Programming Languages and Compilers Comprehensive Exam

September 1999

Please answer each question in a separate booklet.

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

   In Prolog, the standard predicate `member(X,L)`, is defined in the following two clauses:

   ```
   member(X,[X|_]).
   member(X,[_|L]) :- member(X,L).
   ```

   (We will use upper case symbols for variables and lower case for atoms.)

   The predicate `member(X,L)` can be used to check whether `X` is an element of list `L`. For example, the query

   ```
   member(a,[b,a,b,a])
   ```

   succeeds.

   If the first argument is left uninstantiated, then it can be used to iterate through a loop. For example, the query

   ```
   member(X,[a,b,a,c,b])
   ```

   first succeeds with the binding `X=a;`, next, on backtracking, it succeeds with the binding `X=b;` next, with `X=a;` next, with `X=c;` next, with `X=b;` and after that it fails.

   If the second argument is left uninstantiated, then it can be used to generate generic lists in which the first argument appears at greater and greater depth. For example, the query `member(a,L)` first succeeds with the binding `L = [a | _101];` then on backtracking it succeeds with the binding `L = [_102, a | _103];` then with the binding `L = [_104, _105, a | _106];` and so on.

   We will also define two other predicates. The predicate `mem2(X,L)` has the identical definition as `member` and therefore the identical behavior:

   ```
   mem2(X,[X|_]).
   mem2(X,[_|L]) :-  mem2(X,L).
   ```

   The predicate `p(X,L,M)` is defined by a single clause

   ```
   p(X,L,M) :- member(X,L), mem2(X,M).
   ```

   (a) What bindings are returned in successive backtrackings to the query

   ```
   p(Q, [a,b,c,a,b], [c,b,d,c]).
   ```

   (b) List the predicate invocations that are on the activation stack at the time that the query in (a) first succeeds. You need only list the name of the predicates, from bottom to top of the stack. (e.g. `p; mem2; p; member` is the correct form of answer though not, of course, the correct answer.)

   (c) Give the first 5 bindings returned in successive backtrackings to the query

   ```
   p(Q, S, [a,b,c]).
   ```

2. **Enter your answer in the booklet labeled ??.** In the lambda calculus, β-reduction is defined as:

$$(\lambda x.E)M \Rightarrow E[M/x]$$

   (a) Describe in words what this means.

   (b) Give a formal definition of $E[M/x]$, for any expressions E and M.

   (c) In which reduction rule (α, β, η, or δ), if any, is the choice of applicative order evaluation vs. normal order evaluation made explicit? Explain.

   (d) One can view the purely-functional core of ML as syntactic sugar for a typed lambda calculus. Write the type rule for β-reduction in a typed lambda calculus that reflects ML's type system.

   (e) What, precisely, does it mean for an expression in a typed lambda calculus to be well-typed? Write an expression that is valid in the untyped lambda calculus that is not a valid expression in a typed lambda calculus.

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

   One method for approximating the area under a curve (i.e. the definite integral) is the *trapezoidal rule*, in which the result is the sum of the areas of trapezoids under the curve, as pictured below. Note that the area of a single trapezoid under the curve defined by the function f for the interval $[x, x+\delta]$ is $((f(x) + f(x+\delta)) \times \delta)/2$.

   (a) In Ada, write the code to compute the area under a curve using the trapezoidal rule. This code should be parameterized by the function f (which will simply be an Ada function) defining the curve, the endpoints of the curve, and the value of δ (the width of each trapezoid). This parameterization will allow the code to be reused to perform the computation for many different functions f.

   (b) Write the Ada code to approximate the area under the curve defined by the function

$$f(x) = x^3 + 2x^2 + 3x + 1$$

   with endpoints $x = 0.0$ and $x = 10.0$, using the trapezoidal rule with $\delta = 0.5$.

   (c) Write, in ML this time, code that performs the same function as in part (a).

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

   (a) In the context of bottom-up parsing, here is a partial definition of the term *handle*:

   > *The handle of a string s of terminals and non-terminals is a substring s' of s that matches the right hand side of a production in the grammar.*

   This definition is not sufficient. Give the full definition.

   (b) To give an example of why the above partial definition is not sufficient, give a very simple grammar, a string s of terminals and non-terminals that can be produced by the grammar, and a substring s' of s that conforms to the above partial definition but is not, in fact, a handle. Explain why s' is not a handle.

   (c) Shift-reduce parsing is an implementation of *handle pruning*. Describe precisely what handle pruning is (but don't describe shift-reduce parsing).

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

   (a) In the context of compiler optimization, define the term *induction variable*.

   (b) Give an algorithm for finding induction variables. Be sure to specify which program analyses should already have been performed before your algorithm is used.

   (c) Once induction variables are found, what optimization can be performed? Given an example of such an optimization by showing a program fragment (in 3 address code) before and after the optimization.

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

   (a) Describe the relative advantages and disadvantages having the caller save registers vs. having the callee save registers during a procedure call.

   (b) Invent a calling sequence for a simple block-structured imperative language (in which one cannot pass procedures as parameters), for a compiler that uses a single global display for supporting non-local variable references. Write out, in pseudo-code, the code that must be executed by the caller upon procedure call and procedure return as well as the code that must be executed by the callee upon entry to the procedure and exit from the procedure.

   (c) Suppose you have the following statement in a program written in this simple language,

   ```
   a[x] := b + c[y];
   ```

   where `a`, `b`, `c`, `x`, and `y` are all non-local variables. How many memory references are needed to execute this statement? (Don't count operating system overhead such as page-table access, of course). Justify your answer by listing each memory reference.