CMSC 330, Spring 2008, Midterm 2 Practice Problems

1. Context Free Grammars
   a. List the 4 components of a context free grammar.
   b. Describe the relationship between terminals, non-terminals, and productions.
   c. Define ambiguity.
   d. Describe the difference between scanning & parsing.

2. Describing Grammars
   a. Describe the language accepted by the following grammar:
      \[ S \rightarrow abS \mid a \]
   b. Describe the language accepted by the following grammar:
      \[ S \rightarrow aSb \mid \epsilon \]
   c. Describe the language accepted by the following grammar:
      \[ S \rightarrow bSb \mid A \quad A \rightarrow aA \mid \epsilon \]
   d. Describe the language accepted by the following grammar:
      \[ S \rightarrow AS \mid B \quad A \rightarrow aAc \mid Aa \mid \epsilon \quad B \rightarrow bBb \mid \epsilon \]
   e. Describe the language accepted by the following grammar:
      \[ S \rightarrow S \text{ and } S \mid \text{true} \mid \text{false} \]
   f. Which of the previous grammars are left recursive?
   g. Which of the previous grammars are right recursive?
   h. Which of the previous grammars are ambiguous? Provide proof.

3. Creating Grammars
   a. Write a grammar for \( a^x b^y \), where \( x = y \)
   b. Write a grammar for \( a^x b^y \), where \( x > y \)
   c. Write a grammar for \( a^x b^y \), where \( x = 2y \)
   d. Write a grammar for \( a^x b^{y/a} \), where \( z = x+y \)
   e. Write a grammar for \( a^x b^{\sqrt{a}} \), where \( z = x-y \)
   f. Write a grammar for all strings of \( a \) and \( b \) that are palindromes.
   g. Write a grammar for all strings of \( a \) and \( b \) that include the substring \( baa. \)
   h. Write a grammar for all strings of \( a \) and \( b \) with an odd number of \( a \)'s and \( b \)'s.
   i. Write a grammar for the “if” statement in OCaml
   j. Write a grammar for all lists in OCaml
   k. Which of your grammars are ambiguous? Can you come up with an unambiguous grammar that accepts the same language?

4. Derivations, Parse Trees, Precedence and Associativity
   For the following grammar: \( S \rightarrow S \text{ and } S \mid \text{true} \)
   a. List all derivations for the string “true and true and true”.
   b. Label each derivation as left-most, right-most, or neither.
   c. List the parse tree for each derivation
   d. What is implied about the associativity of “and” for each parse tree?

   For the following grammar: \( S \rightarrow S \text{ and } S \mid S \text{ or } S \mid \text{true} \)
   e. List all parse trees for the string “true and true or true”
f. What is implied about the precedence/associativity of “and” and “or” for each parse tree?
g. Rewrite the grammar so that “and” has higher precedence than “or” and is right associative

5. Parsing
For the problem, assume the term “predictive parser” refers to a top-down, recursive descent, non-backtracking predictive parser.
a. Consider the following grammar: \( S \rightarrow S \text{ and } S \mid S \text{ or } S \mid (S) \mid \text{true} \mid \text{false} \)
   i. Compute First sets for each production and nonterminal
   ii. Explain why the grammar cannot be parsed by a predictive parser
b. Consider the following grammar: \( S \rightarrow aS \mid acS \mid c \)
   i. Compute First sets for each production and nonterminal
   ii. Show why the grammar cannot be parsed by a predictive parser.
   iii. Rewrite the grammar so it can be parsed by a predictive parser.
   iv. Write a predictive parser for the rewritten grammar.
c. Consider the following grammar: \( S \rightarrow Sa \mid Sc \mid c \)
   i. Show why the grammar cannot be parsed by a predictive parser.
   ii. Rewrite the grammar so it can be parsed by a predictive parser.
   iii. Write a recursive descent parser for your new grammar
d. Describe an abstract syntax tree (AST)

6. Automata
a. Describe regular grammars.
b. Describe the relationship between regular grammars and regular expressions.
c. Name features needed by automata to recognize
   i. Regular languages (i.e., languages recognized by regular grammars)
   ii. Context-free languages
   iii. All binary numbers
   iv. All binary numbers divisible by 2
   v. All prime binary numbers
d. Compared finite automata, pushdown automata, and Turing machines
e. Describe computability
f. Describe a Turing test

7. OCaml and Functional Programming
a. Define functional programming
b. Define imperative programming

c. Define iterative programming.
d. Define higher-order functions
e. Describe the relationship between type inference and static types
f. Describe the properties of OCaml lists
g. Describe the properties of OCaml tuples
h. Define pattern variables in OCaml
i. Describe the usage of “_” in OCaml
j. Describe polymorphism
k. Write a polymorphic OCaml function
l. Describe variable binding
m. Describe scope
n. Describe lexical scoping
o. Describe dynamic scoping
p. Describe environment
q. Describe closure
r. Describe currying

