CS 39: Theory of Computation **Winter '23**

FINAL EXAM — 03/12/2023

Prof. Chakrabarti ***Student: Amittai Siavava***

Credit Statement

I worked on this final exam alone, with reference to class notes and the following books:

(a) **Introduction to the Theory of Computation** by **Michael Sipser**.

Problem 1.

For a string $w \in \{0,1\}^*$, let $\beta(w)$ denote the integer obtained by interpreting w as a binary number. For example, $\beta(0011) = 3$. We also define $\beta(\varepsilon) = 0$. Define the following language over the alphabet $\{0, 1, \#\}$:

$$L_1 := \left\{x\#y : x, y \in \{0,1\}^* \text{ and } \beta(x) < \beta(y)\right\}.$$

Determine, with proof, whether or not L_1 is context-free.

L_1 is not context-free.

Suppose it is, then by the pumping lemma for context-free grammars, there exists some minimum pumping length p such that for all strings $w \in L_1$ with $|w| \geq p$, there exists a decomposition $w = uvxyz$ such that;

(i) $|vxy| \leq p$

(ii) $|vy| > 0$

(iii) $uv^kwy^kz \in L_1$ for all $k \geq 0$

Pick $s = 10^p\#0^p110^p$. Clearly, $s \in L_1$ since $\beta(10^p) < \beta(0^p110^p)$. Write $s = uvwyz$. There are three possibilities for wyz:

(i) If vwy is to the left of the $\#$ symbol, then pumping up the string disproportionately increases the value of the string to the left, and breaks the condition that.

(ii) If vwy is to the right of the $\#$ symbol, then pumping down the string decreases the value of the string to the right, and breaks the condition.

(iii) If vwy is at the boundary (i.e. vwy contains the $\#$ symbol), then there are two possibilities:

(i) If # occurs in w, then if $|v| > 0$, pumping up the string increases the value of the string to the left, and breaks the condition.

(ii) If # does not occur in w, then pumping down the string removes the # symbol, hence the result cannot be in L_1.

Problem 2.

Define the *reversal* of a language L to be the language $L^R := \left\{x^R : x \in L\right\}$.

(a) Let $M = (() Q, \Sigma, \delta, q_0, F)$ be a DFA. Give a formal description of an NFA that recognizes $\mathcal{L}(M)^R$. Give brief proofs that your construction is complete and sound.

$N = (Q \cup \{s\}, \Sigma, \delta_2, s, F_2)$ where $s \notin Q$, and:

$$\delta_2(s, \varepsilon) = F$$

$$\delta_2(q, a) = \left\{q' : \delta(q', a) = q'\right\}$$

$$F_2 = \{q_0\}$$

Completeness:

Let $w \in \mathcal{L}(M)$ be a string recognized by M. Let $n = |x|$, then there exists a computational path $x_1, x_2, \ldots x_n$ of M on w such that $x_1 = q_0$, $\delta(x_j, w_j) = x_{j+1}$ for all $j < n$, and $x_n \in F$. By the construction of N from M, there exists an ε-transition from s to x_n and there exists a computation path of N on w that takes N from x_n to x_1. Consequently, there exists a computation path of N on w^R that takes N from s to q_0, which is an accepting state of N, so N accepts w^R.

Soundness:

Suppose $w = w_1 w_2 \ldots w_n$ is a string accepted by N. Then there exists some computation path $x_1, x_2, \ldots x_k$ of N on w such that $x_1 = s$, $x_2 \in F$ (where F is the set of accepting states in the original DFA), $\delta_2(x_j, w_j) = x_{j+1}$ for all $j < k$, and $x_k \in F_2$. Ignoring the epsilon transition from s to x_2, we have that $\delta(x_{j+1}, w_j) = x_j$ for all $j < k$, where δ is the original transition function of M. Since $x_k = q_0$, this means there is some computation path of M on w that starts at q_0 and ends at x_2, which happens to be an accepting state in M. Therefore, M accepts w^R.

