CSCE-637 Complexity Theory

Fall 2020

Instructor: Dr. Jianer Chen
Office: HRBB 338C
Phone: 845-4259
Office Hours: TR 1:30 pm–3:00 pm

Solutions to Assignment #1

1. Write a detailed description of a 1-tape Turing machine that accepts the following language:

$$\text{UnEqual} = \{x\#y \mid x \text{ and } y \text{ are binary numbers such that } x \neq y\}.$$

Solution. This problem seems the "complement" of the problem we have discussed in class. However, you need to be careful: an instance that is not in a valid format, such as "101#110#0" should be a NO-instance for both this problem and the problem discussed in class. Therefore, simply reversing the states q_{acc} and q_{rej} is not correct.

The purpose of this problem is to let you gain experience in constructing a Turing machine. After this, you should realize that constructing a Turing machine is very similar to writing a program using conventional programming languages such as a machine assembly language or C.

The transition function δ of the Turing machine $M = (Q, \Sigma, \delta, q_s, q_{acc}, q_{rej})$ is given as follows, in which you can easily identify the state set Q and the alphabet Σ. In particular, q_s is the starting state, and q_{acc} and q_{rej} are the accepting and rejecting states, respectively.

(1)	$\delta(q_s, 0/1) = (q_{11}, \bar{0}/\bar{1}, R)$	$\delta(q_{11}, 0/1) = (q_{11}, 0/1, R)$
(2)	$\delta(q_{11}, \#) = (q_{12}, \#, R)$	$\delta(q_{12}, 0/1) = (q_{12}, 0/1, R)$
(3)	$\delta(q_{12}, \_) = (q_{13}, \_, L)$	$\delta(q_{13}, 0/1/\#) = (q_{13}, 0/1/\#, L)$
(4)	$\delta(q_{13}, \bar{0}/\bar{1}) = (q_2, 0/1, -)$	
(5)	$\delta(q_2, \#) = (q_{3-}, \#, R)$	$\delta(q_{3-}, \bar{0}/\bar{1}) = (q_{3-}, \bar{0}/\bar{1}, R)$
(6)	$\delta(q_{3-}, 0/1) = (q_{acc}, 0/1, -)$	$\delta(q_{3-}, \_) = (q_{rej}, \_, -)$
(7)	$\delta(q_2, 0) = (q_{20}, \bar{0}, R)$	$\delta(q_{20}, 0/1) = (q_{20}, 0/1, R)$
(8)	$\delta(q_{20}, \#) = (q_{30}, \#, R)$	$\delta(q_{30}, \bar{0}/\bar{1}) = (q_{30}, \bar{0}/\bar{1}, R)$
(9)	$\delta(q_{30}, 0) = (q_4, \bar{0}, L)$	$\delta(q_{30}, 1/\_) = (q_{acc}, \_, -)$
(10)	$\delta(q_2, 1) = (q_{21}, \bar{1}, R)$	$\delta(q_{21}, 0/1) = (q_{21}, 0/1, R)$
(11)	$\delta(q_{21}, \#) = (q_{31}, \#, R)$	$\delta(q_{31}, \bar{0}/\bar{1}) = (q_{31}, \bar{0}/\bar{1}, R)$
(12)	$\delta(q_{31}, 1) = (q_4, \bar{1}, L)$	$\delta(q_{31}, 0/\_) = (q_{acc}, \_, -)$
(13)	$\delta(q_4, \bar{0}/\bar{1}) = (q_4, \bar{0}/\bar{1}, L)$	$\delta(q_4, \#) = (q_5, \#, L)$
(14)	$\delta(q_5, 0/1) = (q_5, 0/1, L)$	$\delta(q_5, \bar{0}/\bar{1}) = (q_2, \bar{0}/\bar{1}, R)$

For all other cases not listed above, let $\delta(*, *) = (q_{rej}, *, -)$.

We use the symbols $\bar{0}/\bar{1}$ to mark the 0/1's, resp., that have been compared by the machine.

Lines (1)-(4) are used to check that the input is in a valid format $x\#y$, where x and y are non-empty binary numbers (if you also allow x and y to be empty, you need to add a couple of more statements). State q_{11} scans the binary bits before the symbol #, while state q_{12} scans the binary bits after #. Note that only when the input is in the valid format $x\#y$, the machine enters the state q_2, which begins the comparison of the next bit of x and y.

There are three cases:

(1) The state q_2 sees #, which means that all bits of x are scanned. Then the machine passes through all scanned bits in y (using state q_{3-}, see line (5)). If there are still unmarked bits in y, then the machine accepts (i.e., $x \neq y$). Otherwise, the machine rejects. See line (6).

(2) The state q_2 sees 0. Then the machine passes through all remaining bits in x (line (7)) and all marked bits in y (line (8)). If the next bit in y is also 0, then this bit does not differ x from y, so the machine uses state q_4 to go back to the next unmarked bit in x, and enters state q_2 again for comparing the next bit of x and y (first statement in line (9), and lines (13)-(14)). If the next bit of y is 1 (or if y has no more un-compared bit), then we have identified $x \neq y$, so the machine stops and accepts (the second statement in line (9)).

