2004 Programming Languages Comp. Solutions

written by 2006 Ph.D. First-Years

1. *Type inference:* `'a list -> int`

2. *Scoping:* Static scope is resolved at compile-time (use the value of the variable declared in the closest enclosing block); dynamic scope is resolved at execution-time (use the value of the variable closest to the current function's activation record on the stack). C++ is a language that uses static scope. Exceptions are a language feature that uses dynamic scope.

3. *Typing and Garbage Collection:*

   (a) This technique is solving the problem of tags needing to be kept alongside pointers in order to make precise garbage collection work. In order for garbage collection to be precise, the garbage collector must know which blocks of memory are actually pointers so that it can follow pointers to access other reachable objects. Keeping these tables around obviates the need to tag each pointer at run-time, thus giving space savings.

   (b) This will work better for ML because it has static types. Lisp and Scheme are dynamically-typed languages, so there is no way to build up a compile-time type table.

   (c) Yes, this is helpful for C programs because you can know which memory contents are actually pointers, instead of having to make conservative guesses (that any sufficiently large integer value that refers to some valid address range on the heap is a pointer). However, it does not solve the problem of pointers to the middle of objects, casts of integers to pointers (and vice versa), and the use of `void*` pointers to emulate object-oriented functionality.

   (d) No, it won't work with polymorphism because a variable can have more than one type at run-time, so there is no way to build up a table at compile-time that maps a variable to one type.

4. *Method Lookup:*

   (a) It is found via a lookup in a method dictionary where the dictionary for `XClass` contains an entry that maps the method name `m` to the address of the first instruction of that method (otherwise, its superclass should contain an entry for `m`). The dictionary for each class also contains a pointer to the dictionary of its superclass, so that if an entry isn't found, then the superclass's dictionary is searched, and so on until either the method is found or an exception results.

   (b) The main data structure is a `vtable`, which is an array containing addresses of the first instruction in each method. It is used by having the compiler hardcode in a fixed constant offset from the beginning of the `vtable` so that overridden methods in a subclass have the same offset as their respective methods in the superclass.

(c) The C++ `vtable` approach is more efficient because a direct array lookup is very fast, whereas the Java method dictionary method is more flexible because it can support dynamic class loading, etc. The `vtable` approach is more appropriate for C++ because performance is paramount for C++ programs, but for Java programs, flexibility and extensibility are more important.