Answer Set 4

Kazi Samin Mubasshir
Email: kmubassh@purdue.edu
PUID: 34674350

Problem 1

Contributors: Farhanaz Farheen, Tania Chakraborty, Devin Atilla Ersoy
Proof. Here is a Turing machine that decides the language:

1. If the head sees symbol b at the beginning, move the head right by one position.
2. Move the head to the right as long as it sees the symbol a
3. If the head sees b followed by the blank symbol, *accept*.

Formal description:

$$M = (Q, \sum, \Gamma, \delta, q_1, q_{accept})$$

$$Q = \{q_1, q_2, q_3, q_{accept}\}$$

$$\sum = \{a, b\}$$

$$\Gamma = \{a, b, \sqcup\}$$

The state diagram of the Turing Machine is given in Figure 1

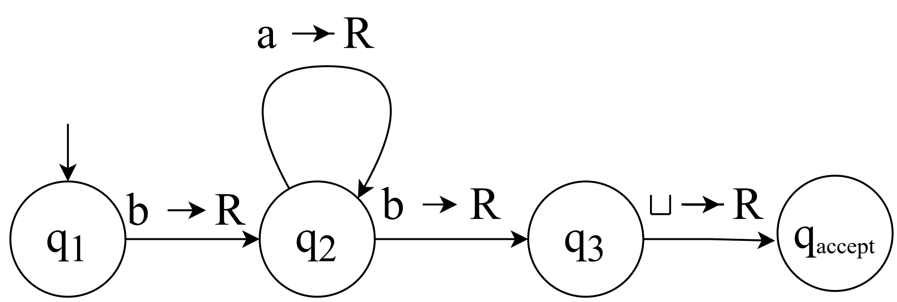

Figure 1: State Diagram of *Turing Machine* that decides language A

Complexity: The machine runs in $O(n)$ as it moves the head from left to right from the beginning of the input to the end once and decides the language.

Problem 2

Contributors: Farhanaz Farheen, Tania Chakraborty, Devin Atilla Ersoy

Answer to 2a:
Let K and L be languages that are decidable. Language $KL = \{xy|x \in K\ and\ y \in L\}$ is produced by the concatenation of languages K and L. Since K and L can be decided, it follows that there are Turing machines TM_K and TM_L that can decide K and L, respectively.

We can build a Turing machine TM_{KL} that decides KL in order to demonstrate that it is decidable. The following procedure can be used to build the machine: Take into account the input string w. We must determine whether w is of the type xy for $x \in K$ and $y \in L$. If this is the case, then there must be a point at which we may divide w into x and y. Since there are only a finite number of possible ways to divide the string, we may test every possibility, accept it if it exists, and reject it if it doesn't.

i Divide input w into strings xy by attempting every feasible split

ii Input x to TM_K and y to TM_L

iii If TM_K and TM_L concur, accept; otherwise, reject.

If a computation path is accepted, the split has been successful, and the string is in KL. If every possible computing step fails, the string is not in KL. It is obvious that the machine TM_{KL} halts in either scenario.

Thus, KL is decidable and this shows that the collection of $Turing-recognizable$ languages is closed under the operation of concatenation.

Answer to 2b: Let L_1 and L_2 be languages in NP and $V_i(x,c)$ be an algorithm for $i = 1, 2, \cdots$ that determines whether a string x and a potential certificate c are indeed certificates for $x \in L_i$.

Thus,

- $V_i(x,c) = 1$ if certificate c verifies $x \in L_i$, and
- $V_i(x,c) = 0$ otherwise

Due to the fact that L_1 and L_2 are both in NP, we know that $V_i(x,c)$ terminates in polynomial time $O(|x|^d)$ for some constant d.

We will build a polynomial-time verifier V_3 for L_3 to demonstrate that $L_3 = L_1 \cup L_2$ is also in NP. We can simply design a verifier $V_3(x,c) = V_1(x,c) \vee V_2(x,c)$ because a certificate c for L_3 will have the property that either $V_1(x,c) = 1$ or $V_2(x,c) = 1$. Evidently, $x \in L_3$ only applies if and only if c is a certificate that makes $V_3(x,c) = 1$ possible. Also, the new verifier, V_3, will operate over the polynomial time $O(2(|x|^d))$.

The union L_3 of the two languages in NP is therefore also in NP, making NP closed under union.

