

UNIVERSITY OF TORONTO  
Faculty of Arts and Science  
MAY EXAMINATIONS 2000

CSC 324H1 Y  
St. George Campus

Duration — 3 hours

Examination Aids: No Exam Aids Allowed

Student Number: \_\_\_\_\_

Last Name: \_\_\_\_\_

First Name: \_\_\_\_\_

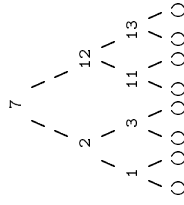
*Do not turn this page until you have received the signal to start.*  
(In the meantime, please fill out the identification section above,  
and read the instructions below carefully.)

# 1: \_\_\_\_\_ / 6  
# 2: \_\_\_\_\_ / 10  
# 3: \_\_\_\_\_ / 4  
# 4: \_\_\_\_\_ / 6  
# 5: \_\_\_\_\_ / 6  
# 6: \_\_\_\_\_ / 4  
# 7: \_\_\_\_\_ / 6  
# 8: \_\_\_\_\_ / 8  
# 9: \_\_\_\_\_ / 6  
# 10: \_\_\_\_\_ / 10  
# 11: \_\_\_\_\_ / 10  
# 12: \_\_\_\_\_ / 4  
# 13: \_\_\_\_\_ / 12  
# 14: \_\_\_\_\_ / 23  
TOTAL: \_\_\_\_\_ / 115

*Good Luck!*

**Question 1.** Tree Swap in Scheme [6 MARKS]

We might think of a list graphically as a tree structure. A binary tree is a data structure consisting of nodes and two branches leading to the left and right subtrees. Each node has exactly one value *value* and two branches, *left* and *right*. For example, the list (7 (2 (1 (0) (3 (0) (0))) (11 (0) (0) (13 (0) (0)))) represents the following tree:



Write a recursive Scheme function (`swap-branches tree`) that takes a tree and returns another tree which is the input tree where each node has its left and right children swapped. For example,

```
1 ]=> (swap-branches '(7 (2 (1 (0) (3 (0) (0))) (12 (11 (0) (0) (13 (0) (0))))))
;Value 1: (7 (12 (13 (0) (0)) (11 (0) (0))) (2 (3 (0) (0)) (1 (0) (0))))
```

Use (`root tree`), (`left tree`), and (`right tree`) as helping functions. These functions are defined as follows:

```
(define (root tree) (car tree))
(define (left tree) (cadr tree))
(define (right tree) (caddr tree))
```

This final examination consists of 14 questions on 14 pages (including this one). When you receive the signal to start, please make sure that your copy of the examination is complete. Answer each question directly on the examination paper, in the space provided, and use the reverse side of the pages for rough work. (If you need more space for one of your solutions, use the reverse side of the page and indicate clearly which part of your work should be marked.)

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly.

**General Hint:** We were careful to leave ample space on the examination paper to answer each question, so if you find yourself using much more room than what is available, you're probably missing something. Also, remember that hints are just hints: you are not required to follow them!

**Question 2.** Deep Mapping in Scheme [10 MARKS]

Using Scheme, define a map-like function called `gmap`. If `F` is a unary function (a function taking one argument) and `Exp` is a number or a nested list of numbers, then `(gmap F Exp)` is derived from `Exp` by replacing every number, `N`, by `(F N)`. For example,

```
(gmap zero? '(0 1 0 2 3)) => (#t #f #t #f #f)
```

```
(gmap (lambda (X) (+ 1 X)) '(1 (2 3) (4 (5) 6))) => (2 (3 4) (5 (6) 7))
```

```
(gmap (lambda (X) (+ 1 X)) '()) => ()
```

```
(gmap (lambda (X) (* X X)) '5) => 25
```

(a) [4 MARKS] Write the function `gmap` in Scheme, but do *not* use `map` in your definition.

(b) [6 MARKS] Write the function `gmap` in Scheme and use `map` to simplify your definition.

