x number) (y number) ….),
- etc uses the most specific foo
- Redefines all the foo programs any time another defgeneric of foo

Compiling/ inferring types in Common Lisp

- (defun square(x)(* x x))
- Vs something like..

```
(defun square(x)
 (typecase x
      (fixnum (* x x))
      (double-float (* x x))
      (single-float (* x x))
      (otherwise (* x x))))  ;; bignum, complex, rational…
```

The Type Wars

- Competing views on static vs. dynamic vs no typing
- Static typing proponents say:
  - Static checking catches many programming errors at compile time
  - Avoids overhead of runtime type checks
- Dynamic (or no)typing proponents say:
  - Static type systems require saying many things twice or more, creating programming errors
  - Rapid prototyping easier in a dynamic type system

The Type Wars (Cont.)

- In practice, many statically typed languages fail to fill all needs and have some "escape" mechanism
  - Unsafe casts in C, Java, Fortran, PL/I…
- Some languages try to infer types with minimal or even no hints (ML). This seems to be the best of both worlds, but the need to be able to infer types constrains programs.

…. Lisp compilers can infer types sometimes, but generally need nudges to work.

The Type Wars (Cont.)

- In 'real world':
  - 10-100 people write modules, interfaces, etc then debug,document,test,debug,test,test,test, beta release, v1.0, v1.1, … strong typing can prop up the weak, find certain mistakes without testing all paths
- In the student environment:
  - 1-2 people write, debug, debug, debug, if it works once turn it in. done… prototyping tools rewards the clever
  - (comment from Paul Graham re beating the averages) http://paulgraham.com/avg.html