Practice Problems for Comp 333 Final Spring 2015

Note: There is also a file of concepts you need to know for the final.

1. Use grammar G1 to find a parse tree for the following program. Start with <prog> as the root of your tree. Do not expand B1, B2, S1 or S2. Assume they will expand into boolean expressions and statements as needed.

***begin x = 6; if ( B1) then if (B2) then S1 endif else S2 endif end***

1. List the grammar rules that are needed to add a *read statement* and a *write statement* to grammar G1. The read statement should read exactly one variable at a time. The write statement should be able to print as many variables as desired in one statement. For example, **read a** and **write(x,y,z)** should be legal statements. Do not rewrite the grammar here. Just list the new rules.

**G1: Grammar for a small language ( SML)**

**<prog>** 🡪 **begin <stmt\_list> end**

**<stmt\_list>** 🡪 **<stmt> | <stmt>; <stmt\_list>**

**<stmt>** 🡪 **<assign\_stmt> | <if\_stmt>**

**<assign\_stmt>**🡪 **<var> = <expression>**

**<if\_stmt>** 🡪 **if ( <bool\_expr>) then <stmt\_list> endif**

**| if( <bool\_expr>) then <stmt\_list> else <stmt\_list> endif**

**<expr>** 🡪 **<expr> <op> <expr>**

**| ( <expr>)| <var> |<int>**

**<op>** 🡪 **+ | - | \* | /**

**<rop>** 🡪 **< | <= | > | >= | != | ==**

**<rel\_expr>** 🡪 **<expr> <rop> <expr>**

**<bop>** 🡪 **and | or**

**<bool\_expr>** 🡪 **<rel\_expr>**

**|<bool\_expr> <bop> <bool\_expr> | not <bool\_expr> | ( <bool\_expr>)**

**<var>** 🡪 **a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z**

**<digit>** 🡪 **0|1|2|3|4|5|6|7|8|9|**

**<int>** 🡪 **<digit> | <digit> <int>**

1. Use grammar G2 to find a parse tree for the expression

***(a + 12) \* b – 2\*x + 3***

**G2: Unambiguous Expression Grammar**

**E 🡪 E + T | E -T |T**

**T 🡪 T \*F | T / F | F**

**F🡪 ( E ) | <var> | <int>**

**<var> 🡪 a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z**

**<digit> 🡪 0|1|2|3|4|5|6|7|8|9|**

**<int> 🡪 <digit> | <digit> <int>**

1. Consider the following grammar.

<list> 🡪 <list> <stmt> //rule 1

<list> 🡪 <stmt> //rule 2

<stmt> 🡪 <var> := <num> ; //rule 3

<var> 🡪 x|y|z //rule 4

<num> 🡪 0|1|2|3|4|5|6|7|8|9 // rule 5

Show that ***x := 6; y := 7;*** can be derived from <list>. Use a left-most derivation. (Not a parse tree). Use one rule for each step. Indicate rule used for each step.

1. In this problem you will show that the following grammar G3 is ambiguous

S🡪 if (E) then A | if (E) then A else A

E 🡪 u | v | w

A 🡪 S | x | y

Terminals = { if , then, else, (, ), u,v, w, x , y}

Nonterminals = { S,A,E}

Let S be the start symbol.

* 1. What do you need to do to show that G3 is ambiguous?
  2. Use the string ***if( u ) then if( v ) then x else y*** to show that G3 is ambiguous.

1. ( 12 pts) Describe the regular expression in English. To illustrate your description, give examples of strings that match the pattern and strings of digits, periods and spaces that don’t match the pattern.

D+[.D+] {<space>D+[.D+]}\*

D 🡪 0|1|2|3|4|5|6|7|8|9

1. Write a recursive Scheme function called *squares* that takes list1 and list2 as parameters and returns #t if the elements of list2 are the squares of the elements of list 1, in the same order. It should return #f otherwise. Assume that list1 and list2 are lists of numbers.