8. OCaml Types & Type Inference
   Give the type of the following OCaml expressions:
   a. []
   b. 1::[]
   c. 1::2::[]
   d. [1;2;3]
   e. [[1];[1]]
   f. (1)
   g. (1,"bar")
   h. ([1,2], ["foo","bar"])  
   i. [(1,2,"foo");(3,4,"bar")]
   j. let f x = 1
   k. let f (x) = x *. 3.14
   l. let f (x,y) = x
   m. let f (x,y) = x+y
   n. let f (x,y) = (x,y)
   o. let f (x,y) = [x,y]
   p. let f x y = 1
   q. let f x y = x*y
   r. let f x y = x::y
   s. let f x = match x with [] -> 1
   t. let f x = match x with (y,z) -> y+z
   u. let f (x::_) = x
   v. let f (_,y) = y
   w. let f (x::y::_) = x+y
   x. let f = fun x -> x + 1
   y. let rec x = fun y -> x y
   z. let rec f x = if (x = 0) then 1 else 1+f (x-1)
   aa. let f x y z = x+y+z in f 1 2 3
   bb. let f x y z = x+y+z in f 1 2
   cc. let f x y z = x+y+z in f
   dd. let rec f x = match x with
        [] -> 0
        | (::_t) -> 1 + f t
   ee. let rec f x = match x with
        [] -> 0
        | (h::t) -> h + f t
ff. let rec f = function
    [] -> 0
    | (h::t)  -> h + (2*(f t))

gg. let rec func (f, l1, l2) = match l1 with
    [] -> []
    | (h1::t1) -> match l2 with
      [] -> [f h1]
      | (h2::t2) -> [f h1; f h2]

9. OCaml Types & Type Inference
   Write an OCaml expression with the following types:
   a. int list
   b. int * int
   c. int -> int
   d. int * int -> int
   e. int -> int -> int
   f. int -> int list -> int list
   g. int list list -> int list
   h. ‘a -> ‘a
   i. ‘a * ‘b -> ‘a
   j. ‘a -> ‘b -> ‘a
   k. ‘a -> ‘b -> ‘b
   l. ‘a list * ‘b list -> (‘a * ‘b) list
   m. int -> (int -> int)
   n. (int -> int) -> int
   o. (int -> int) -> (int -> int) -> int
   p. (‘a -> ‘b) * (‘c * ‘c -> ‘a) * ‘c -> ‘b

10. OCaml Programs
   What is the value of the following OCaml expressions? If an error exists, describe the error.
   a. 2 ; 3
   b. 2 ; 3 + 4
   c. (2 ; 3) + 4
   d. if 1<2 then 3 else 4
   e. let x = 1 in 2
   f. let x = 1 in x+1
   g. let x = 1 in x ; x+1
   h. let x = (1, 2) in x ; x+1
   i. (let x = (1, 2) in x) ; x+1
   j. let x = 1 in let y = x in y
   k. let x = 1 let y = 2 in x+y
   l. let x = 1 in let x = x+1 in let x = x+1 in x
   m. let x = x in let x = x+1 in let x = x+1 in x
   n. let rec x y = x in 1
   o. let rec x y = y in 1
p. let rec x y = y in x 1
q. let x y = fun z -> z+1 in x
r. let x y = fun z -> z+1 in x 1
s. let x y = fun z -> z+1 in x 1 1
t. let x y = fun z -> x+1 in x
u. let rec x y = fun z -> x+1 in x 1
v. let rec x y = fun z -> x+y in x
w. let rec x y = fun z -> x y in x 1
x. let rec x y = fun z -> x z in x 1
y. let x y = y 1 in x
z. let x y = y 1 in x
aa. let x y = y 1 in x 1
bb. let x y = y 1 in x fun z -> z + 1
cc. let x y = y 1 in x (fun z -> z + 1)
dd. let a = 1 in let f x y z = x+y+z+a in f 1 2 3
e. let a = 1 in let f x y z = x+y+z+a in f 1 2 -3

11. OCaml Programming
   a. Write an OCaml function named \texttt{fib} that takes an int \texttt{x}, and returns the Fibonacci number for \texttt{x}. Recall that \texttt{fib(0) = 0, fib(1) = 1, fib(2) = 1, fib(3) = 2}.
   b. Write an OCaml function named \texttt{concat} which takes 2 lists and returns the concatenated list.
   c. Write an OCaml function named \texttt{map_odd} which takes a function \texttt{f} and a list \texttt{lst}, applies the function to every other element of the list, starting with the first element, and returns the result in a new list. Use \texttt{map_odd} and \texttt{fib} applied to the list \([1; 2; 3; 4; 5; 6; 7]\) to calculate the Fibonacci numbers for 1, 3, 5, and 7.
   d. Given the \texttt{fold} function, write an OCaml function named \texttt{all_true} which may be applied to a list of booleans \texttt{lst} so that it returns true only if all elements of \texttt{lst} are true.
   e. Write an OCaml function named \texttt{paths_blocked f m n b} that computes the number of paths from \((m,n)\) to \((1,1)\) that pass through exactly \(b\) blocked intersections. So \texttt{paths_blocked f m n 0} should yield the same result as \texttt{paths}. The number of blocked intersections (on the path) increases by 1 every time the path passes through a blocked intersection. Thus leaving \((m,n)\) or arriving at \((1,1)\) will not count as passing through a blocked intersection even if \((m,n)\) and \((1,1)\) are blocked.
   f. Write an OCaml function named \texttt{nth} which has a tuple of an int named \texttt{n} and a list as a parameter and which returns the \texttt{n}'th element of the list. The list's first element is considered to be element number 1. For example:

\[
\texttt{nth (1, [2; 4; 6; 8; 10]) would return 2} \\
\texttt{nth (2, [2; 4; 6; 8; 10]) would return 4} \\
\texttt{nth (3, ["hi"; "ciao"; "bye"]) would return "bye"}
\]
You can assume the list will always have an \texttt{n}'th element to be returned, and you can also assume that \texttt{n} > 0. It doesn't matter if your function would generate any incomplete match warnings.