CMSC330 Spring 2008 Midterm #2

Discussion TA & Time (pick one): Eylul @ 10am Eylul @ 11am Wanli @ 11am

Do not start this exam until you are told to do so!

Instructions

- You have 70 minutes to take this midterm.
- This exam has a total of 112 points, so allocate 33 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- If you have a question, please raise your hand and wait for the instructor.
- Answer essay questions concisely using 1 sentence, or *at most* 2 sentences. Longer answers are not necessary and a *penalty may be applied*.
- In order to be eligible for partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit will not be given for illegible answers.

1. (14 pts) Context Free Grammars & Automata
    a. (2 pts) Explain how context free grammars are used for programming languages.
    b. (2 pts) Describe the relationship between derivations and sentential forms.
    c. (2 pts) Describe the language accepted by the grammar: $S \rightarrow aaSb \mid aSb \mid \varepsilon$
    d. (4 pts) Write a grammar for $a^x b^y a^z$, where $z = 2x - y$, for $x,y,z \geq 0$
    e. (2 pts) Name features needed by automata to recognize all binary numbers with more 1's than 0's.
    f. (2 pts) Explain why a finite automaton with 2 stacks can recognize many more languages than a finite automaton with 1 stack.

2. (14 pts) Derivations, Parse Trees, Precedence and Associativity
    For the following grammar: $S \rightarrow S$ and $S \mid$ not $S \mid$ true $\mid$ false
    a. (4 pts) List all left-most derivations for the string "not true and true"
    b. (2 pts) Draw the parse tree for one of the left-most derivation above.
    c. (6 pts) Rewrite the grammar so that "and" is left associative and has lower precedence than "not".
    d. (2 pts) Is your rewritten grammar ambiguous?
        (*Full credit only for plausible rewritten grammar*)

3. (16 pts) Parsing
    For the problem, assume the term "predictive parser" refers to a top-down, non-backtracking, recursive descent parser.
    a. (10 pts) Consider the following grammar: $S \rightarrow Ac \mid b \quad A \rightarrow aS \mid \varepsilon$
        i. (4 pts) Compute First sets for each production and nonterminal
        ii. (4 pts) Write a predictive parser for the grammar
        iii. (2 pts) Use your parser to parse the string "abc". Show the sequence of calls in the parse, and what symbols remain at each point.
    b. Consider the following grammar: $S \rightarrow aSc \mid ab \mid a$
        i. (2 pts) Show why the grammar cannot be parsed by a predictive parser.
        ii. (4 pts) Rewrite the grammar so it can be parsed by a predictive parser, using the rules presented in class for left factoring & eliminating left recursion.

4. (8 pts) OCaml and Functional Programming
    a. (2 pts) Describe one advantage of functional programming
    b. (2 pts) Describe the difference between the usage of ";" and "," in OCaml
    c. (2 pts) Describe the relationship between type inference and polymorphic types
    d. (2 pts) Describe the difference between function pointers and closures

5. (10 pts) OCaml Types & Type Inference 1
   Give the type of the following OCaml expressions:
   a. (2 pts) [1,"bar"]
   b. (2 pts) let rec f x = match x with
      ```
      [] -> []
      | (h::t) -> (h+1)::(f t)
      ```
   c. (2 pts) let f (x::y) = [y;[x]]
   d. (4 pts) let f x y z = y x

6. (12 pts) OCaml Types & Type Inference 2
   Write an OCaml expression with the following types:
   a. (2 pts) string list list
   b. (4 pts) 'a * ('b list) -> ('a * 'b) list
   c. (6 pts) (int -> 'a) -> (int -> 'a)

7. (12 pts) OCaml Programs
   What are the values of the following OCaml expressions? If an error exists, describe the error.
   a. (2 pts) 1 + 2 ; 3 + 4
   b. (2 pts) [1;"foo"]
   c. (2 pts) let x = 1 in let y = x+2 in let x = y+3 in x+4
   d. (3 pts) let x y = fun z -> z+y in x 1 2
   e. (3 pts) let x y = fun z -> y z in x (fun x -> x+3) 4

8. (26 pts) OCaml Programming
   For the following problems, you may use helper functions, but no library functions.
   You are given the curried version of the fold function:
   ```
   let rec fold f a l = match l with
     [] -> a
   | (h::t) -> fold f (f a h) t
   ```
   a. (4 pts) Using the curried version of the *fold* function, write an OCaml function named *reverse* that when applied to a list *lst* returns the list in reverse order.
      Example: reverse [1;3;5;2;4] = [4;2;5;3;1]
   b. (10 pts) Using the curried version of the *fold* function, write an OCaml function named *filter* with type ('a -> bool) -> 'a list -> 'a list) that takes two arguments: a predicate function *pred* with type ('a -> bool), and list *lst* with type ('a list). *filter* returns only the elements of *lst* that return true when evaluated by *pred*. The filtered elements must be returned in their order in *lst*. You may use the reverse function above.
      Example: filter (fun x -> (x > 2)) [1;3;5;2;4] = [3;5;4]
   c. (12 pts) Write an OCaml function named *rev_map* which takes a function *f* and a list *lst*, applies *f* to every element *lst*, and returns the results in a new list in reverse order. You must implement rev_map as a single pass over the input list (i.e., you cannot first apply map, then reverse the result).
      Example: rev_map (fun x -> x+1) [1;3;5;2;4] = [5;3;6;4;2]