YOUR NAME PLEASE: ******* SOLUTIONS *******

[illegible] 4/3/15

Computer Science 201
Practice Second Exam
(90 minutes)

Closed book and closed notes. Show ALL work you want graded on the test itself.

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. If a problem asks for a Racket procedure, you *may* define auxiliary procedures; please describe what they do.

GOOD LUCK!

1.(a) (6 points)
We define a "plus-times expression" recursively as follows.
A Racket number is a plus-times expression.
A list containing three elements is a plus-times expression provided
(1) the first element is a plus-times expression,
(2) the second element is the symbol 'plus or the symbol 'times, and
(3) the third element is a plus-times expression.

Write a Racket procedure (pt-exp? value) that takes an
arbitrary Racket value and returns #t if it is a plus-times
expression according to this definition, or #f it is not.

Examples:

```
(pt-exp? -13) => #t
(pt-exp? 'apple) => #f
(pt-exp? '(times 3 4)) => #f
(pt-exp? '(2 plus #t)) => #f
(pt-exp? '((3 plus 2) times 4)) => #t
(pt-exp? '((2 times 3) plus (4 times (1 plus 3)))) => #t
```

```
(define (pt-exp? value)
  (cond
    [(number? value)
     #t]
    [(or (not (list? value))
         (not (= 3 (length value))))
     #f]
    [else
     (and
       (pt-exp? (first value))
       (member (second value) '(plus times))
       (pt-exp? (third value)))]))
```

1.(b) (6 points)
Write a Racket procedure (evaluate exp) that
takes a plus/times expression exp as defined in 1.(a)
and returns its value, interpreting plus as addition
and times as multiplication.

Examples:
```
(evaluate -13) => -13
(evaluate '((3 plus 2) times 4)) => 20
(evaluate '((2 times 3) plus (4 times (1 plus 3)))) => 22
```

```
(define (evaluate exp)
  (if (number? exp)
      exp
      (let ((op (if (equal? 'plus (second exp)) + *)))
        (op
          (evaluate (first exp))
          (evaluate (third exp))))))
```

2.(a) (6 points)
Draw a combinational circuit with
inputs r, a, b and outputs x, y that computes the following.
If r = 0 then x = a and y = b; if r = 1 then x = b and y = a.

The available types of gates are: NOT and 2-input AND, OR, XOR.
Make sure you label the input and output wires of your circuit,
and label your NOT, AND, OR and XOR gates (which can be represented
by rectangles with the correct labels.)

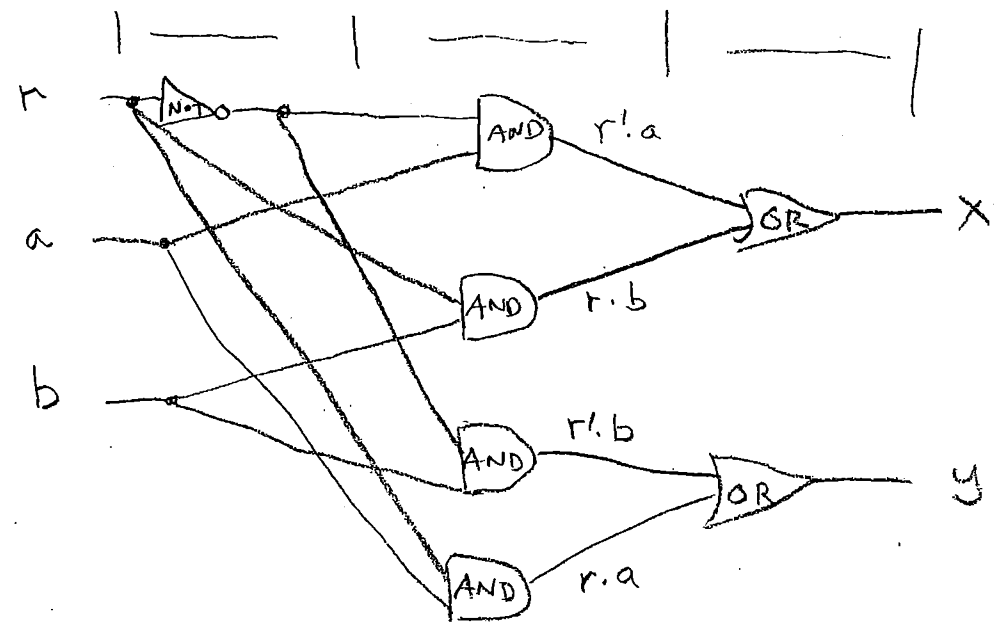
Give a brief argument for the correctness of your circuit
and state how many gate delays it requires to compute its outputs.

