Programming Languages and Compilers
Comprehensive Exam

September 1996

Please answer each question in a separate booklet.

1. **Enter your answer in the booklet labelled with this question number**.

   Tasks in Ada 83 provide concurrency. The type of a task determines the code executed by the task. Consider the following statement:

   > Tasks in Ada 83 are too constraining. It is not possible to construct an array of tasks, for example, if the tasks execute different code. The type of a task should be a function of its entries, but not of its code body.

   Give the arguments for and against this statement. Specifically, consider the following questions:

   (a) To what extent is this statement correct? Is it possible to simulate such an array?

   (b) Suppose thus restriction were relaxed. What would be the consequences of this change? Consider (i) flexibility and (ii) efficiency.

   (c) Why do you think the Ada 83 designers chose the approach they did?

2. **Enter your answer in the booklet labelled with this question number**.

   In Pascal, the statement

   `goto` *label*

   transfers control to the statement preceded with *label*. Consider an extension of Pascal in which label variables are allowed, there is a statement to assign labels to label variables, a goto is allowed to reference any label variable which is in scope, and the lifetime of values of label variables is unbounded.

   Compare the effect of this extension with that of adding first-class functions (i.e. closures) to Pascal. Specifically:

   (a) can this extension be used to simulate first-class functions? If so, how?

   (b) can this extension provide capabilities not provided by first-class functions? If so, what?

   (c) What are the implications of this extension on the implementation.

3. **Enter your answer in the booklet labelled with this question number**.

   (a) Give a grammar for which an SLR parse table can be constructed. (No need to show the table).

   (b) Give a grammar for which an SLR parse table cannot be constructed, but for which an LALR parse table can be.

   (c) What is the advantage of using an LALR parse table rather than a canonical LR parse table? Explain fully why this advantage arises (using a tiny example if necessary).

Exam continued on back of this page

4. **Enter your answer in the booklet labelled with this question number**.

   (a) Define the term *available expression* (in the context of program optimization), and state why one would want to compute the set of available expressions at some program point.

   (b) Give the dataflow equations used to compute available expressions on entry to, and exit from, basic blocks.

   (c) State how these equations are different for those used to compute reaching definitions.

5. **Enter your answer in the booklet labelled with this question number**.

   Consider the following Prolog definition:

```
cc([H|T], L, [H|R]) :- cc(T,L,R).
cc([],L,L).
```

   (a) Trace the execution of:

```
:- cc([1,2], [3], L).
```

   i.e. list the successive predicates and their solutions.

   (b) Show how this predicate can be used to:

      i. determine the last three elements of a list L;

      ii. test if list L1 is a prefix of list L2;

   (c) Show how this predicate can be used to generate:

      i. all sublists of a list L, sequentially;

      ii. a list containing all unique sublists of a list L;

   (d) Compare the above definition with the following:

```
qcc([H|T], L, [H|R] :- !, qcc(T,L,R).
qcc([],L,L).
```

6. **Enter your answer in the booklet labelled with this question number**.

   A *copying garbage collector* works by keeping two heaps, only one of which is in use at any time. When the heap in use is full, the garbage collector traverses the graph (of live data) copying each object it encounters into the other heap. When all objects have been copied, execution resumes using the heap into which the live objects had been copied.

   (a) What is the cost of copying garbage collection, compared to the traditional mark-sweep collection?

   (b) During the traversal of the graph, the collector might encounter an object that has already been copied to the other heap. Does the collector need to know the new location of the copied object? If so, state why and suggest how the collector could determine what the new location is. If not, state why not.

   (c) In its most naive implementation, a copying collector – like a mark sweep collector – uses a stack to implement a traversal of the graph. Since this requires setting aside stack space, it would be nice to avoid this. Suggest a method for avoiding the use of such a stack altogether (hint: find a way of keeping track of objects that have already been traversed but whose children have not).