YOUR NAME PLEASE:

SOLUTIONS DA 4/11/17

Computer Science 201
Second Exam
April 11, 2017
7-8:30 pm

Closed book and closed notes. Show ALL work you want graded on the test itself. Use the backs of pages as additional work space.

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. We define a "number-tree" data structure recursively as follows.
Base case: a Racket number by itself is a number-tree.
Recursive case: if T1 and T2 are number-trees, then there is a
number-tree T whose left subtree is T1 and whose right subtree is T2.

This is reflected in the following Racket struct definition:

```
(struct nt (left right) #:transparent)
```

This defines a constructor nt, a type predicate nt?, and selectors
nt-left and nt-right, for the left and right subtrees.
The Racket predicate (number? value) tests whether a value is a number.

Examples of number-trees:

```
(define t1 44)
(define t2 (nt 2 33))
(define t3 (nt (nt 13 4) (nt (nt 6 39) 52)))
```

1.(a) (6 points)
Write a *single* Racket procedure (number-tree? value) that takes an
arbitrary Racket value and returns #t if value is a number-tree according
to the definition above, and #f otherwise.

Examples:

```
(number-tree? t1) => #t  (and similarly for t2 and t3 defined above)
(number-tree? '(2 13)) => #f
(number-tree? (nt (nt 17 'leaf2) 51)) => #f
```

```
(define (number-tree? value)
  (cond
    [(number? value)
     #t]
    [(nt? value)
     (and (number-tree? (nt-left value))
          (number-tree? (nt-right value)))]
    [else
     #f]))
```

1.(b) (3 points)
Write a *single* Racket procedure (min-leaf tree) that takes a number-tree and returns the minimum value of any number that appears in the tree.
Examples:
(min-leaf 23) => 23
(min-leaf (nt 27 14)) => 14
(min-leaf (nt (nt (nt 13 6) (nt 2 0)) 3)) => 0

```
(define (min-leaf tree)
  (if (number? tree)
      tree
      (min (min-leaf (nt-left tree))
           (min-leaf (nt-right tree)))))
```

1(c) (3 points)
Write a *single* Racket procedure (compute proc tree) that takes a procedure proc and a number-tree as arguments. You may assume proc returns a number when applied to two numbers as arguments. The value of (compute proc tree) is defined recursively: if tree is just a number, the number is returned; otherwise, proc is applied to the results of applying compute (with proc) to the left and right subtrees of tree.
Examples:
(compute + 137) => 137
(compute + (nt (nt 4 (nt 5 8)) (nt 10 15))) => 42
(compute min (nt (nt 2 4) (nt 3 5))) => 2
(compute - (nt (nt 2 4) (nt 3 6))) => 1

```
(define (compute proc tree)
  (if (number? tree)
      tree
      (proc (compute proc (nt-left tree))
            (compute proc (nt-right tree)))))
```

2.(a) (6 points)
Construct a combinational (that is, loop-free) circuit to compute the outputs d and e from the inputs a, b and c, given the following truth table:

