Final Exam

Two page cheat sheet

Conflict Mon 8-11

Structure

- 20pt T/F
- 5 x 10pt problems
  - 1 x NP-hardness
  - 2 x Midterm 1
  - 2 x Midterm 2

1. Which of the following are a good English specifications of a recursive function that could possibly be used to compute the edit distance between two strings $A[1\,..\,n]$ and $B[1\,..\,n]$?

   - Yes ~~No~~ — $Edit(i, j)$ is the answer for i and j.
   - Yes ~~No~~ — $Edit(i, j)$ is the edit distance between $A[i]$ and $B[j]$.
   - Yes ~~No~~ —

$$Edit[i,j] = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ Edit[i-1, j-1] & \text{if } A[i] = B[j] \\ \overset{\text{min}}{\cancel{\max}} \left\{ \begin{array}{c} 1 + Edit[i, j-1] \\ 1 + Edit[i-1, j] \\ 1 + Edit[i-1, j-1] \end{array} \right\} & \text{otherwise} \end{cases}$$

   - Yes ~~No~~ — $Edit[1\,..\,n, 1\,..\,n]$ stores the edit distances for all prefixes.
   - ~~Yes~~ No — $Edit(i, j)$ is the edit distance between $A[i\,..\,n]$ and $B[j\,..\,n]$.
   - Yes ~~No~~ — $Edit[i, j]$ is the value stored at row i and column j of the table.
   - ~~Yes~~ No — $Edit(i, j)$ is the edit distance between the last i characters of A and the last j characters of B.
   - Yes ~~No~~ — $Edit(i, j)$ is the edit distance when i and j are the current characters in A and B.
   - ~~Yes~~ No — $Edit(i, j, k, l)$ is the edit distance between substrings $A[i\,..\,j]$ and $B[k\,..\,l]$.
   - Yes ~~No~~ — *[I don't need an English description; my pseudocode is clear enough!]*

(f) Suppose we want to prove that the following language is undecidable.

$$\textsc{Muggle} := \{\langle M\rangle \mid M \text{ accepts } \texttt{SCIENCE} \text{ but rejects } \texttt{MAGIC}\}$$

Professor Potter, your instructor in Defense Against Models of Computation and Other Dark Arts, suggests a reduction from the standard halting language

$$\textsc{Halt} := \{\langle M, w\rangle \mid M \text{ halts on inputs } w\}.$$

Specifically, suppose there is a Turing machine DETECTOMUGGLETUM that decides MUGGLE. Professor Potter claims that the following algorithm decides HALT.

<u>DECIDEHALT($\langle M, w\rangle$):</u>
  Encode the following Turing machine:
    <u>RUBBERDUCK(x):</u>
      run M on input w — halts
      if x = MAGIC
          return FALSE
      else
          return TRUE
  return DETECTOMUGGLETUM($\langle$RUBBERDUCK$\rangle$)

Which of the following statements is true for all inputs $\langle M, w\rangle$?

~~Yes~~	No	If M rejects w, then RUBBERDUCK rejects MAGIC.
~~Yes~~	No	If M accepts w, then DETECTOMUGGLETUM accepts $\langle$RUBBERDUCK$\rangle$.
Yes	~~No~~	If M rejects w, then DECIDEHALT rejects $\langle M, w\rangle$.
~~Yes~~	No	DECIDEHALT decides the language HALT. (That is, Professor Potter's reduction is actually correct.)
Yes	~~No~~	DECIDEHALT actually runs (or simulates) RUBBERDUCK.

(d) Which of the following languages are ***decidable***?

RACECAR

- ~~Yes~~ No — Binary representations of all perfect squares
- ~~Yes~~ No — $\{xy \in \{\mathtt{0},\mathtt{1}\}^* \mid yx \text{ is a palindrome}\}$

CAR|RACE

- Yes ~~No~~ — $\{\langle M\rangle \mid M \text{ accepts the binary representation of every perfect square}\}$
- Yes ~~No~~ — $\{\langle M\rangle \mid M \text{ accepts a finite number of non-palindromes}\}$
- ~~Yes~~ No — The set of all regular expressions that represent the language $\{\mathtt{0},\mathtt{1}\}^*$. (This is a language over the alphabet $\{\mathtt{Ø},\mathtt{\varepsilon},\mathtt{0},\mathtt{1},\mathtt{*},\mathtt{+},\mathtt{(},\mathtt{)}\}$.)

(e) Which of the following languages can be proved undecidable ***using Rice's Theorem***?

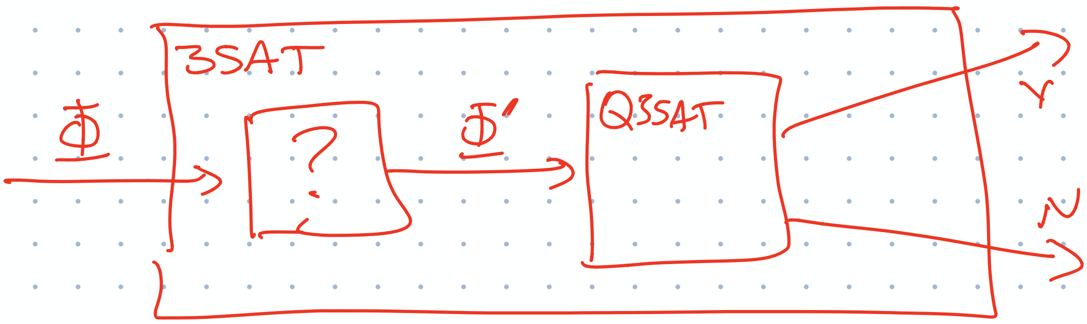
- ~~Yes~~ No — $\{\langle M\rangle \mid M \text{ accepts a finite number of strings}\}$
- Yes ~~No~~ — $\{\langle M\rangle \mid M \text{ accepts both } \langle M\rangle \text{ and } \langle M\rangle^R\}$
- ~~Yes~~ No — $\{\langle M\rangle \mid M \text{ accepts exactly 374 palindromes}\}$
- Yes ~~No~~ — $\{\langle M\rangle \mid M \text{ accepts some string } w \text{ after at most } |w|^2 \text{ steps}\}$