(b) Let $k \geq 3$ be an integer. Prove that there exists a k-state DFA M_k over the alphabet $\{0, 1\}$ such that every DFA that recognizes $\mathcal{L}(M_k)^R$ requires at least 2^{k-2} states.

Hint: Look to the languages we considered very early in the course for inspiration: you're looking for a language that admits a small DFA whose reversal requires an exponentially larger DFA. For the proof of the 2^{k-2} lower bound, use the techniques and results you learnt in Homework 3. If you can't quite prove the lower bound as stated, you can still earn almost full credit for proving a $2^{\Omega(k)}$ bound.

Let $L = (0 \cup 1)^{k-2}1(0 \cup 1)^*$. Then L is the language of strings in which the $k-2$th character is a 1. This is easily recognizable by a DFA with only k states:

$$M = (Q, \{0,1\}, \delta, q_0, F) \text{ where}$$

$$Q = \{q_i : 0 \leq i < k-2\} \cup \{q_{accept}, q_{reject}\},$$

$$F = \{q_{accept}\},$$

$$\delta(q,x) = \begin{cases} q_{accept} & \text{if } x = 1 \text{ and } q = q_{k-3} \\ q_{accept} & \text{if } q = q_{accept} \text{ and } x \in \{0,1\} \\ q_{reject} & \text{if } x = 0 \text{ and } q = q_{k-3} \\ q_{reject} & \text{if } q = q_{reject} \\ q_{i+1} & \text{if } x \in \{0,1\} \text{ and } q = q_i \text{ and } i < k-2 \\ q_{reject} & \text{otherwise (in case of invalid strings!)} \end{cases}$$

The idea is to use states to count the number of characters read in from the input. We make $k-2$ transitions from state q_0 to state q_{k-3}. We then check whether the letter $k-1$ is a 0 or a 1 and accept or reject accordingly. Both q_{accept} and q_{reject} are trap states; the remainder of the string is only checked to make sure it is a valid string in the language.

Now, consider the reversal of M. That is, the DFA recognizing the language

$$\mathcal{L}(M)^R = (0 \cup 1)^* 1 (0 \cup 1)^{k-2}.$$

Let M^R be a DFA recognizing $\mathcal{L}(M)^R$. Since the length of the input string is not known ahead of time, M^R must track the previous $k-2$ characters read in from the input at any given point. For instance, if $k-1 = 4$ (i.e. we want the 4-th last character to be a 1), then each of $\{000, 001, 010, 011, 100, 101, 110, 111\}$ is in a unique equivalence class and must be tracked separately since reading some sequence in each case, reading some sequence of characters results in a unique sequence of transitions.

In total, the last $k-2$ characters characters must be tracked in order to determine the letter at position $k-1$ from the end after any sequence of transitions, and there are a 2 possibilities for each of the $k-2$ positions in the string that are being tracked, so M^R must have at least 2^{k-2} states.

Problem 3.

A language L over an alphabet Σ is *symmetric* if $L = L^R$. Define the language

$$\mathsf{SYM}_{\mathsf{CFG}} : \{\langle G \rangle : G \text{ is a CFG and } \mathcal{L}(G) \text{ is symmetric } \}.$$

Determine, with proof, whether $\mathsf{SYM}_{\mathsf{CFG}}$ is decidable and whether it is recognizable.

T_0 = "On input $\langle G \rangle$, where G is a CFG over Σ:

1. Construct a PDA P such that $\mathcal{L}(P) = \mathcal{L}(G)$.
2. Construct a TM T such that P recognizes the computation tableau of T.
3. Output $\langle T \rangle$."

