YOUR NAME PLEASE:

**SOLUTIONS**

Computer Science 201
Exam 1
February 21, 2017
7-8:30 pm

Closed book and closed notes. No electronic devices. Show ALL work you want graded on the test itself. Page 13 has additional work space. There are 6 problems, worth 10 points each.

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!
1.a) (5 points)
Write a Racket procedure (count-greater n lst) that takes an integer n and a list of integers lst and returns the number of elements of lst that are strictly greater than n.

Use only the Racket procedures and special forms: +, >, length, define, lambda, empty?, first, rest, cons, append, list, equal?, if, cond, else, integer constants, and the quoted empty list '().

Do not define any procedures other than count-greater.

Examples:
(count-greater 3 '(4 5 3 4 4 3 0)) => 4
(count-greater -1 '(6 -5 0 -1 4)) => 3

(define (count-greater n lst)
  (cond
   [(empty? lst) 0]
   [ (> (first lst) n)
     (+ 1 (count-greater n (rest lst)))]
   [else
     (count-greater n (rest lst))]))
1. (b) (4 points)
Draw a tree of recursive calls, with return values, for the application (diffs '(2 3 8 14)), where the procedure diffs is defined as follows.

```
(define (diffs lst)
  (if (<= (length lst) 1)
      '()
      (cons (- (first (rest lst)) (first lst))
            (diffs (rest lst))))))
```

```
(diffs '(2 3 8 14)) => '(1 5 6)
    /
   Cons 1
    /
   Cons 5
    /
   Cons 6
    /
   Cons 14
```

1. (c) (1 point)
Is the procedure diffs in part 1(b) tail recursive?
Why or why not?

**No, because a value is cons'd onto the recursive call, (diffs (rest lst)), before it is returned.**
2.(a) (6 points)
Consider the Boolean function \( f \) presented by the following truth table:

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
<th>( f(x,y,z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Write down a Boolean expression to represent \( f(x,y,z) \) using the variables \( x, y, z \).
(Please use dot for AND, plus for OR and prime for NOT.)

Justify the correctness of your expression.

\[ x' \cdot y' \cdot z + x' \cdot y \cdot z' + x \cdot y' \cdot z' \]

Correct by the Sum-of-Products algorithm.

2.(b) (4 points)
Give a Boolean expression (using AND, OR, NOT as in part 2(a)) corresponding to each of the following statements involving the Boolean variables \( p, q, \) and \( r \).

You DO NOT need to justify the correctness of your answers.

(i) All three of \( p, q \) and \( r \) are false.

\[ p' \cdot q' \cdot r' \]
2. (b) continued.

(ii) The number of true variables among \( p, q, \) and \( r \) is either 1 or 2.

\[
(p'q'r' + p.q'r)'
\]

(or various equivalent expressions)

(iii) The variable \( p \) is true and at least one of the other variables is false.

\[
p \cdot (q' + r')
\]

(or various equivalent expressions)

(iv) The values of \( p \) and \( q \) are equal or the values of \( q \) and \( r \) are equal.
(Recall that "or" is inclusive.)

\[
p.q + p.q' + q'r + q'r'
\]

(or various equivalent expressions)
3. (a) (6 points)
Define a procedure (sublists n lst) that takes a positive integer n and a list lst of distinct integers, and returns a list of lists, containing every contiguous sublist of lst of length n.

Use only the Racket procedures and special forms: +, >, length, define, lambda, empty?, first, rest, cons, append, list, equal?, if, cond, else, integer constants, and the quoted empty list '().

You may write one or more auxiliary procedures -- please explain what they do.

Examples:
(sublists 3 '((4 7 2 9 1)) => '((4 7 2) (7 2 9) (2 9 1))
(sublists 2 '((9 4 7 0 9)) => '((9 4) (4 7) (7 0) (0 9))
(sublists 3 '((4 6 2)) => '((4 6 2))
(sublists 4 '((3 9 8)) => '())

; Auxiliary procedure to return a list of the first k elements of lst -- assumes length of lst is at least k.
(define (initial k lst)
  (if (> k 0)
      (cons (first lst)
            (initial (+ k -1) (rest lst)))
      '()))

(define (sublists n lst)
  (if (> n (length lst))
      '()
      (cons
       (initial n lst)
       (sublists n (rest lst)))))))
3. (b) (4 points) Draw a tree of recursive calls for the application \( (\text{rec } 3) \), where the procedure \( \text{rec} \) is defined below. Give the final value of the application; you do not need to give intermediate values.

\[
(\text{define } (\text{rec } n)) \\
(\text{if } (= n 1) \\
\quad '1) \\
(\text{append}) \\
(\quad (\text{rec } (- n 1)) \\
(\quad (\text{cons } n (\text{rec } (- n 1))))))
\]

\[
(\text{rec } 3) \Rightarrow '1(1 \ 2 \ 3 \ 1 \ 2 \ 1)
\]

\[
\text{append} \ (\text{rec } 2) \ (\text{cons } 3 (\text{rec } 2)) \Rightarrow '1(3 \ 1 \ 2 \ 1)
\]

\[
\text{append} \ (\text{rec } 1) \ (\text{cons } 2 (\text{rec } 1)) \\
\Rightarrow '1(1) \Rightarrow '1(1) \Rightarrow '1(1)
\]

\[
\text{append} \ (\text{rec } 1) \ (\text{cons } 2 (\text{rec } 1)) \\
\Rightarrow '1(1)
\]
4. For each of the three Turing machines below do the following.

Show the configuration of the machine after each of the first four steps, starting with the given initial configuration. Determine whether the given machine will eventually halt when started in the given initial configuration, giving reasons for your answer. (Recall that b is the blank symbol.)

