Name: Rollno:

CSE340: Theory of Computation (Homework Assignment 4)

Due Date: 7th November, 2017 (in class)

Total Number of Pages: 6 Total Points 55

Question 1. Which of the following languages are decidable/undecidable? Prove your answer by either giving an algorithm or a proof of undecidability.

(a) (5 points) $L_1 = \{\langle M, N\rangle \mid M, N \text{ are two TMs and } M \text{ takes fewer steps than } N \text{ on input } \epsilon\}$

Solution: Undecidable.

Consider the language

$$L_\epsilon = \{\langle M\rangle \mid M \text{ accepts } \epsilon\}.$$

We will use the fact that L_ϵ is undecidable.

Claim 1. $L_\epsilon \leq_m L_1$.

We will construct a computable function f that takes as input $\langle M\rangle$ and produces an output $\langle M_1, M_2\rangle$ such that $\epsilon \in L(M) \iff M_1$ takes fewer steps than M_2 on ϵ.

The reduction function f

Input: $\langle M\rangle$

1. Construct a TM M_1 that does the following on every input:
   - Simulate M on ϵ.
   - If M accepts ϵ then *accept* and if M rejects ϵ then go into an infinite loop.
2. Construct a TM M_2, that goes into an infinite loop on every input (in particular, on ϵ).

Output: $\langle M_1, M_2\rangle$

Note that M_2 always takes infinite number of steps on ϵ.

Proof of correctness

Now,

$$\epsilon \in L(M) \implies M_1 \text{ accepts } \epsilon \text{ in finite number of steps} \implies M_1 \text{ takes fewer steps than } M_2 \text{ on } \epsilon$$

$$\epsilon \notin L(M) \implies M_1 \text{ goes into an infinite loop on } \epsilon \implies M_1 \text{ takes same number of steps as } M_2 \text{ on } \epsilon$$

Therefore, $\epsilon \in L(M) \iff M_1$ takes fewer steps than M_2 on ϵ and hence $L_\epsilon \leq_m L_1$. This proves that L_1 is undecidable.

(b) (5 points) $L_2 = \{\langle M \rangle \mid M$ takes at most 2^{340} steps on some input$\}$

Solution: Decidable.

Algorithm

Input: $\langle M \rangle$

1. On every input of length at most 2^{340}, run M for at most 2^{340} steps.
2. If M accepts any such input within this time, then *accept* else *reject*.

Note that if an input is accepted within 2^{340} steps then only the first 2^{340} bits of the input are of any relevance to the algorithm. Hence we need to consider only inputs of length at most 2^{340}.

(c) (5 points) $L_3 = \{\langle M \rangle \mid$ there are infinitely many TMs equivalent to $M\}$

Solution: Decidable.
We use the fact that every TM has infinitely many equivalent TMs. Hence every TM is accepted (we only need to check whether the input correctly encodes a TM or not).

(d) (5 points) $L_4 = \{\langle M, N \rangle \mid L(M) \cap L(N)$ is infinite$\}$

Solution: Undecidable.
Consider the language

$$\overline{FIN} = \{\langle M \rangle \mid L(M) \text{ is infinite}\}.$$

We will use the fact that $\overline{FIN}$ is undecidable.
Claim 2. $\overline{FIN} \leq_m L_4$.
We will construct a computable function f that takes as input $\langle M \rangle$ and produces an output $\langle M_1, M_2 \rangle$ such that $L(M)$ is infinite $\iff L(M_1) \cap L(M_2)$ is infinite.

The reduction function f

Input: $\langle M \rangle$

1. Set $M_1 := M$.
2. Construct a TM M_2 that accepts all inputs (i.e. $L(M_2) = \Sigma^*$).

Output: $\langle M_1, M_2 \rangle$

Proof of correctness

Now,

$$L(M) \text{ is infinite} \iff L(M_1) \cap \Sigma^* \text{ is infinite} \iff L(M_1) \cap L(M_2) \text{ is infinite}$$

Therefore, $L(M)$ is infinite $\iff L(M_1) \cap L(M_2)$ is infinite and hence $\overline{FIN} \leq_m L_4$. This proves that L_4 is undecidable.

Question 2. (8 points) In class we showed that REG_{TM} is not Turing recognizable. Prove that REG_{TM} is also not co-Turing recognizable.

Solution: Since we have seen that A_{TM} is Turing recognizable and undecidable, therefore A_{TM} is not co-Turing recognizable. Therefore it is enough to show that $A_{TM} \leq_m REG_{TM}$.

We will construct a computable function f that takes as input $\langle M, w\rangle$ and produces an output $\langle M'\rangle$ such that M accepts $w \iff L(M')$ is regular.

The reduction function f

Input: $\langle M, w\rangle$

Construct a TM M' that does the following on every input x:

- If x is of the form 0^n1^n then *accept*.
- Simulate M on w.
- If M accepts w then *accept* and if M rejects w then *reject*.

Output: $\langle M'\rangle$

Proof of correctness

Now,

$$M \text{ accepts } w \implies L(M') = \Sigma^* \implies L(M') \text{ is regular}$$

$$M \text{ does not accept } w \implies L(M') = \{0^n1^n \mid n \geq 0\} \implies L(M') \text{ is not regular}$$

Therefore, M accepts $w \iff L(M')$ is regular and hence $A_{TM} \leq_m REG_{TM}$. This proves that REG_{TM} is not co-Turing recognizable.

Question 3. One of the following two languages is Turing recognizable and the other is not. State which is which and give proofs for your answer.

(a) (6 points) $A = \{\langle M\rangle \mid |L(M)| \geq 340\}$

Solution: A is Turing recognizable.