a	b	c	d	e
0	0	0	0	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

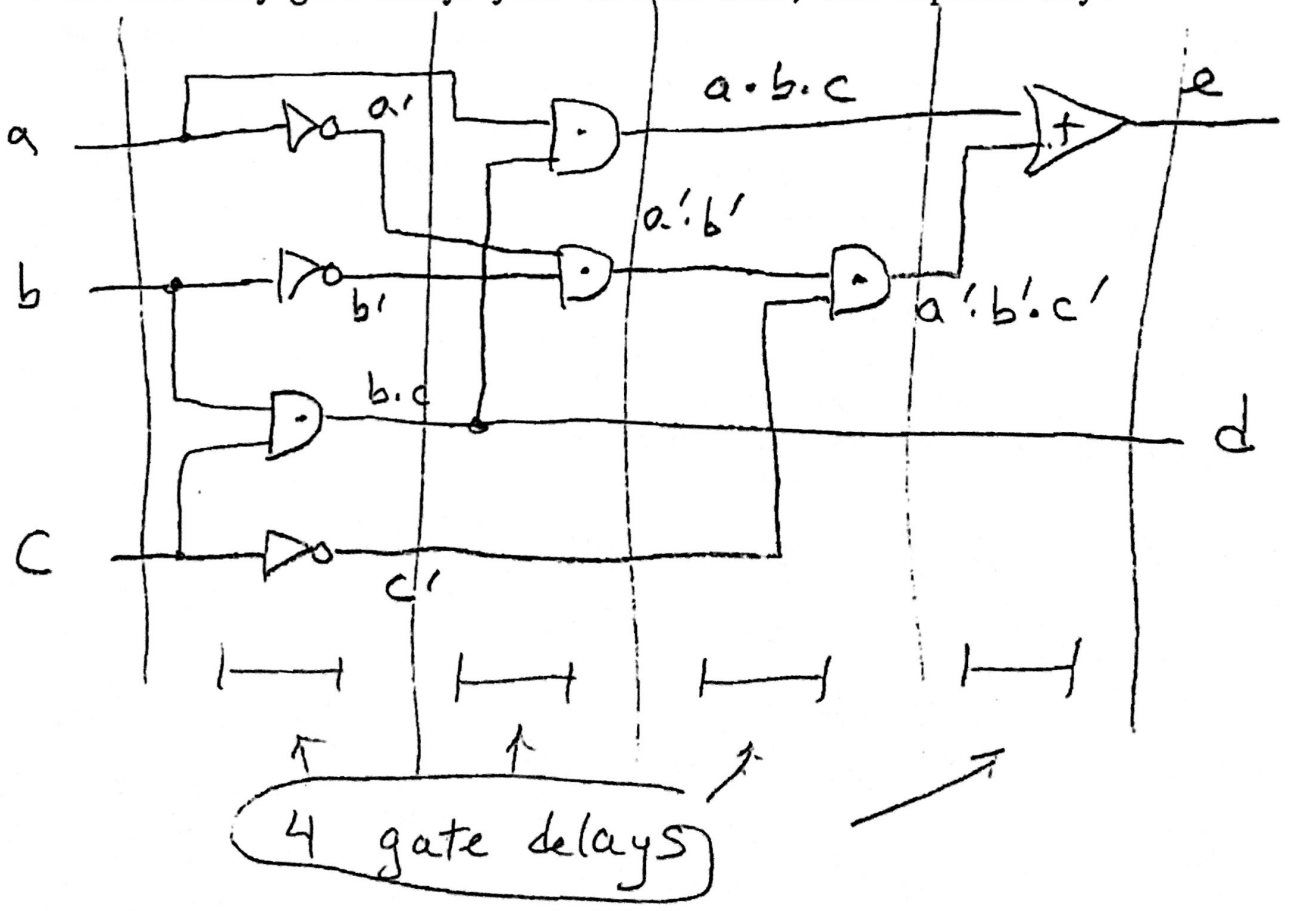
$d = a' \cdot b \cdot c + a \cdot b \cdot c \equiv b \cdot c$

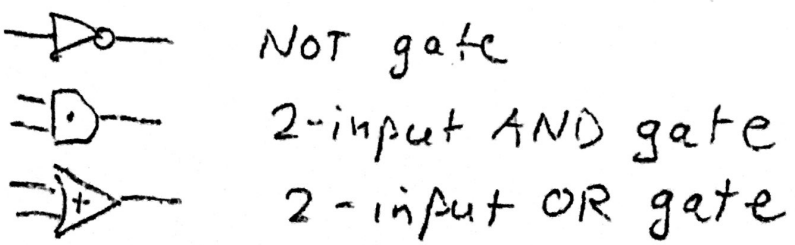
$e = a' \cdot b' \cdot c' + a \cdot b \cdot c$

[by sum-of-products]

Use only the following types of gates: NOT, 2-input AND, 2-input OR.
You may use rectangles (labeled with gate types) for gates.
Make sure your input and output wires are labeled.