Usage: ( squares ‘( 3 7 9 - 5 ) ‘( 9 49 81 25) ) returns #t

Usage: (squares ‘ ( 5 2 10 ) ‘ ( 25 1 100) ) returns #f

Usage: (squares ‘( 4 ) ‘( 16 20) ) returns #f

Usage: (squares ‘() ‘() ) returns #t

Use the following outline for your function :

if lengths of list1 != length of list2

return #f

else if list1 is null

return #t

else if first element of list2 == the square of the first element of list1

return ???? <- fill in this

else

return false

1. (10 pts)
   1. Let alist = ‘(a (b c) ( d e f) ) . What is the value of

(cons (car (cdr (cdr alist) )) (cdr (cdr alist) ) ) ? Show work.

* 1. Suppose f1 is defined as

( define f1

( lambda ( a k)

( if (null? a)

'( )

(cons k ( f1 ( cdr a) ( + k 1)))

) ))

What is returned by the Scheme expression (f1 ‘(a b c d) 1) ? Show work by showing all calls to f1.

1. Write a recursive Scheme function, *keeper*, which takes a positive integer k and a list of strings as arguments and returns the list of those strings which have length k. Do not use filter or map.

Usage: (keeper 3 ‘( “art” “ baker” “ antelope” “cat” “blue” ) ) returns ‘( “art” “cat”)

[**Hint:** Use the string-length function. For example, (string-length “catapult” ) is 8. ]

1. Use the Java program Scoper2 on next page to answer the following questions.
   1. Identify and label the scoping blocks. **Draw scope boxes on the Scoper2 code on next page.**
   2. List all variables in the program and their scope visibility. Subscript all variables with the block number they are defined in. **Put your answers in empty three-column box below.**
   3. What is the referencing environment at line 21?
   4. What is printed out by this program?

1.   
 2   
 3 public class Scoper2  
 4 {  
 5 public static int t = 3;   
 6 public static int b = 8;  
 7 public static int x = 6;  
 8   
 9 public static void g ( int x )  
10 {  
11 if( x < b)  
12 {  
13 int t = 2;  
14 System.out.println( x + t);  
15 b = b + 1;  
16 }  
17 else  
18 {  
19 int r = 5 \* t;  
20 t = t + 4;  
21   
22 System.out.println( r);   
23 }   
24 }  
25   
26 public static void main( String[] args)  
27 {   
28 g(x +1);  
29 g(x + 3);  
30 System.out.println( t + " " + b + " " + x );  
31 }  
32 }

Variable (subscripted) Visible in Blocks Not Visible in Blocks

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. 1. Suppose the program below uses dynamic scope rules. Draw the function calling tree for the program below and use it to discuss how dynamic scoping works. What is printed out by the program?

int a = 100; //global

int b = 25; //global

int f ( )

{ int a = 5; int b = 10; h(); }

int g ( )

{ print a; print b; }

int h()

{ int b = -2;

a = a +1; g();

}

main ()

{

int a = 20;

print “call f”

f( );

print “ call h”

h() ;

}

* 1. What is printed out by this program if static scope rules are used instead?

1. Consider the following program written in C-like syntax:

|  |  |
| --- | --- |
| int f( int a, int b) {  if( a < b)  b = b - a;  else  a = a – b;  return a+b;  } | void main() {  int x = 7;  int y = 10;  int u = f(x,y);  int v = f( y,x),  print ( x, y , u, v); // prints the values of x , y, u  } |

* 1. What is printed out if the parameters are all passed by value? Show work.
  2. What is printed out if the parameters are all passed by reference? Show work

1. Scheme lists. What expressions are returned by the following Scheme expressions?
2. ( cdr ( car ( cdr ( cdr '( (10 20 ) ( 30 40 ) ( 50 60) ) ) ) ) )
3. (car ( cdr (list ( + 10 20 ) ( + 30 40) ( + 50 60) ) ) )
4. (map (lambda (x) ( + ( \* 2 x ) 1 ) ) ‘( 3 4 5 ) )
5. ( let ( [ x '(a b c d)])

(cons ( car x)

(cons (cadr x)

(cons 'e (cddr x)))))

1. (filter (lambda (u) ( > (length u) 2) ) ‘( (a b ) ( e r t y) ( c d ) (2) ( a b c)) )
2. Consider the program below. Assume that static scope rules are used and that parameters are passed by value. Draw in the scoping blocks. Show what the run-time stack looks when the program reaches location A. Carefully draw the stack frames showing only variables defined in the block that the stack frame represents. What is printed out by this program? Show work.

|  |
| --- |
| int a = 11; //global  void sub1( int b)  {  int x = a;  a = b +x;  }  void sub2 ( int u)  {  int y = u +1;  int a = 30;  void sub3 ( )  {  int y = 15;  a = 40;  //location A  sub1( 20 ) ;  }  sub3();  **print a + “ “ + y;**  }  void main()  {  int x = 9;  sub2(x);  **print a + “ “ + x;**    } |

1. Use the Grammar G4 below for this problem.

**Grammar G4**

<program> 🡪 begin <stmt\_list> end . | begin <var\_decl> ; <stmt\_list> end .

