Programming Languages and Compilers Comprehensive Exam

May 1998

Please answer each question in a separate booklet.

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

   Tail recursive is the term applied to a recursive function whose recursive call is the last operation of the function, as in

   ```
   (define (f x) (if (= x 0) 1 (f (- x 1))))
   ```

   Tail recursive functions can be compiled into loops. For example, the above function could be translated into the intermediate form:

   ```
   Proc  f(x):
     top:  cmp x,0          ; compare x to 0
           jeq out          ; if they're equal, jump to "out"
           dec  x           ; decrement x
           jmp  top         ; jump back to top of Loop
      out: ret 1            ; return 1
   ```

   (a) What is the advantage of translating a tail recursive function into a non-recursive function containing a loop?

   (b) Notice that the usual recursive factorial function

   ```
   (define (fac x) (if (= x 0) 1 (* x (fac (- x 1)))))
   ```

   is not tail-recursive. Translate `fac` into a scheme procedure `tail-fac` that is tail recursive.

   (c) Given a subset of Scheme with the syntax defined by:

   ```
   <proc_def> ::=  (define (<id> <arg>*) <exp>)

   <exp> ::= <number> | <id> |
             (lambda (<id> <arg>*) <exp>) |
             (if <exp> <exp> <exp>) |
             (<op> <arg>*) |
             (let ((<id> <exp>) (<id> <exp>)*) <exp>) |
             (<exp> <exp>*)

   <arg> ::= <id>
   ```

   Write in Scheme a procedure `(tail?  def)` for determining if `def` is a tail-recursive procedure definition in the above subset of Scheme. Assume that `def` is syntactically correct, so no error checking is necessary. For example,

   ```
   >(tail? '(define (f x) (if (= x 0) 1 (f (- x 1)))))
   #t

   >(tail? '(define (fac x) (if (= x 0) 1 (* x (fac (- x 1))))))
   ()
   ```

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

   (a) What is the primary difference between Ada's generics and C++'s templates? Note: The answer is not "C++ is object-oriented but Ada isn't".

   (b) What is the advantage of Ada's choice with respect to this difference? What is the advantage of C++'s choice? Again, your answers here should be completely independent of the fact that C++ is object-oriented and Ada isn't.

   (c) Which feature of ML is most similar to Ada's generic package facility? Write a simple Ada generic package and write the corresponding code in ML.

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

   (a) The built-in SETL infix operator n `npow` s returns the set of all subsets of set s that have n elements. Write your own procedure `npower(N,S)` that accomplishes the same thing and uses the powerset operator, `pow`.

   (b) Now write this procedure without using the `pow` operator.

   (c) What computational savings is achieved by using your answer to (b) rather than (a)?

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

   (a) Construct a non-deterministic finite automaton (NFA) for recognizing $(a|bc)^*abc$.

   (b) Transform the NFA you constructed above to a deterministic finite automaton (DFA) that recognizes $(a|bc)^*abc$.

   (c) Give the algorithm for translating an NFA into a DFA that recognizes the same language.

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

   (a) Consider the following expression:

   ```
   (a + b) * (c - (d + e))
   ```

   Assuming that a machine has only register-to-register operations other than `LOAD` and `STORE`, what is the minium number of registers needed to evaluate this expression. Assume the variables are initially in memory.

   (b) Give an algorithm for computing the minimum number of registers required to evaluate an arithmetic expression. Assume the expression is represented as an abstrat syntax tree (AST) and is really a tree (not a DAG).

   (c) Describe how code generation would proceed for arithmetic expressions (again, assuming they are represented by trees not DAGS) in order to use the minimum number of registers.

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

   Any program analysis that traces information through procedure calls is an *interprocedural* program analysis.

   (a) Name an interprocedural analysis that is useful for program optimization. Give a simple example of a piece of code before and after an optimization that uses that analysis.

   (b) Of the following languages, which is the easiest (i.e. cheapest computationally) to perform interprocedural analysis on: Fortran IV, Pascal, C++ ? Why? Of these languages, which is the hardest to perform an interprocedural analysis on? Why?

   (c) Sketch the algorithm for computing the analysis you described in part (a). You can use any framework you like (e.g. interprocedural dataflow analysis, abstract interpretation, etc.)