Turing recognizable algorithm for A

Input: $\langle M\rangle$

1. Initialize a counter c to 0.
2. Iterate through all strings x in a lexicographical order by running the TM M on x.
3. Whenever a string is accepted increment the counter.
4. If the counter reaches 340 then *accept*

Note: We execute step ii. of the above algorithm by running the strings in a "diagonal manner" as discussed in class. Consider a matrix whose rows contain the list of all strings and whose columns corresponding to the number of steps a string is run for. We cover the matrix by covering each diagonal at a time. By this approach, if M halts on a string then it would halt after a certain number of steps, and every string on which M halts will be covered by this approach.

Proof of correctness

On every input $\langle M \rangle$ such that $|L(M)| \geq 340$, the above algorithm will halt and accept because of the reason mentioned above.

(b) (6 points) $B = \{\langle M \rangle \mid |L(M)| \leq 340\}$

Solution: We use the fact that $\overline{A_{TM}}$ is not Turing recognizable.
Claim 3. $\overline{A_{TM}} \leq_m B$.
We will construct a computable function f that takes as input $\langle M, w \rangle$ and produces an output $\langle M' \rangle$ such that M does not accept $w \iff |L(M')| \leq 340$.

The reduction function f

Input: $\langle M, w \rangle$

Construct a TM M' that does the following on every input x:

- Simulate M on w.
- If M accepts w then *accept* x.
- If M rejects w then *reject* x.

Output: $\langle M' \rangle$

Proof of correctness

Now,

$$M \text{ does not accept } w \implies L(M') = \emptyset \implies |L(M')| \leq 340$$

$$M \text{ accepts } w \implies L(M') = \Sigma^* \implies |L(M')| > 340$$

Therefore, M does not accept $w \iff |L(M')| \leq 340$ and hence $\overline{A_{TM}} \leq_m B$. This proves that B is not Turing recognizable.

Question 4. Prove that the following problems are NP-complete.

(a) (7 points) $\mathsf{LPATH} = \{\langle G, s, t, k \rangle \mid G$ has a simple path of length at least k from s to $t\}$

Solution:

1. Showing that LPATH is in NP.
   - Certificate: A sequence of vertices $(v_1, \ldots, v_m)$.
   - Polynomial time verification:
     - Check if $m \geq k$
     - Check if $v_1 = s$ and $v_m = t$.
     - Check if (v_i, v_{i+1}) is an edge for all i.
     - For every $i \neq j$, check if $v_i \neq v_j$.
     - If all the above checks are satisfied then *accept*, else *reject*.

2. Choosing a suitable NP-complete problem. We will show that

$$\mathsf{UHAMPATH} \leq_p \mathsf{LPATH}.$$

3. The reduction. Let $\langle G = (V, E), s, t\rangle$ be an instance of UHAMPATH. We will construct an instance of LPATH, $\langle G' = (V', E'), s', t', k\rangle$ as follows:

$$\begin{aligned} G' &= G \\ s' &= s \\ t' &= t \\ k &= n \qquad \text{where } n \text{ is the number of vertices in } G \end{aligned}$$

4. The construction of G' can be achieved in linear time from the graph G.

5. Proof of correctness.

   G has a Hamiltonian path from s to t if and only if the length of the path is the number of vertices in G.

(b) (8 points) $\mathsf{DS} = \{\langle G, k\rangle \mid \exists\, S \subseteq V(G) \text{ with } |S| \leq k, \text{ and every vertex in } V(G) \setminus S \text{ has a neighbor in } S\}$
(Hint: You may try reducing VertexCover to DS)

Solution:

1. Showing that DS is in NP.
   - Certificate: A subset of vertices $S = \{v_1, \ldots, v_m\}$.
   - Polynomial time verification:
     - Check if $m \leq k$
     - For every $v \in V(G) \setminus S$, check if v has a neighbor in S.
     - If the above checks are satisfied then *accept*, else *reject*.

2. Choosing a suitable NP-complete problem. We will show that

$$\mathsf{VertexCover} \leq_p \mathsf{DS}.$$

3. The reduction. Let $\langle G = (V, E), k\rangle$ be an instance of VertexCover. We will construct an instance of DS, $\langle G' = (V', E'), k'\rangle$ as follows:

   The idea is to replace every edge of G with a triangle.

$$\begin{aligned} V' &= V \cup \{v_{e_i} \mid e_i \in E\} \\ E' &= E \bigcup_{e=(u,v)\in E} \{(u, v_e), (v, v_e)\} \\ k' &= k \end{aligned}$$

4. The construction of G' can be achieved in linear time from the graph G.

5. Proof of correctness.

   If G has a vertex cover of size k then the same set of vertices will form a dominating set in G'. Let v be a vertex in G' that is not in the dominating set. If v is in G, then there exists some neighbor of v in the vertex cover and hence in the dominating set. If v is not in G, then one of the two neighbors of v must be in the vertex cover (hence in the dominating set) as the corresponding edge must have been covered by one of its two endpoints. Hence G' has a dominating set of size k.

   If G' have a dominating set S' of size k then there is a corresponding dominating set S of G' of size k that has vertices only from G. This is because if v_e is a vertex in S' that is not in G then we can replace it with one of its two neighbors to get the set S. This would still ensure that v_e has a neighbor in the dominating set S. Now without loss of generality we have a dominating set S of G' consisting of vertices only from G. If an edge $e = (u, v) \in G$ is not covered by S then $v_e \in G'$ will not have a neighbor in S (u and v are its only two neighbors). This contradicts the fact that S is a dominating set. Hence S forms a vertex cover in G.