**Question 3.** List Manipulation in Scheme [4 MARKS]

Each of the following expressions has a missing piece. Give the name of a single Scheme function that completes the expression, making it have the value shown; or write "cannot" if no Scheme built-in function causes the expression to have the value shown.

Expression	Desired value	function name that completes the expression (or "cannot")
<code>( '(a b) '(c d))</code>	<code>((a b) (c d))</code>	
<code>( '(a b) '(c d))</code>	<code>((a b) c d)</code>	
<code>( '(a b) '(c d))</code>	<code>(a b (c d))</code>	
<code>( '(a b) '(c d))</code>	<code>(a b c d)</code>	

**Question 4.** Function Calling Mechanisms in Scheme [6 MARKS]

Write three different Scheme expressions, each of which returns the sum of integers `a`, `b` and `c`.

Expression using <code>apply</code> :	
Expression using <code>eval</code> :	
Expression using neither <code>apply</code> nor <code>eval</code> :	

**Question 5.** Scheme Code Analysis [6 MARKS]

Consider the following Scheme function:

```
(define (y s lst)
  (cond ((null? lst) ())
        ((equal? s (car lst)) lst)
        (else (y s (cdr lst))))
  )
```

- (a) [3 MARKS] What does this function do? Illustrate your answer with 3 non-trivial sample calls to *y*, showing the call and the resulting output for each example.

- (b) [3 MARKS] Derive the Big-Oh worst-case time complexity of *y*. Show your work.

**Question 6.** Scoping Rules [4 MARKS]

Consider the following program:

```
int x;

procedure set_x(int n) {
  x = n;
}

procedure print_x() {
  print x;
}

procedure first() {
  set_x(1);
  print_x;
}

procedure second() {
  int x;
  set_x(2);
  print_x;
}

set_x(0);
first();
print_x;
second();
print_x;
```

- (a) [2 MARKS] What does this program print if the language uses static scoping?

- (b) [2 MARKS] What does this program print if the language uses dynamic scoping?

**Question 7.** Parameter Passing [6 MARKS]

Consider the following program:

```
int a[5]; // Int array indexed from 0 to 4
int i;

procedure p(int x) {
    x = 0;
    i = 2;
    x = 6;
}

a[0] = 0; a[1] = 1; a[2] = 2; a[3] = 3; a[4] = 4;
i = 1;
p(a[i]);
print i;
print a; // prints the whole array
```

(a) [2 MARKS] What does this program print if the language uses call-by-value?

(b) [2 MARKS] What does this program print if the language uses call-by-value-result?

(c) [2 MARKS] What does this program print if the language uses call-by-name?

**Question 8.** Family Relations in Prolog [8 MARKS]

Given the predicates:

```
father(X,Y) % X is the father of Y
mother(X,Y) % X is the mother of Y
female(X) % X is female
male(X) % X is male
```

(a) [4 MARKS] Define the following predicates in First-Order Logic:

```
sibling(X,Y) % X is a sibling of Y
sister(X,Y) % X is a sister of Y
brother(X,Y) % X is a brother of Y
```

(b) [4 MARKS] Define the above three predicates in Prolog, as well as the following fourth one:

```
descendent(X,Y) % X is a descendent of Y
```

**Question 9.** Mystery Prolog Predicates [6 MARKS]

The following three mystery predicates are defined in Prolog:

```
p([X|Y],Z) :- p(X,W), p(Y,V), q(W,V,Z).
p(X,[X]) :- not(r(X)).
p([], []).
```

```
q([W|X],Y,[W|Z]) :- q(X,Y,Z).
q([],X,X).
```

```
r([]).
r([X|Y]).
```

- (a) [2 MARKS] In plain English, explain what predicate **r** does. Give a simple, concise explanation illustrated by two non-trivial examples.

- (b) [2 MARKS] In plain English, explain what predicate **q** does. Give a simple, concise explanation illustrated by two non-trivial examples.

- (c) [2 MARKS] In plain English, explain what predicate **p** does. Give a simple, concise explanation illustrated by two non-trivial examples.