x = r'*a + r*b
y = r'*b + r*a

If r = 0 then r' = 1, and x = 1*a + 0*b = a, and y = 1*b + 0*a = b
If r = 1 then r' = 0, and x = 0*a + 1*b = b, and y = 0*b + 1*a = a

(Or a truth table and sum-of-products, possibly with simplifications.)

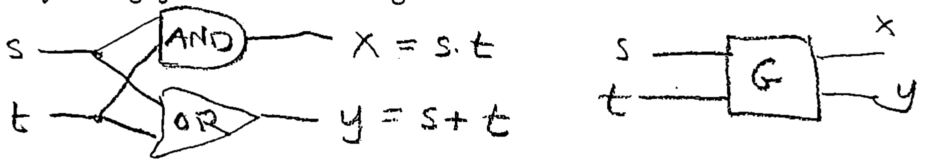
The circuit requires 3 gate delays.



2.(b)
Here is the truth table of a gate G with inputs s and t and outputs x and y.

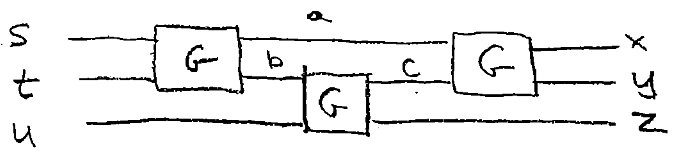
s	t	x	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

(i) (2 points) Show that G can be implemented by a circuit with one gate delay using just AND and OR gates.



(ii) (4 points) Show that three copies of G, and no other gates, can be used to implement a circuit with three inputs s, t, u and three outputs x, y, z such that sorting the values of s, t, u into increasing order gives the values of x, y, z. Justify the correctness of your circuit.

For example, if s = 1, t = 1, u = 0, then x = 0, y = 1, z = 1.



s	t	u	a	b	c	x	y	z
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	1	0	0	0	1
0	1	1	0	1	1	0	1	1
1	0	0	0	1	0	0	0	1
1	0	1	0	1	1	0	1	1
1	1	0	1	1	0	0	1	1
1	1	1	1	1	1	1	1	1

Here are the final configurations for all the wires for every possible input s, t, u. Note that x, y, z gives the values of s, t, u sorted order.

3. Recall that the TC-201 assembly-language instructions are:
halt, load address, store address, add address, sub address,
input, output, jump address, skipzero, skippos, skiperr,
loadi address, storei address

Also the directive: data number
reserves one memory location and stores the number in it.

3.(a) (8 points)
Write a TC-201 assembly-language program to read in two numbers from the user, print out the maximum of the two, and halt. Please use symbolic opcodes and addresses. You may assume that the user's numbers are in the range of -1000 to 1000.

For example, one run of your program might be:

```
input = 33
input = -64
output = 33
```

```
        input
        store x
        input
        store y
        sub   x
        skippos        skips if (y-x) > 0, i.e., y > x
        jump  printx
printy: load y
        output
        halt
printx: load x
        output
        halt
x:      data 0
y:      data 0
```

3.(b) (4 points)
Briefly describe each of the following

(i) 16-bit sign/magnitude representation of numbers

The leftmost (high-order) bit is the sign: 1 for - and 0 for +. The remaining bits give the magnitude (absolute value) of the number in 15-bit unsigned binary.

(ii) the program counter

The program counter is a register that holds the address of the next instruction to be executed.

(iii) the arithmetic error bit of the TC-201

The arithmetic error bit is set to 1 if an add or sub instruction produces a result that cannot be represented in 16-bit sign/magnitude representation; otherwise, add or sub sets it to 0. It is tested by the skiperr instruction, which skips if it is 1 and does not skip if it is 0. It is set to 0 by the skiperr instruction.

