Programming Languages and Compilers Comprehensive Exam

May 1997

Please answer each question in a separate booklet.

1. **Enter your answer in the booklet labeled 1.**

   (a) Why are higher order functions a desirable property in a programming language? What are the drawbacks of including higher order functions in a language?

   (b) Write ML functions that implement the operators $\sum$ (sum), $\prod$ (product), and $\circ$ (compose).

   (c) Write one ML function `F` such that $\sum$ and $\prod$ can be defined as values returned by `F`. Show how $\sum$ and $\prod$ are defined using `F`.

   (d) In ML, formal parameters must be used monomorphically. Thus, the definition

   ```
   fun f g = g 1 + g true
   ```

   is type-incorrect. Suggest an extension to ML that would allow formal parameters to be used polymorphically. You may require the programmer to supply additional declarations if necessary. Once you have done this, describe how ML's type checking algorithm would have to be modified to handle your extension.

2. **Enter your answer in the booklet labeled 2.**

   (a) Given a directed acyclic graph (DAG) represented as a set of nodes and a set of edges (each edge is represented as a pair of nodes), write the SETL function that performs a topological sort. It should return a tuple of the nodes.

   (b) Write a function `Perm(S)` in SETL that computes the set of all permutations of the elements of set `S`. For example, if `S` = $\{$`1,2,3`$\}$, then `Perm(S)` returns

   ```
   {[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,2,1],[3,1,2]}
   ```

   (c) Write a function `AllTop(n,e)`, where `n` and `e` are the nodes and edges of a DAG, that uses `Perm` to construct the set of all possible permutations of the nodes and then returns the set of all such permutations that represent a topological sorting of the nodes.

3. **Enter your answer in the booklet labeled 3.**

   (a) Give an example of an LR(0) item. Then, give an intuitive explanation of what an LR(0) item corresponds to during parsing.

   (b) Give the algorithm for computing $closure(I)$, where I is a set of LR(0) items. Intuitively, what does the closure of a set of LR(0) items represent?

   (c) Define the function $goto(I, X)$, where I is a set of LR(0) items and X is a grammar symbol. Then, give the algorithm for constructing the canonical collection of sets of LR(0) items.

   (d) Describe *informally* how, once the canonical collection of sets of LR(0) items has been computed, an SLR parse table is constructed.

4. **Enter your answer in the booklet labeled 4.**

   (a) One of the obstacles to implementing efficient register allocation in C is `&`, the address-of operator. Explain why this is so and give an example in C that illustrates the problem.

(b) Register allocation can be modeled using graph coloring. The graph coloring problem is: Given an undirected graph and some integer k, determine if k colors are sufficient to assign each node a color such that no neighboring nodes (i.e. nodes that share an edge) have the same color.

Explain the correspondence between register allocation and graph coloring. What do the nodes and edges in the graph correspond to? In what situation is spill code (i.e. code to store register values in memory) necessary?

(c) The graph coloring problem is known to be NP-complete. Suggest a heuristic that colors the graph in polynomial time, and describe the actions necessary (and their corresponding actions by the register allocator) if the coloring fails.

5. **Enter your answer in the booklet labeled 5.**

Answer **<u>either</u>** part a **<u>or</u>** part b, **<u>not</u> <u>both</u>**.

(a) i. Give the algorithm for CPS conversion of Scheme expressions (assume only DEFINE, IF, LET, and LAMBDA expressions along with application and atomic expressions in the source language).

ii. Describe the advantages of performing CPS conversion during compilation.

iii. If, during the execution of a program, you were able to watch the stack and the heap as execution progressed, could you tell whether the compiler had performed CPS conversion or not? If your answer is yes, give an example of a Scheme program for which a CPS-based compiler would generate code that caused different stack and heap behavior than the code generated by a direct-style compiler (assume both compilers perform the same optimizations and use the same memory allocation strategy). If your answer is no, sketch a formal proof that the stack and heap behavior would be the same.

(b) i. In the context of a flow graph, what does it mean for a node N to dominate a node M? Again in the context of a flow graph, give a formal definition of a *back edge* and a *loop*.

ii. A *reducible* flow graph is a flow graph that, when all back edges are removed, is acyclic. Write a short program, in any language you like, whose flow graph representation would be irreducible (i.e. not reducible), and draw the flow graph.

iii. If possible, translate your program from part ii above into a language P whose control flow constructs consist only of while loops, if-then-else statements, and procedure calls. If it is not possible to translate your program from part ii into P, then write a new program in P whose flow graph would be irreducible. If even that is not possible, prove formally why you can't write such a program in P.

6. **Enter your answer in the booklet labeled 6.**

(a) What is the default parameter passing mechanism in Algol60? Give a precise formal definition of the meaning of an Algol procedure call (not its implementation).

(b) Write a short program fragment that would execute in linear time using any other popular parameter passing mechanism but runs in quadratic time using Algol's default parameter passing mechanism.

(c) Suppose you are writing an Ada compiler in which you want to exploit the freedom to choose a parameter passing mechanism as long as the parameter mode, `in`, `out`, or `in out`, is supported. The choice of the mechanisms used for a given procedure call `f(x1,...,xn)` would depend on certain features of `f` and each `xi`. As formally as possible, quantify these features and give the equations that your compiler would have to solve in order to make the best choice.