<var\_decl> 🡪 int <var\_list>

<stmt\_list> 🡪 <stmt> | <stmt>; <stmt\_list>

<stmt> 🡪 <assign\_stmt> | <while\_stmt> | <if\_stmt>

<assign\_stmt>🡪 <var> = <expr>

<while\_stmt> 🡪 while ( <bexpr) <stmt\_lst> endwhile

(if\_stmt> 🡪 if (<bexpr>) <stmt\_lst> endif | if (<bexpr>) <stmt\_lst> else <stmt\_lst> endif

<var> 🡪 *letter*

<var\_list> 🡪 <var> | <var>, <var\_list>

<expr> 🡪 <expr> <op> <expr> | (<expr> ) |<var> | *integer*

<op> 🡪 + | \* | - | /

<bexpr> 🡪 <expr> <relop> <expr>

<bexpr> 🡪 (<bexpr> ) | not <bexpr> | <bexpr> and <bexpr> | <bexpr> or <bexpr>

<relop> 🡪 < | <= | > | >= | == | !=

* 1. Draw a parse tree for the following program using G4. Start with <program>

begin

while ( x > 0) x = x – 1 endwhile

end.

* 1. Fix the syntactic errors in the following program. The program should satisfy the syntax of Grammar G4

begin

int x = 20;

int y = 10;

while ( x > 0 && y != x )

begin

y = y + 1

x = x – 1

end ;

y = 5

end

1. Consider the following Scheme function. For all parts of the question, assume that lst1 is a list of numbers or the empty list and, similarly, assume that lst2 is a list of numbers or the empty list.

(define ( snap lst1 lst2 )

(if ( or ( null? lst) (null? lst2))

‘()

( cons ( + ( car lst1 ) (car lst2) ) (snap ( cdr lst1) ( cdr lst2) ) ) ) )

* 1. What does the snap function do? Illustrate with an example
  2. What will the snap function do if lst1 and lst2 have different lengths? Illustrate your answer with an example.

1. This problem uses a Prolog database with the following facts and rules.

|  |  |
| --- | --- |
| female(sue).  female( joan).  female(ann).  female( mary).  male(joe).  male(bob).  male(carl).  male(andy). | parent( sue, carl). //sue is a parent of carl  parent (sue, judy).  parent(sue, andy).  parent( bob, ann).  parent( joan, mary).  parent(carl, lisa).  sibling(X,Y) :- parent(Z,X), parent(Z,Y), not( X = Y). |

* 1. Write a Prolog rule for the uncle relation that uses the sibling relation [ Note: uncle(X,Y) means that X is an uncle of Y]
  2. List all prolog calls and instantiations that are made to find all solutions to query ?- sibling(carl, B)

Complete the following Prolog definition for count where count (X,List,R) is true if R is the number of times X is in List.

count(X, [], 0).

count(X, [H|T], R) :- X = H, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

count(X, [H|T], R ) :- not(X = H), \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a Prolog predicate that computes the sum of the positive numbers in a list of numbers and computes the sum of the negative numbers in the same list of numbers. Usage:

sumsPN( [2,3, -5,7,2, -10, 3, 0], PosSum, NegSum) returns PosSum = 17 and NegSum = -15.

1. Write Prolog predicates that
   1. find the length of a list *length(L,Result).*
   2. determine if all elements of a list are unique *all\_different(L).*
   3. finds a list that contains the intersection of two lists *intersection(L1,L2,Result).*
   4. finds all distinct numbers X,Y and Z such that X + Y + Z = N and X,Y and Z are between 1 and N. *partitions(X,Y,Z, N).* Use between(N,M,X). See Problem 24.
2. Repeat problems 21a-c for Scheme functions. For 21d just create a Scheme function that determines if inputs X,Y,Z and N meet the conditions. You do not have to generate all solutions in Scheme.
3. Consider the following expression grammar. The capital letters are the nonterminals.