4.(a) (3 points)
M1 is given by instructions:

(q1, x, q1, y, R), (q1, y, q1, x, R), (q1, b, q2, x, L),
(q2, y, q2, y, R)

The initial configuration for M1:
1) b y y x b

\[ q_1 \]

2) b x y x b

\[ q_1 \]

3) b x x x b

\[ q_1 \]

4) b x y y b

\[ q_1 \]

5) b x y y x b

\[ q_2 \]

This computation halts after the next step, because the next configuration is:
6) b x x y x b

\[ q_2 \]

and no instruction is defined for current state = \( q_2 \), current symbol = x
4. (b) (3 points)
M2 is given by the instructions:

\[(q_1, x, q_2, x, R), (q_1, y, q_3, y, R),
(q_2, x, q_2, x, R), (q_2, y, q_2, x, R), (q_2, b, q_3, b, L),
(q_3, x, q_3, y, R), (q_3, y, q_3, y, R), (q_3, b, q_1, b, L)\]

The initial configuration for M2:

1. \(bxyyb\)
2. \(bxyyb\)
3. \(bxyyb\)
4. \(bxyyb\)
5. \(bxyyb\)
6. \(bxyyb\)
7. \(bxyyb\)
8. \(bxyyb\)

This computation does not halt, because configuration (8) repeats configuration (6).

4. (c) (4 points)
M3 is given by the instructions:

\[(q_1,b,q_2,1,R), (q_1,1,q_2,1,L),
(q_2,b,q_1,1,L), (q_2,1,q_3,1,R)\]

The initial configuration for M3 is:

1. \(b\)
2. \(b\)
3. \(b\)
4. \(b\)
5. \(b\)
6. \(b\)
7. \(b\)

This computation halts in configuration (7) because no instructions are defined for current state = 83.
5. (10 points, 2 points for each part)

5.(a) State the theorem we proved in class about the Halting Problem for Racket programs. (Note: DO NOT prove the theorem, just state it.)

There can be no procedure \( \text{halts? proc expr} \) such that

\[
(\text{halts? proc expr}) \Rightarrow \begin{cases} 
\#t & \text{if } (\text{proc expr}) \text{ halts} \\
\#f & \text{otherwise} 
\end{cases}
\]

5.(b) Give an example of the cd command in Linux, and briefly describe what it does.

```plaintext
> cd cs201
```

changes the current working directory to cs201

5.(c) Give a Racket expression, using only cons, numbers, and the quoted empty list \'(\), that evaluates to \'(1 (2) 3)\).

```
(cons 1 (cons (cons 2 '(()) (cons 3 '())))
```

5. continued.

5.(d) For the following procedure find the values of \( f(1) \), \( f(2) \), and \( f(3) \). Rewrite the procedure using the special form let so that there is exactly ONE recursive call \( f(-n\,1) \).

\[
\begin{align*}
\text{(define } (f\,n) \quad & (f\,1) \Rightarrow 2 \\
\text{ (if } (=\,n\,1) \quad & (f\,2) \Rightarrow \!(1\,(2\,2)) \\
\quad 2 \quad & (f\,3) \Rightarrow \!(1\,(2\,2)\,(2\,2)) \\
\text{ (list } (f\,(-\,n\,1)) \quad & \text{define } (f\,n) \\
\quad (f\,(-\,n\,1)))))) \quad & \text{(if } (=\,n\,1) \\
\text{ )} \quad & 2 \\
\text{ )} \quad & (let \ [[(\text{prev } (f\,(-\,n\,1)))]] \\
\text{ )} \quad & (\text{list } \text{prev prev prev}))
\end{align*}
\]

5.(e) Consider the following procedure with positive integer arguments \( x \) and \( y \). On which pairs of arguments will it halt, and, when it does halt, what is the returned value in terms of the input values of \( x \) and \( y \)?

\[
\begin{align*}
\text{(define } (\text{what } x\,y) \quad & \text{(cond} \\
\text{ [ (= x y) (list x y)]} \quad & \text{[(> x y) (\text{what } (- x\,1)\,(+ y\,1))]} \\
\text{ [else (what } (+ x\,2)\,(+ y\,1))])}
\end{align*}
\]

This always halts. The returned value is a list \( (z\,z) \), where

1. If \( x = y \) then \( z = x = y \)
2. If \( x < y \) then \( z = 2y - x \)
3. If \( x > y \) and \( x - y \) is even, then \( z = \frac{x + y}{2} \)
4. If \( x > y \) and \( x - y \) is odd, then \( z = \frac{x + y + 3}{2} \).

11
6. (10 points, 2 points for each part)
Evaluate the following Racket expressions.
(Remember to show your work to enable partial credit.)

Example: \( (\text{cons } 1 '(6 5)) \Rightarrow '(1 6 5) \)

(a) \( (\text{append } '(1 2) '(3 4)) \Rightarrow '(1 2 3 4) \)

(b) \( (\text{list } (+ 1 3) (* 4 2) (- 11 5)) \Rightarrow '(4 8 6) \)

(c) \( (\text{map length } '((3 (2 1)) 4) ((())) ((6 5)))) \Rightarrow '(2 1 1) \)

(d) \( (\text{let } ((\text{lst1 } '(1 2 3 4 5 6 7 8 9)))
           (\text{let } ((\text{lst2} (\text{filter} (\text{lambda} (x) (\text{not} (= 0 (\text{remainder} x 3))))
                                \text{lst1})))
           (\text{map} (\text{lambda} (y) (- y 1)) \text{lst2})) \Rightarrow '(0 1 3 4 6 7) \)

(e) \( (\text{let } ((p (\text{lambda} (q) (\text{lambda} (x) (q (q x)))))
           (r (\text{lambda} (y) (+ y y))))
           (\text{list} (r 5) ((p r) 3)) \Rightarrow '(16 12) \)

//