Aughdon Breslin and Isabella Cruz 8 Dec 2020

"I pledge my honor that I have abided by the Stevens Honor System."

Problem 1

(10 points) Show that the language L = {<M, w, k> : TM M accepts input w and never moves its head beyond the first k tape cells} is decidable.

A:

Idea: M moving past the kth cell depends only on what the first k cells initially hold. So there are a finite number, $|\Gamma|^k$, of possible combinations for what those first k tape cells hold, which means a finite number of things to check.

Strat: For each combination $x \in |\Gamma^k|$ labeling the first k tape cells, start M with tape x and run M(w), keeping track of the number of moves, m. If M is within the first k tape cells, then there must be a finite number of distinct configurations of M; let n be that number.

If M makes more than n moves without passing the kth cell, then M must have repeated a configuration. Via the Pigeonhole Principle, if there are only n configurations and each move either moves to a new configuration or repeats a previous one, then if there have been more than n moves made, a configuration must have been repeated. This means M is in a loop of configurations.

For each combination x, M can: accept/reject while staying within the first k tape cells in m < n moves, it can make m = n moves while staying within the first k tape cells, thus showing from above that M is in a loop of configurations and thus will never move its head beyond the first k tape cells, or it can move past the first k tape cells in m < n moves.

If M follows the first case, then if M accepts we accept, since M accepts input w and never moves its head beyond the kth tape cell. If M rejects, we discard the current x and move onto the next one. If M follows the second case, we discard the current x and move onto the next one. If M follows the third case, we discard the current x and move onto the next one. If no x causes M to accept, then we reject.

Since every input is decided in a finite amount of time, language L, {<M, w, k> : TM M accepts input w and never moves its head beyond the first k tape cells}, is decidable.

Problem 2

(10 points) Recall that $A <_p B$ if there is a polynomial-time computable function f such that $w \in A \Leftrightarrow f(w) \in B$.

a. Show that the relation $<_p$ over languages is transitive.

A:

Let $A <_p B$ and $B <_p C$. This means there exist polynomial-time computable functions $f_{AB}, f_{BC}: \Sigma^* \to \Sigma^*$ such that $w \in A \Leftrightarrow f_{AB}(w) \in B$ and $w \in B \Leftrightarrow f_{BC}(w) \in C$.

Thus, by defining a polynomial-time computable function $f_{AC} = f_{AB}(f_{BC}(w))$, we show $w \in A \Leftrightarrow f_{AC}(w) \in C$ is valid. Therefore, $A <_p C$, and the relation $<_p$ over languages is transitive.

b. Show that if $\forall A, B \in P$, if $B \neq \Phi$ and $B \neq \Sigma^*$ then $A \leq_p B$.
(Hint: this is easier than it seems since $A \in P$. Now use the definition.)

Let word $c \in B$ and word $d \notin B$. Define function f as follows: If $w \in A$, then $f(w) = c$. If $w \notin A$, then $f(w) = d$. Since $A \in P$, f is computable in polynomial time. Since this is a polynomial-time reduction, $A <_p B$.

This does not work for $B = \Phi$ or $B = \Sigma^*$, since for $B = \Phi$, a contradiction arises when we have $c \in B$, since by definition, no element can belong to Φ, and for $B = \Sigma^*$, a contradiction arises when we have $d \notin B$, since by definition, every element must belong to Σ^*.

c. Show that if P = NP then every language, other than Φ and Σ^*, in P is NP-Complete. (You can use the result of part (b) even if you didn't prove that.)

A language B is NP-complete if $B \in NP$ and $\forall A \in NP$: $A <_p B$.

If P = NP, then $\forall B \in P$, $B \in NP$ and $\forall A \in NP$, $A \in P$.
Now, since we know all languages in NP are also in P and vice versa, we can use the proof in (b) to show that $\forall A, B \in P$, if $B \neq \Phi$ and $B \neq \Sigma^*$ then $A \leq_p B$.

Therefore, if P = NP, then every language other than Φ and Σ^* in P is NP-complete since both conditions, $B \in NP$ and $\forall A \in NP: A <_p B$, are satisfied.

Problem 3

(20 points) Behold, a genie appears before you! Given a formula $\Phi(x_1, x_2, \ldots x_n)$ in conjunctive normal form with n boolean variables, the genie will correctly tell you (in one step) whether or not the formula is satisfiable. Unfortunately, the genie will not give you a truth assignment to the variables that makes the formula true.

Your problem is to figure out a satisfying truth assignment when the genie says the formula is satisfiable. You can present the genie with a polynomial (in n) number of queries.

i. (5 points) Give a high-level description of your algorithm, with enough detail.