Grammar G2:

E 🡪 V E A | V

V 🡪 a | b | c| x | y | z

A 🡪 + | - | \* | /

* 1. Give a leftmost derivation of **x y b \* +** . Show all steps.
  2. Show that there is no parse tree for the expression **a b + \***  .

1. Consider the following acyclic, directed graph represented as edge facts. Here edge(a,b) means that there is an edge from a to b. Draw a picture of the graph.

edge(b,d).

edge( c,e).

edge(b,c).

edge(a,b).

edge(a,c).

Review the Prolog path predicate path(X,Y,P) which sets P to the path if there is a path from X to Y and false otherwise. (See Power Points Chap 11, Part 3 where this predicate is defined. **Trace your solution** on path( a,e,L). Show all calls to path and edge and show where backtracking is used. Apply the edge facts in order listed. **Which path is found first?**

1. Use *between(M,N,X)* to write a Prolog predicate, *factors( [X,Y],N)*, that generates all pairs of positive integers whose product is N. For example,

?- factors( [X,Y], 10).

X = 1 Y = 10;

X = 2, Y = 5;

X = 5, Y = 2;

false.

1. Use the Producer Consumer java concurrency program (NEXT 4 PAGES) to answer these questions. These are independent questions.
   1. What happens to the behavior of the program if we remove the notifyAll() statement from the read method in the Buffer class? Be specific.
   2. What happens to the behavior of the program if we replace the wait() statement in the write method in the Buffer class with a Thread.sleep( 1000) statement.
   3. Modify the producer and consumer tasks ( WRITE ON PROGRAM) so that the producer stops writing numbers to the buffer after she writes 50 numbers. The last number she writes to the buffer should be -999. Modify program so that consumer stops reading numbers after he reads the -999. There should be no deadlocks.

**/\* Consumer Producer Problem\*/**  
  
 import java.util.\*;  
 import java.util.concurrent.\*;  
 public class ProducerConsumer  
 {  
 private static Buffer buff = new Buffer();  
   
 public static void main( String[] args)  
 {  
 ProducerTask producer = new ProducerTask();  
 ConsumerTask consumer = new ConsumerTask();  
 Thread pthread = new Thread(producer);  
 Thread cthread = new Thread(consumer);  
 pthread.start();  
 cthread.start();  
   
 }

//Nested class   
 public static class ProducerTask implements Runnable  
 {  
 public void run()  
 {  
 try {  
 while( true)  
 {  
 int x = (int)(Math.random()\*100) + 1;  
 buff.write(x);  
 System.out.println("Producer writes " + x);  
 Thread.sleep((int)(Math.random()\*1000));  
 }  
 }  
 catch( InterruptedException ex)  
 { ex.printStackTrace(); }  
   
 }  
   
 }

//Nested class  
 public static class ConsumerTask implements Runnable  
 {  
   
 public void run()  
 {  
 try{  
 while(true) {  
 int u = buff.read();  
 System.out.println("\t\t\t\tConsumer reads " + u);  
 Thread.sleep((int)(Math.random()\*3000));  
 }  
 }  
 catch(InterruptedException ex)  
 { ex.printStackTrace();  
 }  
   
 }   
 }

/\* Nested class. Buffer is a Monitor \*/  
 public static class Buffer  
 {  
 private static final int CAPACITY = 10;  
 private LinkedList<Integer> queue = new LinkedList<Integer>();  
   
 public synchronized void write( int k)  
 {  
 try {  
 while( queue.size() == CAPACITY)  
 wait();   
 queue.addLast(k);  
 notifyAll();  
 }  
 catch( InterruptedException ex)  
 { ex.printStackTrace(); }  
   
 }  
   
 public synchronized int read()  
 {  
 int u = 0;  
   
 try {  
 while(queue.size() == 0)  
 wait();  
 u = queue.removeFirst();  
 notifyAll();  
   
 }  
 catch ( InterruptedException ex)  
 { ex.printStackTrace(); }  
 return u;  
 }  
   
 }