(iv) the fetch/execute cycle

If the run flag is 1, the next instruction is fetched from the memory register whose address is in the program counter. The opcode and address of the instruction are interpreted, and the configuration of the machine is changed accordingly, including updating the program counter.

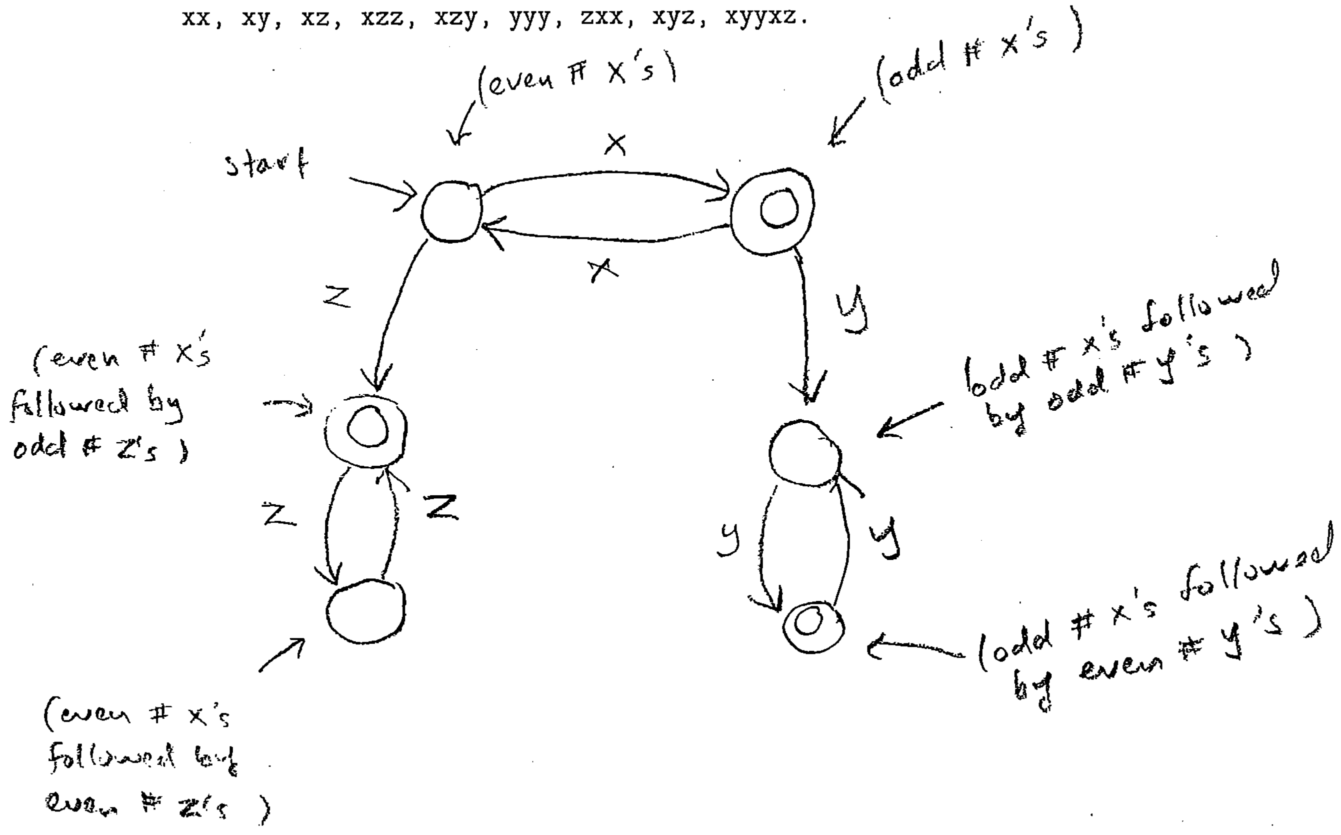
```
4.(a) (6 points)
Let the alphabet be {x, y, z} and write a regular expression
for each of the following languages over this alphabet.
Please use **only** the operations: or, concatenation, Kleene star.


(i) All strings over the given alphabet.

    (x|y|z)*


(ii) All strings consisting only of an odd number of x's,
with no y's or z's.

    x(xx)*


(iii) All strings not containing any y's.

    (x|z)*


(iv) All strings containing exactly zero or one occurrences of z.

    (x|y)* | (x|y)*z(x|y)*


(v) The strings xyz, yzx, zxy

    (xyz | yzx | zxy)


(vi) All strings in which no z has a y anywhere to its right.

    (x|y)* | (x|y)*z(x|z)*

    (or, more simply: (x|y)*(x|z)*)
```

