Australian National University

COMP3630 Theory of Computation

Assignment 03

Edited by LaTeX

College of Engineering and Computer Science

©*Author*

Jimmy Lin

u5223173

¶ *Lecturer*

Jinbo Huang

¶ *Lecturer*

Dirk Pattinson

† *Release Date*

May. 7 2013

‡ *Due Date*

May. 20 2013

τ *Time Spent*

18h

May 20, 2013

Contents

1 Turing Machine Design

Design a Turing machine that shifts the entire input string to the right by one place, under the following conditions: The input alphabet is $\{0, 1\}$ and the Turing machine has a single tape with a single track. Describe the Turing machine in words, and draw its transition diagram.

1.1 Describe the Turing machine in words

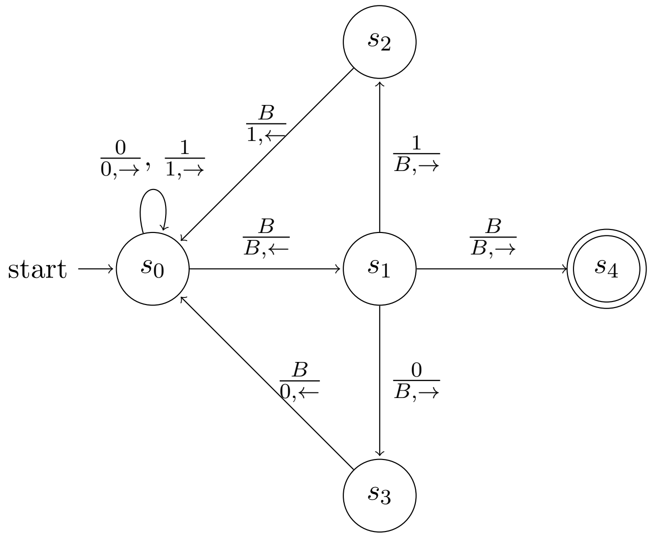
Since the functionality of the desired Turing Machine is to shift the entrie input string to the right by one place, it is not applicable to directly read symbol from left cell, sweep it and write to the right-neighboured cell (the right one would be covered).

Rather, we should first move the head of tape rightward without changing any symbol until meeting the first blank cell and then execute the copy task leftward recursively. After that, in order to copy content of each cell to the neighbour cell on its right, obviously, we ought to sweep the objective cell, and move right. Then write down the recorded symbol and move back to the objective cell. Finally move left to the next objective cell and repeat the copy task described above if the symbol in the tape is not blank.

Note that since only one memory device is allowed, we have to use state to record what symbol we have just read, and it is also practical to apply this approach for an alphabet with only two characters.

1.2 Draw the transition diagram

The transition diagram is shown in the following figure,



start
s_0
s_1
s_2
s_3
s_4
$\frac{0}{0,\rightarrow}$, $\frac{1}{1,\rightarrow}$
$\frac{B}{1,\leftarrow}$
$\frac{1}{B,\rightarrow}$
$\frac{B}{B,\leftarrow}$
$\frac{B}{B,\rightarrow}$
$\frac{B}{0,\leftarrow}$
$\frac{0}{B,\rightarrow}$

Fig. TM M_0 with acceptance by final state

Note that when we first sweep the cell to blank symbol (see s_1) before copying its content to the adjacent right cell. In this manner, it is unnecessary to specify more transitions because the cell we just copy must be blank (when copy task starts up, TM M must read blank symbol at state s_0 because we sweep the cell whose content is just copyed. And in s_2 and s_3, only blank symbol will be read.). Finally, we terminate the Turing Machine when there is blank symbol to read at state s_1.

2 Enumerating a Recursively Enumerable Language

Given a Turing machine M that accepts a language L, informally but clearly show that a Turing machine M can be constructed to enumerate all members of L in the following sense:

(i) Whenever M' enters a special state p, the string to the left of the tape head is a member of L.

(ii) Every member of L appears on the tape at some point in the aforementioned way.

Keep in mind that the Turing machine M may not halt on all inputs.

2.1 Construct special TM M'

The diagram that summarises the description of construction is shown in the following.