Assumption: the formula is satisfiable

M = On input X where X represents a formula in CNF and x_k represents its different variables
Ask the genie if the formula is satisfiable:
    If no; the formula is not satisfiable
    Else: continue
For every x_k where $1 \leq k \leq n$:
    Set all occurrences of x_k = TRUE in the formula
    Ask Genie if formula is satisfiable:
        If yes; x_k = TRUE
        If no; x_k = FALSE

Every line inside the for loop takes one step so our time complexity follows a polynomial number of queries.

ii. (2 points) What is the maximum number of queries made by your algorithm?

The maximum number of queries made by our algorithm is n (one query for each x_k).

iii. (3 points) Explain why your algorithm correctly finds a satisfying assignment for a satisfiable formula.

Our algorithm essentially guesses each boolean value and uses the genie to check if the guess is correct. If there exists a case such that the formula is satisfiable when x_k is true, the formula will remain satisfiable when x_k is set to the boolean value true. Conversely, if we set x_k to be true and the formula is no longer satisfiable, we know that x_k must be false. Then we can continue building the satisfying truth assignment until we've gone through all x_k.

iv. (10 points) A second genie appears! Given an undirected graph, this genie will correctly tell you whether or not the graph has a Hamiltonian cycle. How will you use this genie to find a Hamiltonian cycle in any graph that has one in polynomial time?

M2 = On input G where G is an undirected graph
Ask the genie if G has a hamiltonian cycle:
    If no; there's no cycle
    Else: continue
For every vertex in G:
    For every edge that connects to that vertex:
        Remove the edge
        Ask the Genie if the graph has a hamiltonian cycle:
            If yes; continue (do not add the edge back)
            If no; add the edge back into the graph and continue

The remaining graph is the path of the hamiltonian cycle in graph G. We removed all edges not part of the cycle when we removed each edge at a time and asked the genie if there was still a hamiltonian cycle in the graph. If there was no longer a hamiltonian cycle, that edge must have been included in said cycle, so we add it back. Else it is not part of the cycle. Thus the only remaining edges make up the path of the hamiltonian cycle in G.

This algorithm uses nested for loops and increments based on the number of vertices and the degree of each vertex. A vertex in a graph with a hamiltonian

cycle can have a degree of at most V where V = the number of vertices in the graph (V-1 edges that could connect from other vertices and one possible reflexive edge). Thus the time complexity of this algorithm is O(V*V) or $O(V^2)$.

Problem 4
(10 points)
(a) Give a high-level description of a linear time algorithm to determine if a directed graph contains a directed cycle.

TM N: On input G (where G is a directed graph)

1. Run a Depth-First-Search on graph G
   Mark each vertex you visit as you traverse through the graph
   If you hit a dead end (a vertex with an out-degree of 0);
   remove the current vertex, and continue the DFS
   If the pointer reaches a marked edge;
   ACCEPT
2. If the DFS completes and we've not accepted; REJECT

This algorithm works because it marks all vertices that are potentially part of a directed cycle and removes vertices when we backtrack in our DFS. When we backtrack, we know that the path of vertices was definitely not part of a directed cycle by nature of a DFS. If the DFS hits a marked edge, we have successfully found a cycle and are done.

This algorithm just runs a DFS on graph G taking O(|V| + |E|) which is linear time.

(b) Next, suppose you are given a formula in 2CNF with n variables $x_1, ..., x_n$. Construct a graph with 2n vertices, one for each literal, and for every clause construct two edges as follows:

If the clause is of the form $(x_i \vee x_j)$, add the directed edges $(\neg x_i, x_j)$ and $(\neg x_j, x_i)$.
If the clause is of the form $(\neg x_i \vee x_j)$, add directed edges (x_i, x_j) and $(\neg x_i, \neg x_j)$.

In general, if the clause is (a, b), add directed edges (¬a, b) and (¬b, a).
Describe how you would use your algorithm from part (a) to determine if the 2CNF formula is satisfiable and prove that your algorithm for satisfiability is correct

Once you convert the 2CNF to a graph you would check for a cycle containing both a variable and its negation using a depth-first search. If such a cycle exists, the formula must be unsatisfiable. For example, if a cycle of the form

A → ... → ¬A → ... → A were to exist, that means that the directed edges (A → ¬A) AND (¬A → A) would have to exist. They would have to simplify to true in order for the 2CNF to be satisfiable. Clearly, this is not possible because the boolean value of a variable flips when it is negated, so a contradiction arises since a variable and its negation cannot both be true. Hence, if a cycle exists containing a variable and its negation in a 2CNF graph, that 2CNF must be unsatisfiable. If a cycle was found and it does not carry such a contradiction, there must exist a case that satisfies the 2CNF. This is because every variable in the cycle can be set to true or false, and there would be no negated version anywhere else in that cycle that could contradict it. Therefore, there will be a valid combination that will satisfy the 2CNF formula.