State how many gate delays your circuit uses, and explain why.



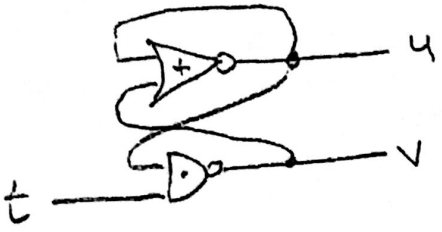


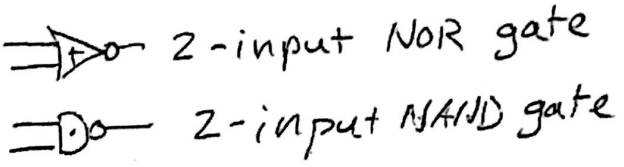
2.(b) (4 points)
Consider the sequential (that is, "loopy") circuit with wires t, u, v, and two gates arranged so that

u = NOR(u,v)
v = NAND(u,t)

x	y	NOR(x,y)	NAND(x,y)
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

Draw a diagram of this circuit (rectangles with labels indicating the gate type are fine.) Also, find all the stable configurations of its three wires, and justify your answer.





(1) In any configuration where $u = 1$, after one delay, $u = 0$ because $0 = NOR(1,0) = NOR(1,1)$, so no such configuration is stable.

(2) If $u = 0$, then after one delay, $v = 1$ because $1 = NAND(0,0) = NAND(0,1)$, so the only stable configurations must have $u = 0$ and $v = 1$. This is stable for $t = 0$ or $t = 1$ because $0 = NOR(0,1)$, $1 = NAND(0,0)$ and $1 = NAND(0,1)$.

Stable:

t	u	v
0	0	1
1	0	1

2.(c) (2 points)
Is {AND, OR} a Boolean basis? Why or why not?

No, it is not a Boolean basis because we cannot make a circuit for x' using just AND and OR. This is because $x \cdot x \equiv x$ and $x + x \equiv x$, so no matter how we combine the input x with AND and OR gates, we can only get x.

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 a zero-terminated sequence of numbers, print out two numbers: the sum of the positive numbers in the sequence, and the sum of the negative numbers in the sequence, and then halt. You may ignore the possibility of arithmetic error. If there are no positive numbers in the sequence, the first output number should be zero, and if there are no negative numbers in the sequence, the second output number should be zero.

Please use symbolic opcodes and symbolic (not numeric) addresses.
An example of the behavior of your program:

```
input = 17
input = -4
input = -2
input = 3
input = -5
input = 0
output = 20
output = -11
```

```
read-next: input
           skipzero
           jump choose
           load pos-sum
           output
           load neg-sum
           output
           halt
choose:    skippos
           jump negative
           add pos-sum
           store pos-sum
           jump read-next
negative:  add neg-sum
           store neg-sum
           jump read-next
pos-sum:   data 0
neg-sum:   data 0
```

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

(i) 8-bit one's complement representation of integers.

If first bit is 0, value is the unsigned binary value of remaining 7 bits. If first bit is 1, value is unsigned binary of the result of complementing each of the remaining 7 bits.

(ii) The TC-201 program counter (pc).

The program counter is a 12 bit register in the CPU that holds the address of the next instruction to be executed.

(iii) The fetch-execute cycle.

The process of fetching from memory the instruction indicated by the program counter, decoding and executing the instruction, and appropriately updating the program counter.

(iv) What the TC-201 loadi instruction does.

loadi address takes the contents of the memory register at address and uses its low-order 12 bits to determine another address, and copies the contents of the memory register at this second address to the accumulator.

4.(a) (6 points)
Consider the alphabet of symbols {w, x, y, z}.
Write a "basic regular expression" for each of the following sets of strings over this alphabet. Recall that in a basic regular expression we have operations for union (|), repetition 0 or more times (*), and concatenation (centered dot).