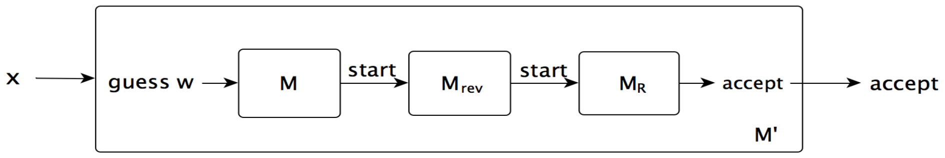


Fig.2. Construction for Turing Machine M'

Note that the special state p in our proposal is instantiated to be the accepting state of M'. And Turing Machine M' use the same tape to simulate all three TMs, M, M_{rev}, M_R.

Specifically, construct the Turing Machine M' as follows:

- Ignore its own input x and Guess arbitrary w
- Simulate given TM M on guessed string w. The functionality of TM M here is to disable the TM M' with the string w that is not accepted by given TM M.
  - if the accepting state of M is reached, which means $w \in L(M)$, TM M' starts to simulate TM M_{rev} using the same tape. (tape symbols and tape head remained unchanged and are used as input of M_{rev})
  - If the accepting state of M is never reached, that is, $w \notin L(M)$. Therefore, it would be impossible to for TM M' to enter its accepting state.
- Simulate M_{rev} if activated. And the TM M_{rev} is constructed by the reversing all the transitions of M, make start state of M as accepting state and replace accepting state of M to be start state. The functionality of TM M_{rev} is to recover the original input w to TM M.
  - For all strings that are input to M_{rev} with specific position of tape head, the accepting state of M_{rev} will be reached, that is, they must be accepted by M_{rev}.
- Simulate M_R if activated. The TM M_R is designed to move the tape head of TM M' rightward until it meets first blank symbol. This is easy to simulate. In this manner, the recovered string w, which is accepted by M can be in the left of tape head.
  - If TM M_R meets first blank symbol, goes into accepting state of M_R and trigger the acceptance of TM M'.

Next explain why the Turing Machine M' we just construct satisfies the requirement (i) and (ii).

For requirement (i). Since only if guessed string $w \in L(M)$, w will make M into its accepting state, thereby activating M_{rev}, and all w that activates M_{rev} would lead M_{rev} to its accepting state, it can be derived that the string w will activate M_R only if $w \in L(M)$. Since all strings that activate M_R will lead M_R to accepting state and thus trigger acceptance of M', combined with the fact that M_R only move the position of the tape head without changing cells' content. Therefore, if TM M' enter its accepting state, the guessed w must be in $L(M)$.

For requirement (ii). TM M' will guessed all possible strings in the very beginning. And if the guessed string $w \in L(M)$, as described in specfication of M', this string will definitely pass the filtering TM M, then be recovered by M_{rev} and finally be set in the left of tape head of M' by M_R.

3 Proving Undecidability

Consider the following theorem and attempted proof thereof. Identify the flaw in the attempted proof, and give a correct proof.
Theorem: The set of all (encodings of) Turing machines that have a *useless* state is undecidable, where a useless state is defined as a state that is never visited on any input string.
Attempted proof (sketch): Reduce the universal language to this problem. Given (M, w), construct a Turing machine Q that replaces its input string with (M, w), simulates the universal Turing machine, and enters the accepting state q_{accept} if and only if the (simulated) universal Turing machine accepts (M, w). The reduction works as the Turing machine Q will have a useless state (q_{accept}) if and only if the Turing machine M does not accept the string w.

3.1 Flaw of attempted proof

Flaw: The reduction did not relate the problem of whether given TM M accept certain string w to our objective problem, which is whether one coded TM will have useless state. Specifically, if M accept the string w, the accepting state of Q will be reached, which means q_{accept} is not useless. But under this circumstance, we still have no idea about whether the TM Q has useless state. (there may exist some other states of TM Q to be useless or may not.) Therefore, what the reduction in the attempted proof do is to relate the problem of whether TM M accepts string w to the problem of whether accepting state of Q, q_{accept} is useless state, but not our objective problem, that is, whether the TM Q has a useless state.

