YOUR NAME PLEASE:
P. M. T. Solutions

Computer Science 201a
Practice Midterm
October 16, 2000

Open book and open notes. Show ALL work you want graded on the test itself, including the backs of pages as necessary.

For problems that do not ask you to justify the answer, an answer alone is sufficient. However, if the answer is wrong and no derivation or supporting reasoning is given, there will be no partial credit.

GOOD LUCK!

<table>
<thead>
<tr>
<th>problem</th>
<th>points</th>
<th>actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. Define a procedure (bit? x) that returns
#t if its argument is 0 or 1,
#f otherwise.

Examples:

(bit? 0) => #t,
(bit? 1) => #t,
(bit? 2) => #f,
(bit? '(a b)) => #f

(define bit?
  (lambda (x)
    (or (equal? x 0) (equal? x 1)))))
2. Define a procedure (cb ls) that takes a list ls and returns the number of top-level items in ls that are 0's or 1's. Your procedure should do flat recursion and generate a recursive process. You may use the predicate bit? from problem 1.

Example: (cb '(0 0 1 ((0 a) c 1) 1 0)) => 5

(define cb
  (lambda (ls)
    (cond
      ((null? ls) 0)
      ((bit? (car ls)) (+ 1 (cb (cdr ls))))
      (else (cb (cdr ls))))))
3. Define a procedure (cb-it ls count)
that is an iterative version of cb from problem 2.
That is, (cb-it ls 0) should return the same value as (cb ls).

(define cb-it
 (lambda (ls count)
   (cond
     ((null? ls) count)
     ((bit? (car ls)) (cb-it (cdr ls) (+ 1 count)))
     (else (cb-it (cdr ls) count)))))
4. Define a procedure (cb-deep ls)
   that returns the number of 0’s or 1’s
   at any level of the list ls.

   Example: (cb-deep '(0 0 1 ((0 a) c 1) 1 0)) => 7

   (define cb-deep
     (lambda (ls)
       (cond
         ((null? ls) 0)
         ((pair? (car ls)) (+ (cb-deep (car ls)) (cb-deep (cdr ls))))
         ((bit? (car ls)) (+ 1 (cb-deep (cdr ls))))
         (else (cb-deep (cdr ls)))))))
5. Recall the definition of map:

(define map
  (lambda (proc ls)
    (cond
      ((null? ls) '())
      (else (cons (proc (car ls)) (map proc (cdr ls)))))))

(a) What is the value of
(map (lambda (x) (* x x)) '(3 5 7))?

This returns a list of the squares of the input list:
(9 25 49)

(b) What is the value of
(map (lambda (x) (cons x '())) '(a (b c) (d) e))?

This wraps each top-level element of the list in another pair of parentheses:  ((a) ((b c)) ((d)) (e))
6. Consider the following Scheme expression:

```
(let ((x 5) (y 2))
  (let ((x (* y y)) (y (+ x x)))
    (list x y)))
```

(a) Rewrite the above expression using lambdas and no lets.

```
((lambda (x y)
    ((lambda (x y)
       (list x y)) (* y y) (+ x x)) 5 2)
```

(b) What is the value of the let expression given above?

```
(4 10)
```

(c) Draw a picture of the environments (global and local) when the (list x y) expression is evaluated as part of the evaluation of the let expression above. (Don’t forget pointers from one environment to another.)

```
-------------
|  global environment  |
|______________________|
^  
^______
|  x  | 5  |
|______|

local environment of outer let

^______
|  y  | 2  |
|______|

^______
|  x  | 4  |
|______|

local environment of inner let

^______
|  y  | 10 |
|______|
```
7. Consider the following RAM program, P.

1. lda 1  
2. sub 2  
3. jnz 7  
4. sti 1  
5. sta 1  
6. jmp 2  
7. hlt

Suppose we set up memory with 0 in all words except

1: 6  
2: 3  
3: 1

Simulate the program P started at instruction 1 until it halts, and show the value of PC and AC before the execution of each instruction. Also, describe the final contents of memory.