(3) The state q_2 sees 1. This case is handled similarly as case (2) (see lines (10)-(12)). □

2. A language L is *decidable* if there is a Turing machine that always halts and accepts L. We say that *a language L_1 is reducible to another language L_2* if there is a Turing machine (i.e., an algorithm) that always halts, and on any (yes or no) instance x_1 of L_1, produces an instance x_2 of L_2 such that x_1 is a yes-instance of L_1 if and only if x_2 is a yes-instance of L_2.

Consider the following language:

$$\textsc{Test} = \{(M; x, y) \mid \text{on input } x, \text{the Turing machine } M \text{ outputs } y\}.$$

Show that the problem TEST is undecidable. (*Hint:* write an algorithm that reduce HALTING problem to TEST, and use the fact that HALTING is undecidable.)

Solution. We show how HALTING is reduced to TEST. For this, we need to give an algorithm $\mathcal{A}_{red}$ that on an input $z_1 = (M_1, x_1)$ that is an intance of HALTING, produces an output $z_2 = (M_2, x_2, y_2)$ that is an instance of TEST, such that z_1 is a YES-instance for HALTING if and only if z_2 is a YES-instance for TEST.

The algorithm $\mathcal{A}_{red}$ works as follows: on input $z_1 = (M_1, x_1)$ that is an instance of HALTING, the algorithm $\mathcal{A}_{red}$ replaces each stop-statement in M_1 by the following statements: (1) statements that erase whatever written on the output tape then write a single symbol '0' on the output tape; and then (2) a stop-statement. Let this new TM be M_2. By this construction, it is easy to see that the TM M_1 halts on input x_1 if and only if the TM M_2 on the same input x_1 outputs a single symbol '0'. That is, (M_1, x_1) is a YES-instance for HALTING if and only if $(M_2, x_1, 0)$ is a YES-instance for TEST.

The TM M_2 can be easily constructed from the TM M_1, so the algorithm $\mathcal{A}_{red}$ always halts.

Now we prove that TEST is undecidable. Assume the contrary that the problem TEST is decidable. Hence, there exists an algorithm $\mathcal{A}_{test}$ that solves TEST and always halts.

Now consider the following algorithm $\mathcal{A}_{halt}$ for HALTING: given an instance (M_1, x_1) for HALTING, the algorithm $\mathcal{A}_{halt}$ first calls the algorithm $\mathcal{A}_{red}$ to take the input (M_1, x_1) and produce an instance $(M_2, x_1, 0)$ for TEST, then call the algorithm $\mathcal{A}_{test}$ to decide if $(M_2, x_1, 0)$ is a YES for TEST. Since both algorithms $\mathcal{A}_{red}$ and $\mathcal{A}_{test}$ always halt, the algorithm $\mathcal{A}_{halt}$ also always halts. Moreover, by the above discussion, (M_1, x_1) is a YES-instance for HALTING if and only if $(M_2, x_1, 0)$ is a YES-instance for TEST. As a result, we would have the algorithm $\mathcal{A}_{halt}$

that solves the HALTING problem and always halts, but this contradicts the fact that HALTING is undecidable. This contradiction proves that the problem TEST is undecidable. □

3. Prove: if $\mathbf{P} = \mathbf{NP}$, then every non-trivial problem in $\mathbf{P}$ is $\mathbf{NP}$-complete. A problem is *non-trivial* if it has both YES-instances and NO-instances.

Solution. Let Q be any non-trivial problem in $\mathbf{P}$. By definition, there exist a YES-instance x_{yes} and a NO-instance x_{no} for Q.

To prove that Q is $\mathbf{NP}$-complete, we need to prove (1) Q is in $\mathbf{NP}$, and (2) Q is $\mathbf{NP}$-hard, i.e., every problem Q' in $\mathbf{NP}$ can be polynomial-time reduced to Q.

Since Q is in $\mathbf{P}$, and $\mathbf{P} = \mathbf{NP}$, so Q is in $\mathbf{NP}$.

To prove that Q is $\mathbf{NP}$-hard, let Q' be any problem in $\mathbf{NP}$. By the assumption $\mathbf{P} = \mathbf{NP}$, Q' is in $\mathbf{P}$. Thus, there is a deterministic polynomial-time algorithm A' that solves Q'. Now consider the following reduction R from Q' to Q:

<u>Reduction R</u>
Input: an instance x' of Q'
1. Call the algorithm A' to decide if x' is a YES-instance of Q';
2. If x' is YES for Q' then let $x = x_{yes}$ else let $x = x_{no}$;
3. Output(x).