4.(b) (6 points)
Give a (possibly incomplete) deterministic finite state acceptor for the following language L. A diagram is fine. Be sure to indicate the start state, the accepting state(s), and the transitions.

The alphabet is {x, y, z} and L is the set of all strings which are either
(1) an odd number of x's followed by an even number of y's,
or
(2) an even number of x's followed by an odd number of z's.

(Note that zero is an even number.)

Examples of strings in the language:
x, xyy, xxx, xxxyyyy, xxzzz, z.
Examples of strings not in the language:
xx, xy, xz, xzz, xzy, yyy, zxx, xyz, xyyxz.



(even # x's)
(odd # x's)
start
x
x
z
y
(even # x's followed by odd # z's)
(odd # x's followed by odd # y's)
z
z
y
y
(odd # x's followed by even # y's)
(even # x's followed by even # z's)

5. Give brief answers to the following questions (2 points each):

5.(a) Explain the difference between the TC-201 instructions "load 17" and "loadi 17".

load 17 copies the contents of memory register 17 to the accumulator. loadi 17 takes the rightmost (low-order) 12 bits of the contents of memory register 17 as a memory address, and copies the contents of that memory register to the accumulator.

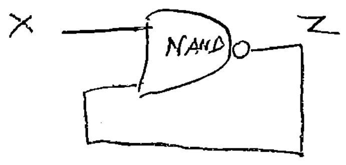
5.(b) Explain what will be printed out in response to the Linux command

```
> grep -E "c(a|d)r" temp.txt
```

This prints out every line of the file temp.txt that contains "car" or "cdr" as a substring.

5.(c) Consider a circuit consisting of one NAND gate and two wires, x and z. The wire z is the output of the NAND gate and also one of its inputs. The other input of the NAND gate is x. For each of the four possible configurations of this circuit, determine whether it is stable.

x	z	stable?
0	0	No (z -> 1)
0	1	Yes
1	0	No (z -> 1)
1	1	No (z -> 0)



5.(c) What is the essential difference between the Halting Problem and the question of whether a given circuit will eventually reach a stable configuration from a given configuration?

In the Halting Problem, a program may run for an arbitrarily large number of steps without repeating a configuration, by using more and more memory. In a circuit, this is not possible: with n wires there are 2^n possible configurations, so after 2^n steps, it must repeat a configuration.

5.(e) How many different 2-argument Boolean functions f(x,y) can you construct using just NOT and XOR gates? Please show your work. Three are listed below.

x	y	x	y	x'	y'	(x XOR y)	(x XOR y)'	(x XOR x)	(x XOR x)'
0	0	0	0	1	1	0	1	0	1
0	1	0	1	1	0	1	0	0	1
1	0	1	0	0	1	1	0	0	1
1	1	1	1	0	0	0	1	0	1

There are just 8 (of the possible 16) different Boolean functions of 2 inputs achievable with XOR and NOT. This means {XOR, NOT} is not a Boolean basis.

5.(e) If L is a language, let reverse(L) be the set of reverses of all strings in L. If L is a regular language, must reverse(L) be a regular language? Why or why not?

Yes, if L is a regular language, then reverse(L) is also a regular language. To see this, suppose L is regular, with regular expression E. We show how to construct a regular expression R(E) for the reverse language recursively as follows. If E is epsilon (the empty string), R(E) is also epsilon. If E is a single symbol b, then R(E) is the same single symbol b. If E is (E1|E2), then R(E) is (R(E1)|R(E2)). If E is (E1)*, then R(E) is (R(E1))*. Finally, if E is E1 concatenated with E2, then R(E) is R(E2) concatenated with R(E2). (Note the opposite order.) Because R(E) is a regular expression for reverse(L), we conclude that reverse(L) is regular.