$T \in E_{TM} \iff T$ does not accept any string

$\iff$ a rejecting state is reached before any accepting state is reached $\iff$ every string in $\mathcal{L}(G)$ is symmetric

$\iff C_0 \# C_1 \# C_2 \# \ldots \# C_n$ is symmetric

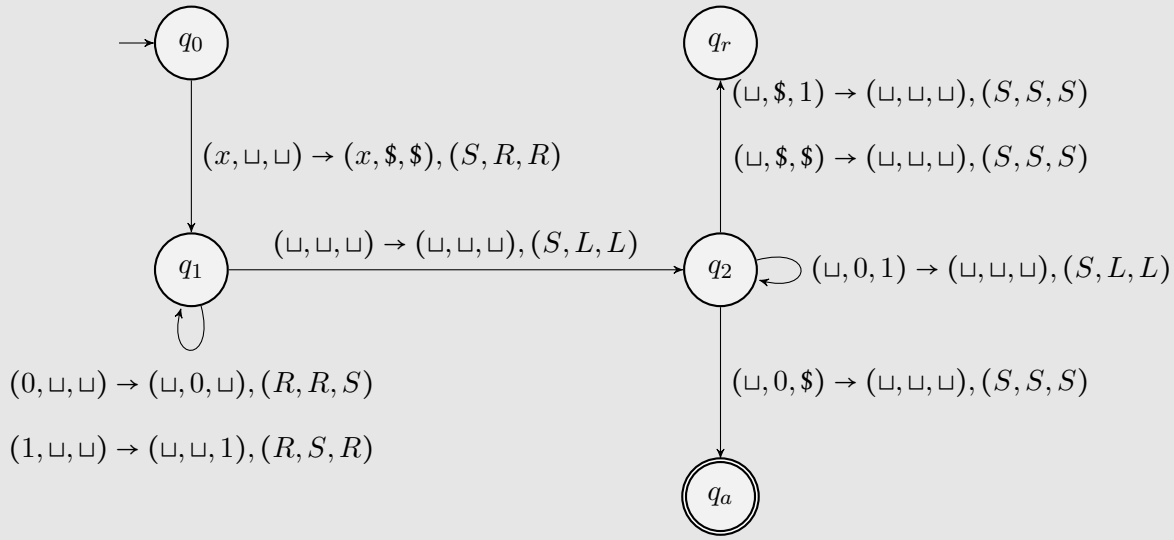
$\iff G \in \mathsf{SYM}_{\mathsf{CFG}}$

Problem 4.

Let $L_4 = \{\langle D\rangle : D \text{ is a DFA over } \{0,1\}^* \text{ and } \exists x \in \mathcal{L}(D) \text{ with } N_0(x) > N_1(x)\}$. Recall that $N_a(x)$ denotes the number of occurrences of a in x. Determine, with proof, whether or not L_4 is decidable and whether it is recognizable.

L_4 is recognizable but not decidable.

Consider the following 3-tape deterministic turing machine T_1 which scans an input string x and copies all the 0's to tape 1 and all the 1's to tape 2 then scans the number of 0's and 1's, accepting if $N_0(x) > N_1(x)$ and rejecting otherwise:



Since the set of all strings, Σ^*, is decidable, let E be a lexicographic enumerator for Σ^*. We can recognize L_4 as follows:

T = "On input $\langle D\rangle$, where D is a DFA;

1. Simulate E to start enumerating strings in Σ^*.

2. For each step $i = 1, 2, 3, \ldots$:

2.1. Let σ_i be the i'th string listed by E.

2.2. Simulate T_1 on input $\langle \sigma_i \rangle$.

2.3. If T_1 rejects, proceed to step $i + 1$.

2.4. If T_1 accepts $\langle w_i \rangle$, simulate D on w_i, If D accepts, Accept . If D rejects, proceed to step $i + 1$."

If the input DFA is in L_4, then the TM eventually halts and accepts. However, if the input DFA is not in L_4, the TM will never halt and will keep testing strings enumerated from Σ^*. Therefore, L_4 is not decidable but it is recognizable.

Problem 5.

We showed in class that the language EQ_{DFA} is decidable. In this problem, for concreteness' sake, assume that EQ_{DFA} only concerns DFAs over the alphabet $\{0, 1\}$ and that DFAs are encoded in such a way that $\text{EQ}_{\text{DFA}} \subseteq \{0, 1\}^*$. Alice asserts that $\text{EQ}_{\text{DFA}} \in \mathsf{P}$, while Bob asserts that EQ_{DFA} is NP-complete. Which of them is right? Prove your answer.