(i) The set of strings consisting of 0 or more x's.

x^* [Note: "consisting of", not "containing"]

(ii) The set of strings of length exactly 2.

$(w|x|y|z)\cdot(w|x|y|z)$

(iii) The set of strings that do not contain all four symbols: w, x, y, z.

$(w|x|y)^* \mid (w|x|z)^* \mid (w|y|z)^* \mid (x|y|z)^*$

(iv) The set of strings in which every w is immediately followed by at least two x's, and every y is immediately followed by at least two z's.

$(x|z|wxx|yzz)^*$

(v) The set of strings that do not have a w anywhere to the right of a z.

$(w|x|y)^*(x|y|z)^*$

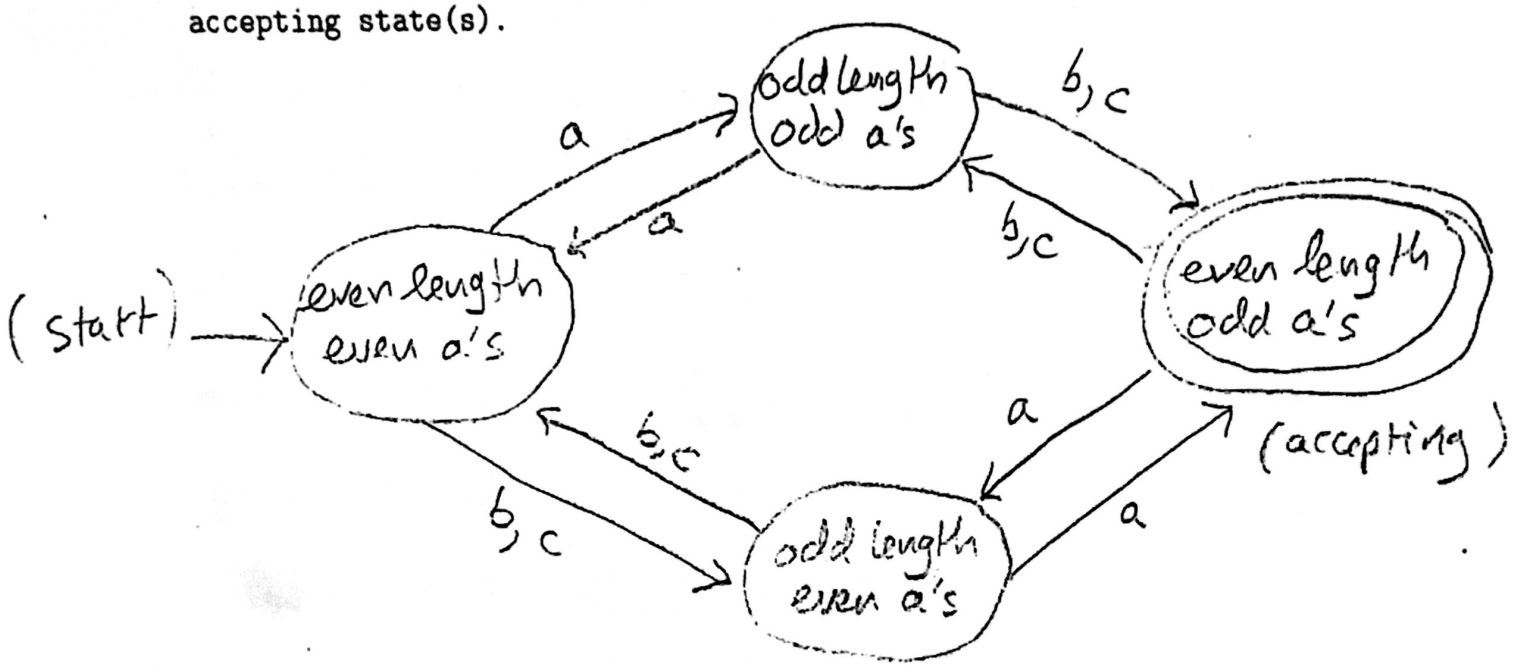
(vi) The set of strings in which the number of x's is a multiple of 3. (Note that 0 is a multiple of 3 because 0 = 0 * 3.)

