

CSC 404 - EXAM 1 - NAME:

Problem 1 (01 and 001 - weee!). For each of the following, construct a finite automaton that recognizes the given language. Then, write the language via regular expressions, implement, and test against the given sets.

a. Bit-strings that contain the substring 01.

Accept: 0001, 001101, 00011001, 1001010 Reject: 0000, 1000, 111110

b. Bit-strings that contain exactly one copy of the substring 01.

Accept: 01, 0011, 0001110, 1000111000, 11110100 Reject: 0101, 0111011, 1101101001

c. Bit-strings that begin and end with different bits.

Accept: 101010, 000101, 01111 Reject: 10100111001, 01010, 00

d. Bit-strings that begin and end with different bits and have even length if it starts with a 1 and odd length if it starts with a 0.

Accept: 101010, 0001101, 00111 Reject: 10100111001, 01010, 010101, 100

e. (Bonus) Bit-strings that contain exactly one copy of the substring 001.

(You can just do the regular expressions for this one :-))

Accept: 10001, 000110, 1000001101, 1101001010 Reject: 010, 0010011, 0001010010, 001001001

Problem 2 (Complements – You are awesome!). In the first part of this problem, we look at the question of seeing if two strings are bitwise complements of each other (i.e., 1s and 0s are swapped).

- a. Describe how a Turing machine, COMPLEMENT, would accept the string 1011#0100. Give a rough idea what your machine will do (you will design a state diagram below).

...	□	1	0	1	1	#	0	1	0	0	□	...
...	□	1	0	1	1	#	0	1	0	0	□	...
...	□	1	0	1	1	#	0	1	0	0	□	...
...	□	1	0	1	1	#	0	1	0	0	□	...
...	□	1	0	1	1	#	0	1	0	0	□	...
...	□	1	0	1	1	#	0	1	0	0	□	...
...	□	1	0	1	1	#	0	1	0	0	□	...

- b. Construct a state diagram for Turing machine COMPLEMENT. Then, implement the machine in turingmachine.io and test it :-).

Problem 3 (Different = awesome). Are strings different? That is, our goal is to design a Turing Machine that accepts strings that are different.

- a. Describe how a Turing machine, DIFFERENT, would accept the string 1011#10101. Give a rough idea what your machine will do (you will design a state diagram below).

...	□	1	0	1	1	#	1	0	1	0	1	□	...
...	□	1	0	1	1	#	1	0	1	0	1	□	...
...	□	1	0	1	1	#	1	0	1	0	1	□	...
...	□	1	0	1	1	#	1	0	1	0	1	□	...
...	□	1	0	1	1	#	1	0	1	0	1	□	...
...	□	1	0	1	1	#	1	0	1	0	1	□	...
...	□	1	0	1	1	#	1	0	1	0	1	□	...

- b. Construct a state diagram for Turing machine DIFFERENT. Then, implement the machine in turingmachine.io and test it :-). Make sure to test a few ‘interesting’ edge cases.

Problem 4 (Parentheses Nesting – Ah the struggle of finding a missing (or)). In this problem, we explore a classic issue – matching left and right parentheses :-/. (The ability to match parentheses is something finite automaton and regular expressions cannot handle!).

- a. Describe how a Turing machine, PARENTHESESNESTING, would accept the string (((()())))

...	□	((()	())	())	□	...
...	□	((()	())	())	□	...
...	□	((()	())	())	□	...
...	□	((()	())	())	□	...
...	□	((()	())	())	□	...

- b. Describe how a Turing machine, PARENTHESESNESTING, would reject the string ())(

...	□	())	(□	...
...	□	())	(□	...
...	□	())	(□	...

- c. Construct a state diagram for Turing machine PARENTHESESNESTING. Then, implement the machine in `turingmachine.io` and test it.

Problem 5 (Ledgers or Ladders?!). A ledger starts from 0 and gains one dollar for each + and loses one dollar for each -. Reading left to right, check that the account never goes into the negative. For example,

Accept: ++++, +---, +++---, +++---+--- Reject: +--, ---, +++---+, +++---++

(Note: there is a reason this problem is right after parentheses nesting – they are very similar problems!)