<table>
<thead>
<tr>
<th>PC</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

halted

The final contents of the memory is zero except for

1: 3  
2: 3  
3: 1  
6: 3
8. Construct a Turing machine M using the alphabet b, 0, and 1, where b is the blank symbol, such that if M is started with its head on the rightmost symbol of a binary number representing the positive integer x, then it eventually halts with its head on the rightmost symbol of a binary number representing the integer (x+1).

Example: (initial tape, final tape, head position shown by ^)

```
b b b 1 0 1 1 b b b  =>  b b b 1 1 0 0 b b b
-----------------^-----          -----------------^-----
```

- Initial state: q0
- Halt state: q2

(q0,0) -> (q1,1,R)  q0 moves left, changing 1’s to 0’s
(q0,1) -> (q0,1,L)  until it finds a 0 or b, which it
(q0,b) -> (q1,1,R)  changes to a 1 and moves R to q1

(q1,0) -> (q1,0,R)  q1 moves right, copying 0’s and 1’s
(q1,1) -> (q1,1,R)  until it finds a b, when it moves L
(q1,b) -> (q2,b,L)  and halts
9. Consider the finite state machine, F, described as follows:

states: q0, q1, q2  
symbols: a, b  
initial state: q0  
accepting states: q1, q2  
transition function:

(q0,a) \rightarrow q1  \quad (q0,b) \rightarrow q0  
(q1,a) \rightarrow q2  \quad (q1,b) \rightarrow q1  
(q2,a) \rightarrow q0  \quad (q2,b) \rightarrow q1  

(a) Draw a (circles and arrows) diagram of F, indicating states, transitions, start state, and accepting states.

******* Drawn by hand in hard copy **********

(b) Does F accept or reject each of the following strings? Show the sequence of states F passes through for each string.

Example: (0) aab \quad \text{accept} \quad q0 \rightarrow q1 \rightarrow q2 \rightarrow q1

(1) a \quad \text{accept} \quad q0 \rightarrow q1

(2) abaa \quad \text{reject} \quad q0 \rightarrow q1 \rightarrow q1 \rightarrow q2 \rightarrow q0

(3) bbb \quad \text{reject} \quad q0 \rightarrow q0 \rightarrow q0 \rightarrow q0

(4) aaabba \quad \text{accept} \quad q0 \rightarrow q1 \rightarrow q2 \rightarrow q0 \rightarrow q0 \rightarrow q0 \rightarrow q1

(5) ababab \quad \text{accept} \quad q0 \rightarrow q1 \rightarrow q1 \rightarrow q2 \rightarrow q1 \rightarrow q2 \rightarrow q1
10. (Dreaded decidabilities)
Are the following problems decidable or undecidable?
Give a brief justification of your answer.

(a)
Input: a string $w$ and a regular expression $E$.
Output: 1 if $E$ matches $w$, 0 if $E$ doesn’t match $w$.

Decidable. There is an algorithm to convert $E$ to an equivalent
finite state machine $M$. Then decide whether $M$ accepts $w$. (Lecture 6)

(b)
Input: a Turing machine $T$ and a tape $t$.
Output: 1 if $T$ doesn’t halt on $t$, 0 if $T$ does halt on $t$.

Undecidable. This is the negation of the halting problem.
Suppose (not-halts? $T$ $t$) is a procedure to solve it. Write:

\[
\text{(define TM-halts?)}
\text{(lambda (T t)}
\text{(- 1 (not-halts? T t)))}
\]

The halting problem for $T$ and $t$ is undecidable (ref. TO 59)
so not-halts? must be undecidable, too.

(c)
Input: a Scheme procedure proc.
Output: 1 if (proc 0) and (proc 1) both halt, 0 otherwise.

Undecidable. Suppose (both-halts? proc) is a procedure to solve it. Write a procedure for the Scheme halting problem:

\[
\text{(define scm-halts?)}
\text{(lambda (proc x)}
\text{(let ((new-proc (lambda (n) (proc x)))}
\text{(both-halts? new-proc)))})
\]

If (proc $x$) halts, then new-proc halts on *every* input (including both 0 and 1), and if (proc $x$) doesn’t halt, then new-proc fails to halt on *every* input. Thus (both-halts? new-proc) returns 1 if (proc $x$) halts, and 0 if (proc $x$) fails to halt. Since scm-halts? is undecidable, so is both-halts.