Clearly the output x of R is a YES-instance for Q if and only if the input x' is a YES-instance for Q'. Moreover, since the algorithm A' is a deterministic polynomial-time algorithm, the Reduction R is also a deterministic polynomail-time algorithm. In conclusion, this shows that Reduction R is a polynomial-time reduction from the problem Q' to the problem Q, that is, $Q' \leq_m^p Q$. Since Q' is any problem in $\mathbf{NP}$, this proves that the problem Q is $\mathbf{NP}$-hard. Combining this with the fact that Q is in $\mathbf{NP}$, we conclude that Q is $\mathbf{NP}$-complete. □

4. Give a detailed proof for the following statement: if $L_1 \leq_L L_2$ and $L_2 \leq_L L_3$, then $L_1 \leq_L L_3$.

Solution. To show $L_1 \leq_L L_3$, we construct a log-space TM M_{1-3} that on an input x_1 produces an output x_3 such that x_1 is a YES-instance for L_1 if and only if x_3 is a YES-instance of L_3.

Since $L_1 \leq_L L_2$, there is a log-space TM M_{1-2} that on an input x_1 produces an output x_2 such that x_1 is a YES-instance for L_1 if and only if x_2 is a YES-instance of L_2. Similarly, since $L_2 \leq_L L_3$, there is a log-space TM M_{2-3} that on an input x_2 produces an output x_3 such that x_2 is a YES-instance for L_2 if and only if x_3 is a YES-instance of L_3.

In principle, the TM M_{1-3} that reduces L_1 to L_3 works as follows: on an instance x_1 of L_1, call the TM M_{1-2} to produce an instance x_2 of L_2, then call the TM M_{2-3} on input x_2 to produce an instance x_3 of L_3. By the definitions of the TMs M_{1-2} and M_{2-3}, it is easy to see that x_1 is a YES-instance of L_1 if and only if x_3 is a YES-instance of L_3.

Some details should be clarified here: the complexity of the log-space TM M_{2-3} is measured by the length of its input x_2, not x_1. Thus, M_{2-3} on input x_2 requires work-space $O(\log |x_2|)$. On the other hand, since M_{1-2} is a log-space TM, which, as we proved in class, runs in polynomial time. Since in each step, M_{1-2} can write at most one symbol on its output tape, the length $|x_2|$ of the output x_2 of M_{1-2} on input x_1 is bounded by a polynomial of $|x_1|$, i.e., $|x_2| \leq d \cdot |x_1|^c$, where c and d are constants. Therefore, $\log |x_2| \leq O(\log |x_1|)$. As a result, the space taken by the TM M_{2-3} on input x_2 is $O(\log |x_2|) = O(\log |x_1|)$. Thus, in the above process, both machine M_{1-2} on input x_1 and machine M_{2-3} on input x_2 require work-space $O(\log |x_1|)$.

The remaining difficulty is where we place the intermediate result x_2. Note that for machine M_{1-2}, x_2 is written on its output tape, which does not count for the work space of M_{1-2}, while for machine M_{2-3}, x_2 is given on its input tape, which also does not count for the work space of M_{2-3}. Now our proposed TM M_{2-3} needs both reading and writing x_2 so that will require space in its work tape. However, the length $|x_2|$ can be too large to fit in $O(\log |x_1|)$ space.

We apply the following trick to bound the work space of M_{1-3} by $O(\log n) = O(\log |x_1|)$. We let M_{1-3} simulate M_{2-3} directly (without given the entire input x_2). The machine M_{1-3} keeps the position H_2 of the input head for the machine M_{2-3} (note that $1 \leq H_2 \leq |x_2|$ so H_2 can be given by $O(\log |x_1|)$ bits, stored in the work tape of M_{1-3}). The value of H_2 can be easily updated when the input head of M_{2-3} moves (by adding 1 to or subtracting 1 from H_2). When M_{1-3} simulates M_{2-3}, in each step M_{1-3} only needs to know the H_2-th symbol on the input tape. In order to get that symbol, M_{1-3} switches to simulate M_{1-2} on input x_1 (note that x_1 is on the input tape of M_{1-3}). However, during the simulation of M_{1-2}, M_{1-3} does not actually write on the output tape. Instead, it remembers the position of the output head H_1 of M_{1-2}. Only when M_{1-2} writes on the position H_2, i.e., when $H_1 = H_2$, M_{1-3} remembers the symbol written by M_{1-2} on that position. Therefore, when the entire computation of M_{1-2} on input x_1 is completed, the TM M_{1-3} knows exactly what symbol is on the position H_2 on the output tape of M_{1-2}, which is the H_2-th symbol in the input x_2 of M_{23}. Now with this input symbol, the TM M_{1-3} switches back to the simulation of M_{2-3} on x_2. Note that the total work space required by M_{1-3} is the work space of M_{1-2} plus the work space of M_{2-3}, which in total is $O(\log |x_1|)$. Thus, the TM M_{1-3} on input x_1 produces the output x_3, using $O(\log |x_1|)$ working space, such that x_1 is a YES-instance of L_1 if and only if x_3 is a YES-instance of L_3.

This completes the proof that $L_1 \leq_L L_3$. $\square$