**Question 10.** Prolog flip Predicates [10 MARKS]

Define Prolog predicates corresponding to the following operations on lists:

- (a) [4 MARKS] Write the `flip2(L1,L2)` predicate, which holds when **L1** and **L2** are lists which contain the same elements, but in “alternating” orders; it “transforms”

```
L1 = [E1, E2, E3, E4, ... En-1, En]
into
L1 = [E2, E1, E4, E3, ... En, En-1].
```

That is, `flip2` “forms” the second list by re-arranging the elements of the first list — exchanging the first element with the second, the third with the fourth, etc. If **L1** has an odd number of elements, then **L1**’s final element “remains in place” in **L2**.

The following statements hold, for example:

```
flip2( [1,2,3,4], [2,1,4,3] ).
flip2( [a,b,c], [b,a,c] ).
flip2( [[a,b],c], [c,[a,b]] ).
```

- (b) [6 MARKS] Write a “deep” form of this predicate, `df1ip2`, which exchanges alternate elements at each level. The following statements hold, for example:

```
df1ip2( [[a,b],[c]], [c],[b,a] ).
df1ip2( [ [1,2], [[3,4,5]],a,b], c, d ], [ [a,[4,3,5]],b], [2,1], d, c ] ).
```

**Question 11.** Unification [10 MARKS]

What is the result of unifying the following pairs of expressions? Write the answer in the appropriate box, or “cannot” if the two expressions cannot be unified.

Expression 1	Expression 2	Unification result
$q(X, f(a), g(Y))$	$q(h(b), Z, g(c))$	
$q(X, f(X), g(Y))$	$q(Z, f(a), g(c))$	
$q(X, f(g(Y)), Y)$	$q(Z, f(Z), h(b))$	
$[X, f(a), Y]$	$[X, Z Y]$	
$[[this, is]   X]$	$[[Y Z], a, list]$	

**Question 12.** Regular Expressions [4 MARKS]

For each of the following languages, write a regular expression for it, or write “cannot” if it cannot be described with a regular expression.

(a) [2 MARKS] All strings of *as* and *bs* that contain the substring *aaa* exactly once.

(b) [2 MARKS] All strings of *as* and *bs* that contain the same number of *as* and *bs*.

**Question 13.** Syntax of Prolog Lists [12 MARKS]

(a) [8 MARKS] Write a context-free grammar in BNF or EBNF for lists in Prolog. Your grammar must allow nested and improper lists, so it should accept  $[1, 2, 3]$ ,  $[]$  as well as  $[1, [3|4]]$ ,  $[], [5, 6, 7| [3, 5]]$ . Use the nonterminal `<atom>` defined below for all the atomic element of your lists—assume that all atoms are single-digit numbers and that the list contains no variables. Use `<list>` as your start symbol. Make sure your grammar is unambiguous.

```
<atom> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```
<list> -->
```

(b) [4 MARKS] Using your grammar, draw a parse tree for the input string  $[1, 2| [3]]$ .

**Question 14.** Short Answers [23 MARKS]

- (a) [4 MARKS] Give an unambiguous context-free grammar in BNF that can generate the language

$$a^n b^m a^{n-m}, \quad n \geq m \geq 0.$$

- (b) [3 MARKS] State two ways in which Prolog differs from pure logic.

- (c) [3 MARKS] Which search strategy is used by Prolog? Give one advantage and one drawback of this strategy.

- (d) [4 MARKS] List four problems with the design of the Perl programming language and, for each problem, indicate if it affects readability, writability and/or reliability.

- (e) [5 MARKS] What is the language generated by the following grammar? Is this grammar ambiguous? If it is ambiguous, prove it; otherwise, explain briefly how you can tell it's not ambiguous.

```
<S> --> a <P> a | b <P> b | aa | bb
<P> --> <empty> | <S>
```

- (f) [4 MARKS] A programmer wishes to write some facts in Prolog that say that all birds fly except penguins and emus. To do this, he writes the following:

```
fly(penguin, no).
fly(emu, no).
fly(OtherBird, yes).
```

Explain why this doesn't do what is wanted, and show how to correct the problem.