Alice is right that $\text{EQ}_{\text{DFA}} \in \mathsf{P}$.

Note that

$$\text{E}_{\text{DFA}} := \{\langle M \rangle : M \text{ is a DFA and } \mathcal{L}(M) = \varnothing\}$$

can be decided as follows:

T_1 = "On input $\langle M \rangle$, where $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA:

1. Simulate a graph search from q_0, taking every possible transition and ignoring loops until every possible path dies.
2. If any state $q_f \in F$ is reachable, then there exists some accepting computational path for some string in the DFA's language, so $\langle M \rangle \notin \text{E}_{\text{DFA}}$. Reject .
3. If no state $q_f \in F$ is reachable, then is no accepting path of M on any string in its alphabet, hence $\mathcal{L}(M) = \varnothing$. Accept ."

Note that since T_1 does not revisit states it runs in $O(n + m)$ where $n = |Q|$ and m is the number of transitions in M.

We can decide EQ_{DFA} as follows:

T_2 = "On input $\langle M_1, M_2 \rangle$, where M_1 and M_2 are DFAs:

(a) Construct a new DFA M such that $\mathcal{L}(M) = (\mathcal{L}(M_1) \setminus \mathcal{L}(M_2)) \cup (\mathcal{L}(M_2) \setminus \mathcal{L}(M_1))$.

(b) Simulate the decider for E_{DFA}, T_1, on $\langle M \rangle$.

(c) If T_1 accepts $\langle M \rangle$, then $\mathcal{L}(M_1) = \mathcal{L}(M_2)$. Accept .

(d) If T_1 rejects $\langle M \rangle$, then $\mathcal{L}(M_1) \neq \mathcal{L}(M_2)$. Reject ."

Claim 5.1. *T_2 runs in polynomial time.*

Proof. First, we show that the construction of M runs in polynomial time. Given $M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$, we construct M as follows:

(i) Define $M = \{(q_i, q_j) : q_i \in Q_1, q_j \in Q_2\}$.

(ii) Define $q_0 = (q_{01}, q_{02})$.

(iii) Define $\delta((q_i, q_j), a) = (\delta_1(q_i, a), \delta_2(q_j, a))$.

(iv) Define $F = \{(q_i, q_j) : q_i \in F_1 \text{ and } q_j \notin F_2 \text{ or } q_i \notin F_1 \text{ and } q_j \in F_2\}$.

If M_1 has n_1 states and m_1 transitions, and M_2 has n_2 stats and m_2 transitions, then M has $n_1 \cdot n_2$ states and $m_1 \cdot m_2$ transitions. Thus, this operation runs in polynomial time (since its runtime is just the sum of all the states and transitions being created).

T_2 then simulates T_1 on M. Since T_1 just performs a breadth-first graph search, it runs in linear time in the sum of the number of states and the number of transitions in M, which we showed above to be polynomial. Since, T_1 runs in polynomial time, $\text{EQ}_{\text{DFA}} \in \mathsf{P}$. □

Problem 6.

You are in charge of analyzing a large social media network modeled as an undirected graph $G = (V, E)$: each vertex is a user and each edge represents a mutual "friendship" relation between two users. A *diverse sample* is a set of users such that no two o them are friends and, furthermore, no two of them have a friend in common. Formally, a set $S \subseteq V$ is a diverse sample if

$$\forall x, y \in S : \{x, y\} \notin E \wedge \nexists w \in V(\{w, x\} \in E \wedge \{w, y\} \in E).$$

As part of your analysis process, you have been asked to determine the largest diverse sample in the social network. After failing to find an efficient algorithm for this, you begin to suspect that this might be impossible. Prove, assuming that $\mathsf{P} \neq \mathsf{NP}$, there is indeed no polynomial-time algorithm for this optimization problem.