- a. Describe how a Turing machine, LEDGER, would accept the string +++---+.

...	□	+	+	-	+	-	-	+	□	...
...	□	+	+	-	+	-	-	+	□	...
...	□	+	+	-	+	-	-	+	□	...
...	□	+	+	-	+	-	-	+	□	...

- b. Describe how a Turing machine, LEDGER, would reject the string +++---+ (negative after the 5th input).

...	□	+	+	-	-	-	+	□	...
...	□	+	+	-	-	-	+	□	...
...	□	+	+	-	-	-	+	□	...
...	□	+	+	-	-	-	+	□	...

- c. Construct a state diagram for Turing machine PARENTHESESNESTING. Then, implement the machine in `turingmachine.io` and test it.

Problem 6 (Three Equal Lengths – Weee!). Consider the (non-regular) language of all strings of 0s followed by an equal number of 1s and then an equal number of 2s,

$$L = \{012, 001122, 000111222, 000011112222, \dots\} = \{0^k 1^k 2^k \mid k = 0, 1, 2, \dots\}$$

- a. Describe how a Turing machine would accept the string 000001111122222

...	□	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	□	...
...	□	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	□	...
...	□	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	□	...
...	□	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	□	...
...	□	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	□	...
...	□	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	□	...

- b. Construct a state diagram for Turing machine, THREEEQUALLENGTHS. Then, implement the machine in turingmachine.io and test it. (Make sure it rejects the last example in the Reject List - this is the one most often missed :-))

Accept: 000011112222, 000001111122222

Reject: 00112, 00122, 001122012

- c. (Bonus) Most *initial* machines decide this language in $\mathcal{O}(n^2)$. Extending the ideas from Section 2.3 (crossing off every other 0 and 1) construct a machine that decides this language in $\mathcal{O}(n \log_2(n))$. If you already did this in parts a and b – yay, I guess you can use this space to draw a sweet picture!

Problem 7 (Bonus).

- a. How are things going in the course? Provide some comments about how the course is going so far (e.g., Activities/Project, Homework, Notes, Videos, D2L).
- b. As we move into the Algorithms/Optimization and Information Theory/Cryptography stage of the course, we move away from the cute machine world and begin ‘actual’ implementation. Here you are free to work in any programming language. I will typically work in Python and C – are there other languages you would like to explore?

Problem 8 (Bonus). It has been a couple weeks since I spammed you with ‘River Crossing’ Problems – with your new found love of river puzzles have you created any fun scenarios of your own? (Wired/Strange rule sets are encouraged.)

Problem 9 (Bonus). Consider the (non-regular) language of all strings that contain at least one 0 followed by 1s then followed by 2s, where the number of 0s and 1s multiply to the number of 2s.i.e.,

$$L = \{012, 01122, 00122, 00112222, 00111222222, \dots\} = \{0^i 1^j 2^k \mid 1 \leq i \text{ and } ij = k\}$$

A Turing machine that decides L is

M = “On input string w :

1. Scan the input from left to right to determine whether it is a member of $0^*1^*2^*$ and reject if it is not
 2. Return the head to the left-hand end of the tape
 3. Cross off a 0 and scan to the right until a 1 occurs. Bounce between 1s and 2s, crossing off one of each until all 1s are gone. If all 2s have been crossed off and some 1s remain, reject
 4. Restore the crossed off 1s and repeat stage 3 if there is another 0 to cross off. If all 0s have been crossed off, determine whether all 2s also have been crossed off. If yes, accept; otherwise, reject
- a. Describe how a Turing machine, M , would accept the string 00111222222 ($i = 2, j = 3, ij = 2 \cdot 3 = 6 = k$)
- | | | | | | | | | | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|-----|
| ... | □ | □ | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | □ | □ | ... |
|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|-----|
- b. Describe how a Turing machine, M , would reject the string 0011222.
- | | | | | | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|---|---|---|-----|
| ... | □ | □ | 0 | 0 | 1 | 1 | 2 | 2 | 2 | □ | □ | ... |
|-----|---|---|---|---|---|---|---|---|---|---|---|-----|
- c. Construct a state diagram for Turing machine M .