University of Toronto
St. George Campus

Computer Science 324
Fall 2004

Midterm Exam
Section L5101
21 October, 2004
DO NOT OPEN THIS BOOKLET UNTIL YOU ARE TOLD TO DO SO

Last Name:

First Name:

Student Number:

Rules:

1. Legibly write your name on this page and every page that you wish to be marked or returned. The pages of this exam will be separated during marking. *Unnamed pages will not be marked.*
2. There are a total of 50 marks and 2 problems. The second problem has 6 parts.
3. Total time is 50 minutes.
4. The exam is closed book, and no aids of any kind are allowed.
5. Write your answers directly on the exam in the space provided, or on the blank pages provided at the end. For each blank page that you use, write the number of the problem that you are solving on that page. Do not write the answer to more than one problem on the same blank page.
6. For rough work, you may use the back of any page. *These will not be marked.*

Do not write below this line.

1:

2:

Total:

Name:

1. (5 marks) A programming language has *referential transparency* when (circle the appropriate answer):

   a. it has pointers.

   b. replacing part of any expression with an expression of the same value does not change the value of the containing expression.

   c. all interfaces are apparent in the syntax.

   d. no function depends on any variable except its formal parameters.

2. (45 marks)

a. (5 marks) How many sentences are in the language generated by this grammar? Show your calculation.

$$
\begin{array}{l}
S \longrightarrow A\ B \\
A \longrightarrow D\ D \\
B \longrightarrow A\ (C \mid D) \\
C \longrightarrow a \mid b \mid c \mid d \mid e \\
D \longrightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9
\end{array}
$$

b. (5 marks) How many sentences are in the language generated by this grammar? Show your calculation.

$$
\begin{array}{l}
S \longrightarrow A\ B \\
A \longrightarrow D\ D \\
B \longrightarrow A\ (C \mid D) \\
C \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \\
D \longrightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9
\end{array}
$$

Name:

c. (5 marks) Recall that this is the way we defined `reduce` in class:

```
(define (reduce fun nulval list)
  (if (null? list) nulval (fun (car list) (reduce fun nulval (cdr list)))))
```

(the Scheme built-in uses a different definition). There is also a Scheme built-in predicate, `member`, which, given an element and a list, returns true if and only if the element is a member of the list. Using these two procedures, define a procedure, `n-intersect`, which takes two sets (unordered lists with no duplicates) as input, and returns the number of elements in their intersection.

```
(define n-intersect
  (lambda (list1 list2)
    (reduce (lambda (x n) (if (member x list2) (+ n 1) n)) 0 list1)))
```

d. (20 marks) A CFG is said to be *recursive* iff it is possible to derive from a non-terminal, N, a string containing N. Define a procedure, `n-sentences`, which takes a *non-recursive* and *unambiguous* CFG as input and counts the number of sentences in the language it generates. You may assume that every non-terminal occurs on the left-hand-side of exactly 1 rule, and that every right-hand-side is either (1) composed from "pipes" (|) and concatenations of non-terminals or (2) a "pipe" of terminal symbols. Under these assumptions, a grammar may be represented as a Scheme association list. The grammar for part (a), for example, is:

```
(define AGRAMMAR '((S (A B))
                   (A (D D))
                   (B (A (pipe C D)))
                   (C (lex a b c d e))
                   (D (lex 0 1 2 3 4 5 6 7 8 9))))
```

Notice the use of `lex` to indicate that a particular RHS is a pipe of terminal symbols (Scheme is not case-sensitive). Concatenations are represented as lists. Concatenations can contain `pipe`s, and `pipe`s can contain concatenations. Assume that the distinguished non-terminal is always called `S`. You may use helper functions and `reduce`. You do *not* have to check that the grammar is non-recursive or unambiguous — just assume it.

```
(define n-cat
  (lambda (cat grammar) (reduce (lambda (rhs n) (* (n-rhs rhs grammar) n)) 1 cat)))

(define n-pipe
  (lambda (pipe grammar) (reduce (lambda (rhs n) (+ (n-rhs rhs grammar) n)) 0 pipe)))

(define n-rhs
  (lambda (rhs grammar) (cond ((symbol? rhs) (n-nonterm rhs grammar))
                              ((number? rhs) (n-nonterm rhs grammar))
                              (else (let ((first (car rhs)))
                                      (cond ((equal? first 'lex) (length (cdr rhs)))
                                            ((equal? first 'pipe) (n-pipe (cdr rhs) grammar))
                                            (else (n-cat rhs grammar)))))))))

(define n-nonterm
  (lambda (nonterm grammar) (n-rhs (cadr (assoc nonterm grammar)) grammar)))

(define n-sentences
  (lambda (grammar) (n-nonterm 'S grammar)))
```

Name:

e. (5 marks) How many sentences are in the language generated by this grammar? Explain your answer.

$$S \longrightarrow S \mid a$$

f. (5 marks) How many sentences are in the language generated by this grammar? Explain your answer.

$$S \longrightarrow a\ S$$

Name:
Blank page: Problem _____

Name:
Blank page: Problem _____