CSCI 341 Workshop 8

NP

November 27, 2025

Problem 1 (Word Acceptance Problem for Finite Automata)**.** Consider the following decision problem: recall that a finite automaton is a quadruple $\mathcal{A} = (Q, A, \delta, F)$ with a set Q of states, a transition relation $\delta \subseteq Q \times A \times Q$, and a set of accepting states $F \subseteq Q$. The *word acceptance problem for finite automata* is the decision problem

$$WAcc = \{(\mathcal{A}, x, w) \mid \mathcal{A} \text{ is a finite automaton, } x \in Q \text{, and } w \in \mathcal{L}(\mathcal{A}, x)\}$$

We are going to show that this language is in NP.

(1) Start by considering the related problem

$$DWacc = \{(\mathcal{A}, x, w) \mid \mathcal{A} \text{ is a deterministic finite automaton, } x \in Q \text{, and } w \in \mathcal{L}(\mathcal{A}, x)\}$$

  (a) Assume you have a class (like in Python or Java or C++) that stores an automaton and allows you to observe the set $\delta(x, a)$ for any $x \in Q$ and $a \in A$. Write some pseudocode that decides whether $(\mathcal{A}, x, w) \in DWacc$ for any instance $(\mathcal{A}, x, w)$ of the problem.

  (b) Come up with a faithful string representation of the deterministic word acceptance problem.
  Hint: how would you describe a finite automaton with a basic text editor?

  (c) Show that (the string representation of) $DWacc$ is in P by adapting your pseudocode from part (a) to run on a Turing machine (do you need multiple tapes?).

(2) Adapt your work from (1) to describe a verifier that solves $Wacc$ in nondeterministic polynomial time.

(3) Doesn't this contradict Rice's theorem?

Problem **2.** [Graph Reachability Problem] A *directed graph* is a pair $\mathcal{G} = (X, \to)$ consisting of a set X of *nodes* and a relation $\to \subseteq X \times X$ of *edges*. Given nodes $x, y \in X$, y is *reachable from* x if there are $x_1, \ldots, x_n \in X$ such that

$$x \to x_1 \to x_2 \to \cdots \to x_n \to y$$

Define

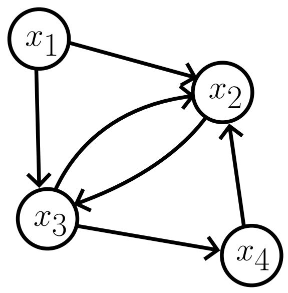
$$\mathbf{DGr} = \{(\mathcal{G}, x, y) \mid \mathcal{G} = (X, \to) \text{ is a directed graph and } x, y \in X\}$$

Then the *reachability problem* is

$$Rch = \{(\mathcal{G}, x, y) \in \mathbf{DGrph} \mid y \text{ is reachable from } x \text{ in } \mathcal{G}\} \subseteq \mathbf{DGr}$$

(1) Write down the sets X and $\to$ of nodes and edges in the directed graph below.

$\mathcal{G} = (X, \to)$



(a) Is $(\mathcal{G}, x_1, x_2)$ in Rch?

(b) Is $(\mathcal{G}, x_2, x_1)$ in Rch?

(c) Is $(\mathcal{G}, x_1, x_4)$ in Rch?

(d) Is $(\mathcal{G}, x_4, x_1)$ in Rch?

(2) Come up with a faithful string representation of the reachability problem, and write down the string representation of $(\mathcal{G}, x_1, x_4)$ above.
Hint: try describing the directed graph above with a basic text editor.

(3) Now we'll show that the reachability problem is in NP. Remember that the goal is to describe a general Turing program that either verifies or disproves that a particular instance of the problem lies in (the string representation of) Rch in polynomial time.

a Write some pseudocode that describes a nondeterministic algorithm for solving the problem. What is a "guess" in this scenario? Write down the definition of "verification in polynomial time" and check that your pseudocode does, in fact, run in polynomial time.

b Now let's think about how to do this with a Turing machine. Describe a few different ways that you might set up the tape to store all of the necessary information about a string representation of $(\mathcal{G}, x, y)$ and the information you need to keep track of as the program runs. Decide which one you like best as a team.

c Use your pseudocode in (a) to describe a Turing machine that implements your algorithm. Check that it runs in nondeterministic polynomial time.

Problem 3 (Strongly Connected Problem)**.** A directed graph $\mathcal{G} = (X, \to)$ is *strongly connected* if for any $x, y \in X$, x is reachable from y and y is reachable from x. Show that the problem

$$SCon = \{\mathcal{G} \mid \mathcal{G} \text{ is a strongly connected directed graph}\}$$

is in NP by cooking up a reduction $r\colon SCon \preceq Rch$ that runs in $\mathcal{O}(n^2)$-time.

If you don't quite get to this problem, don't worry, we will go over it briefly in class on Friday.

Problem 4 (SAT)**.** The set $Form$ of all *(propositional) formulas* is generated by the grammar

$$F \to p \mid (F \wedge F) \mid (F \vee F) \mid (\neg F)$$

where $p \in \mathbb{P}$, $\mathbb{P}$ is the set of *basic propositional formulas*. An *assignment* is a function $\alpha\colon \mathbb{P} \to \{0,1\}$, indicating whether each basic proposition is taken to be "false" or "true". The *truth value of a formula* is computed from an assignment α recursively with the truth table below:

φ_1	φ_2	$\varphi_1 \wedge \varphi_2$	$\varphi_1 \vee \varphi_2$	$\neg\varphi_1$
1	1	1	1	0
0	1	0	1	1
1	0	0	1	0
0	0	0	0	1

- Consider the following decision problem:

  $$FTrue = \{(\varphi, \alpha) \mid \varphi \in Form, \alpha\colon \{p,q,r\} \to \{0,1\}\}$$

  Show that (a string representation of) $FTrue$ is in P.

- A formula $\varphi \in Form$ is *satisfiable* if there exists an assignment $\alpha\colon \mathbb{P} \to \{0,1\}$ that makes $\varphi = 1$ (i.e., true). We obtain the decision problem

  $$SAT = \{\varphi \in Form \mid \varphi \text{ is satisfiable}\}$$

  Use your polynomial time algorithm in the previous part to show that (a string representation of) SAT is in NP.