First, let us define the language

$$\textsc{Diverse} := \{\langle G, k\rangle : G \text{ is a graph and } G \text{ has a diverse sample of size } k\}.$$

If Diverse were decidable, then the problem of finding the largest possible diverse sample could be solved in polynomial time as follows:

T_0 = "On input $\langle G\rangle$, where G is a graph:

1. $S \leftarrow \varnothing$.

2. for $i \leftarrow 1 \ldots |V|$

2.1. if $\langle G, k\rangle \in$ Diverse:

2.2. $S \leftarrow$ diverse sample of size i.

3. Return S.

Thus, we show that Diverse is NP-complete.

Diverse can be solved in $\mathsf{NP}$by guessing a subset $S \subset 2^V$ and checking if the subset is diverse.

We proved in class that the set

$$\textsc{IndSet} := \{\langle G, k\rangle : G \text{ has an independent set of size } k\}$$

is NP-complete. We can reduce IndSet to Diverse as follows:

T_1 = "On input $\langle G, k \rangle$, where G is a graph and k is an integer:

1. Define the set $E' = E \setminus \{\{u, w\}, \{v, w\} : u, v, w \in V, \{u, v\} \notin E, \{\{u, w\}, \{v, w\}\} \subseteq E\}$.

2. Define the graph $G' = (V, E')$.

3. Output $\langle G', k \rangle$."

$$\begin{aligned}
\langle G, k \rangle \in \textsc{IndSet} &\iff \exists S \subseteq V, |S| = k \text{ and } \{u, v\} \notin E \text{ for all } u, v \in S \\
&\iff \{u, v\} \notin E' \text{ and } \nexists w \in V(\{w, x\} \in E' \wedge \{w, y\} \in E') \\
&\iff \langle G', k \rangle \in \textsc{Diverse}
\end{aligned}$$

Since IndSet is NP-complete, Diverse must also be NP-complete. However, if Diverse $\in$ NP and P $\neq$ NP, then Diverse $\notin$ P and there cannot exist a polynomial time algorithm for finding the largest diverse sample.

Problem 7.

A *balanced assignment* to the boolean variable $x_1 \dots x_{2n}$ is one where exactly n of the variables are assigned True and the rest are assigned False . Define the language

$$\text{BAL-SAT} := \{\langle\varphi\rangle : \varphi(x_1 \dots x_{2n}) \text{ is a cnf formula and } \exists \text{ a balanced assignment that satisfies } \varphi\} .$$

Prove that BAL-SAT is NP-complete.

That BAL-SAT can be solved in NP: guess an assignment and check if it satisfies the formula and is balanced.

Reduction of SAT to BAL-SAT:

T = "On input $\langle\varphi\rangle$, where $\varphi(x_1, x_2, \dots, x_n)$ is a cnf formula:

1. Initialize a new formula ψ with all the logical expressions in φ.

2. For $i = 1, 2, \dots, n$:

2.1. Define a new variable x_{i+n}.

2.2. Add the clause $(x_{n+i} \vee \neg x_{n+i})$ to ψ.

3. Return $\langle\psi\rangle$."

$\langle\varphi\rangle \in$ SAT $\iff$ $\exists$ an assignment to $x_1, x_2, \dots, x_n$ that satisfies φ

$\iff$ $\exists$ an assignment to φ with $k \leq n$ variables assigned True , that satisfies φ

$\iff$ $\exists$ an assignment to ψ with k variables from φ assigned True that satisfies ψ

$\iff$ $\exists$an assignment to ψ with n original variables from φ assigned True and $n - k$ new variables assigned True .

$\iff$ $\langle\psi\rangle \in$ BAL-SAT

Problem 8.

What is the most interesting thing that you learned in this course? hat is one skill or technique you feel proud to have mastered as a result of taking this course? Please answer each question in a single sentence.

The most interesting thing I learned in CS-39 was the concept of undecidability and P vs. NP. I feel proud that I can reason about computation in a more abstract and formal way and, for instance, see/prove that some problems are NP-complete and cannot be solved in polynomial time.