Problem 3a

The following arguments can be used to prove that the $D-SAT$ problem is $NP-Complete$.

D-SAT is in NP: If any problem is in NP, then given an instance of the problem and a certificate, it can verify the certificate in polynomial time. This can be done by giving a set of satisfying assignments for the variables in ϕ, and verify if each clause is satisfied in ϕ.

D-SAT is NP-Hard: We then reduce a known $NP-Complete$ problem, in this case $3-SAT$, to our problem in order to prove that $D-SAT$ is $NP-Complete$. Given a $3-CNF$ formula ϕ we create a boolean formula ϕ' by adding a pair of literals $(x \vee x')$ to each clause of ϕ, where x is an additional variable. This reduction works in polynomial time. The two claims that follow are true at this point:

- If ϕ is unsatisfiable, then some clause of ϕ must be $FALSE$, therefore, ϕ' must also be unsatisfiable.
- If $3-SAT$ formula ϕ' is satisfiable, then using the same set of assignment variables in ϕ', we can have both $x=0$ and $x=1$ as the valid assignments to ϕ.

Therefore, $D-SAT$ Problem is $NP-Complete$.

Problem 3b

Contributors: Farhanaz Farheen
Solution:

1. Each clause in a $\neq$ assignment has at least one literal that is allocated 1 and at least one literal that is assigned 0, respectively. This attribute is preserved in the negation of a $\neq$ assignment, making it a $\neq$ assignment as well.

2. We obtain a polynomial time reduction by replacing each clause c_i,

$$(y_1 \vee y_2 \vee y_3)$$

   with the two clauses

$$(y_1 \vee y_2 \vee z_i) \ and \ (\neg z_i \vee y_3 \vee b)$$

   where z_i is a new variable for each clause c_i and b is a single additional new variable. To demonstrate that the reduction provided is effective, we must demonstrate that if the formula ϕ is mapped to ϕ', then ϕ is satisfiable (in the usual sense) iff ϕ' has a $\neq$ assignment.

   - First, if ϕ is satisfiable, then we may extend the assignment to ϕ' and assign z_i to 1 if both literals y_1 and y_2 in clause c_i of ϕ are assigned 0. This will give us a $\neq$ assignment for ϕ'. If not, z_i is set to 0. We then assign b to 0.
   - Second, if ϕ' has an $\neq$assignment we can find a satisfying assignment to ϕ as follows.
     - We can suppose that b is assigned to 0 via the $\neq$ assignment (otherwise, negate the assignment).

- The entire y_1, y_2, and y_3 can not be set to 0 in this assignment since doing so would compel one of the ϕ' clauses to contain just 0s.

As a result, limiting this assignment to the variables of ϕ results in a satisfying assignment.

3. It is obvious that $\neq SAT$ is in NP. As a result, it is NP-complete because $3SAT$ reduces to it.

Problem 4

Contributors: Farhanaz Farheen
Proof: We will prove the contrapositive here. We will show that if $P \neq NP$, then A is not $NP-complete$. By assumption there is a reduction g from SAT to A. The algorithm is as follows:

Algorithm:

1. We are given a formula f, we will keep a list of formulas $y_1, \cdots, y_k$ such that: f is satisfiable iff some of $y_1, ..., y_k$ is satisfiable. Initially the list contains f
2. We alternatingly repeat two kinds of steps:
   (a) Replace every y_i by two formulas: $y_i[true/x]$ and $y_i[false/x]$, obtained by substituting true/false for one of variables.(clearly y_i is satisfiable iff some of $y_i[true/x]$, $y_i[false/x]$ is satisfiable)
   (b) For every pair y_i, y_j such that $g(y_i) = g(y_j)$, remove y_i from the list, leave only y_j(notice that y_i is satisfiable iff some of y_j is satisfiable)

Runtime:

1. g is a polynomial-time reduction to A. Thus $|g(y_i)| < p(|y_i|)$ for some polynomial p. Since there is only one single-letter word of every length, there are only $p(|y_i|) \leq p(|f|)$ possibilities for $g(y_i)$
2. So the list has length $\leq p(|f|)$ after every execution of step 2, and $\leq 2 \cdot p(|f|)$ after every execution of step 1

So there exists a polynomial time algorithm and thus A is not $NP-complete$. This completes the proof.