$\{\langle M\rangle \mid M \text{ accepts } \_\_\_\_\_\_ \}$

2. A ***quasi-satisfying assignment*** for a 3CNF boolean formula Φ is an assignment of truth values to the variables such that *at most one* clause in Φ does not contain a true literal. ***Prove*** that it is NP-hard to determine whether a given 3CNF boolean formula has a quasi-satisfying assignment.

Quasi3SAT

$(a \vee b \vee c) \wedge (\bar{a} \vee \bar{c} \vee d) \wedge (\bar{b} \vee c \vee \bar{d}) \wedge \cdots$



Given Φ build $\Phi' = \Phi \wedge (a \vee b \vee c)$

$\wedge (\bar{a} \vee b \vee c)$

$\wedge (a \vee \bar{b} \vee c)$

$\wedge (\bar{a} \vee \bar{b} \vee c)$

$\wedge \cdots$

$\wedge (\bar{a} \vee \bar{b} \vee \bar{c})$

a new variable

$\Rightarrow$ If Φ has sat assignment

add $a = b = c =$ True

$\rightarrow$ quasi sat assignment for Φ'

because all clauses in Φ ok

exactly one new clause bad.

$\Leftarrow$ If Φ' has quasi-sat assignment

exactly one new clause bad

So all old clauses good $\rightarrow$ Φ satisfied

$\square$

poly time ✓

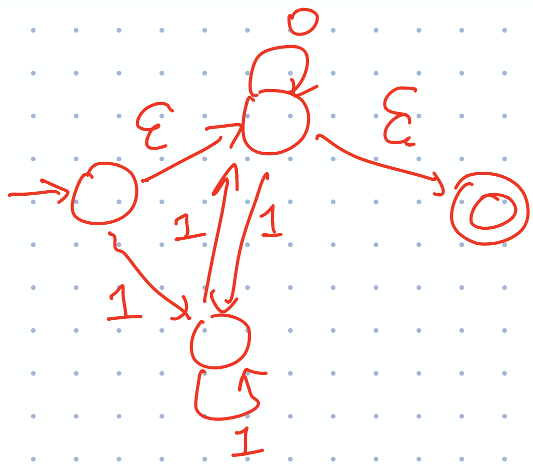
CS/ECE 374 A ✧ Spring 2018 Final Exam ♫ Problem 5	Gradescope name:

(a) Fix the alphabet $\Sigma = \{0, 1\}$. Describe and analyze an efficient algorithm for the following problem: Given an NFA M over Σ, does M accept at least one string? Equivalently, is $L(M) \neq \varnothing$?

(b) Recall from Homework 10 that deciding whether a given NFA accepts *every* string is NP-hard. Also recall that the complement of every regular language is regular; thus, for any NFA M, there is another NFA M' such that $L(M') = \Sigma^* \setminus L(M)$. So why doesn't your algorithm from part (a) imply that P=NP?

(a) Given M as a dir. graph

Is any accept state reachable in M from start state?



WFS

$O(V+E) = O(|Q|^2)$

(b) Changing NFA M into NFA M' where $L(M') = \Sigma^* \setminus L(M)$ takes exp. time.

(as far as we know)

CS/ECE 374 A ✧ Spring 2018	Gradescope name:
Final Exam ♫ Problem 4	

Suppose we want to split an array $A[1..n]$ of integers into k contiguous intervals that partition the sum of the values as evenly as possible. Specifically, define the *cost* of such a partition as the maximum, over all k intervals, of the sum of the values in that interval; our goal is to minimize this cost. Describe and analyze an algorithm to compute the minimum cost of a partition of A into k intervals, given the array A and the integer k as input.

For example, given the array $A = [1,6,-1,8,0,3,3,9,8,8,7,4,9,8,9,4,8,4,8,2]$ and the integer $k = 3$ as input, your algorithm should return the integer 37, which is the cost of the following partition:

$$[\overbrace{1,6,-1,8,0,3,3,9,8}^{37} \mid \overbrace{8,7,4,9,8}^{36} \mid \overbrace{9,4,8,4,8,2}^{35}]$$

The numbers above each interval show the sum of the values in that interval.

Where is the next wall?

MinCost(i,k) = minimum cost of a partition of $A[i..n]$ into k intervals.

We need MinCost$(1,k)$

$$\text{MinCost}(i,k) = \begin{cases} \sum_{j=i}^{n} A[j] & \text{if } k=1 \\ 0 & \text{if } i>n \\ \min\limits_{i \le j \le n+1} \left\{ \max \left\{ \sum_{l=i}^{j-1} A[l],\ \text{MinCost}(j,k-1) \right\} \right\} & \text{o/w} \end{cases}$$

n k

wall between $A[j-1] \mid A[j]$

Memo:

$\boxed{O(n^{\overset{2}{\cancel{3}}}k) \text{ time}}$