$(w|y|z)^*(x(w|y|z)^*x(w|y|z)^*x(w|y|z)^*)^*$

4.(b) (6 points)
Let the alphabet of symbols be {a, b, c}. Construct a complete deterministic finite state acceptor for the set L of all strings over this alphabet that have even length and an odd number of a's.

Examples of strings in L: ccca, babc, bbbaaacc, abcabcabcb
Examples of strings not in L: empty string, a, b, c, aabb, ccccaaa, abc

Recall that a complete deterministic finite state acceptor has exactly one transition (arrow) defined for every state and symbol. A diagram is fine. Be sure to indicate the start state and the accepting state(s).



5. Give brief answers to the following questions (3 points each):

5.(a) Give the 16-bit two's complement representations of two numbers, X and Y, such that subtracting Y from X in the TC-201 would cause the arithmetic error bit (aeb) to be set, and explain why.

X = 1000 0000 0000 0000
Y = 0000 0000 0000 0001

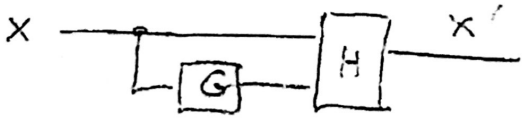
X is the smallest number representable in 16-bit two's complement, −32,768.
Y is 1. So X−Y is −32,769, which is not representable in 16-bit two's complement. Thus, subtracting Y from X would set the aeb.

5.(b) Suppose you have gates G(x) and H(x,y) given by the truth tables below. Is {G, H} a Boolean basis? Why or why not?

x	G(x)
0	0
1	0

x	y	H(x,y)
0	0	1
0	1	1
1	0	0
1	1	1

Yes, {G, H} is a Boolean basis. While neither gate alone gives NOT, they can be combined to give NOT:



X	H(x,G(x))
0	1
1	0

Now consider

$$H(x', y) \equiv H(H(x, G(x)), y)$$

X	y	H(x',y)
0	0	0
0	1	1
1	0	1
1	1	1

This is OR, so we can get {NOT, OR}, which is a Boolean basis.

5.(c) Is it possible to write a Racket program (halts? config) that takes a TC-201 configuration and returns #t if the TC-201 would eventually halt, and #f otherwise? (For any input instructions, assume that the user types 0 in response to every input prompt.)
Please justify your answer.

Yes, it is possible. A configuration of the TC-201 can be represented by $B = (4096 \cdot 16 + 16 + 12 + 2)$ bits. Thus we could use the TC-201 simulator with a limit of $2^B + 1$ steps. If the machine halts before this, we answer #t. Otherwise, the machine must have repeated some configuration and is in an infinite loop, so we can answer #f.

5.(d) Let L1 a regular language over an alphabet A. Define L2 to be the set of all strings over the alphabet A that are in L1 and have odd length.

Must L2 be a regular language? Why or why not?

Yes L2 must be regular. Because L1 is regular, there is a deterministic finite acceptor (dfa) recognizing L1, say M. We make a second copy, M', of M and then combine M and M' into a new dfa M2 that recognizes L2 as follows. The states of M2 are all the states q of M and all the states q' of M'. The start state of M2 is the start state of M. The accepting states of M2 are the accepting states of M'. The transitions $q_i \xrightarrow{a} q_j$ and $q_i' \xrightarrow{a} q_j'$ of M and M' are replaced by $q_i \xrightarrow{a} q_j'$ and $q_i' \xrightarrow{a} q_j$. Even length strings lead to states in M, odd length strings to states in M'. Thus L2 is regular.