3.2 Correct Proof

Correct Proof: We first will prove that the Turing Machine M_c which take a pair of coded TM and one state from that TM as input is undecidable by reducing L_u to $L(M_c)$. And then we reduce to the problem of given TM and one state of that TM, whether the state is useless state to the problem of whether a TM has one useless state.

Assume that there is a TM M_c decide the language $L_c = \{(M, q) \mid \text{q is useless state in TM } M\}$. By this assumption, we can reduce the L_u to L_c, and show that if there exist a M_c decide the L_c, there must be one TM M' to decide L_u. We construct the Turing Machine M' by using M_c as follows.

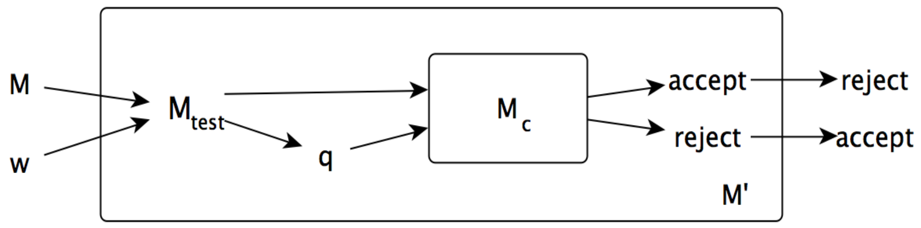


Fig.3. Construction for M' using M_c

Note that TM M' does not run M_{test} but just produces the code of TM M_{test}.

Description for the constructed M' is as follows

- What TM M' do in first step is to generate the coded TM M_{test}. The TM M_{test} has its own input x. But M_{test} ignore its input x and simulate M on string w. If M accepts string w, then M_{test} accepts (M, w). Note that if M does not accept w, M_{test} may not halt, but here we do not run M_{test}, just generate code of M_{test}.
- Let q be the accepting state of M_{test}.
- Simulate M_c on (M_{test}, q). If M_c rejects, M' goes to accepting state. If M_c accepts, M' halts on non-accepting state. (note that this is valid because we have assumed that M_c is decidable.)

As we can see, M' is a TM taking (M, w) as its input. And M' accepts iff $w \in L(M)$. Conversely, M' rejects iff $w \notin L(M)$. (for those $w \notin L(M)$, TM M' always halts). That is to say, M' can decide the $L_u = \{(M, w) \mid w \in M\}$, which contradicts the theorem that universal language L_u is not decidable. Thus, we should negate the original assumption to have the following conclusion.

$$L_c = \{(M, q) \mid q \text{ is useless state in TM } M \text{ is undecidable}\}. \tag{1}$$

After acquiring a fairly important conclusion, let's go back to prove the undecidability of $L_a = \{M \mid M$ has at least one useless state.$\}$ It is evident that we achieve the proof by reducing L_c to L_a. The specific way is as follows.

Given a TM M, to know whether this TM has a useless state, we can traverse all the state of TM M, and ask M_c whether q_i in TM M is a useless state. The final result to the question of whether TM M has one useless state is "no", if and only if all the answers from M_c are "no". Conversely, the result to taht question is "yes" if and only if there is one "yes" from M_c. Therefore, L_c is reducible to L_a in polynomial time. And since we have proven that L_c is undecidable, we can conclude that L_a is also undecidable.

4 Proving Non-Recursive Enumerability

Show that the following is not recursively enumerable: $\{(M_1, M_2) \mid L(M_1) \cap L(M_2) = \emptyset\}$, i.e., the set of pairs of Turing machines the intersection of whose languages is empty.

First of all, we name the objective language to be L_{ie} (subscript means intersection is empty). Formally,

$$L_{ie} = \{(M_1, M_2) \mid L(M_1) \cap L(M_2) = \emptyset\} \tag{2}$$

it should be illustrated the property of complement of universal language and universal language. That is, $\overline{L_u} = \{(M, w) \mid w \notin L(M)\}$ is not recursively enumerable since $L_u = \{(M, w) \mid w \in L(M)\}$ is recursively enumerable but not recursive. Thus, there is no Turing Machine accepting language $\overline{L_u}$ since it is non-RE.

Assume that L_{ie} is recursively enumerable. And suppose one Turing Machine Q that accept the objective language L_{ie}.

Then we start to construct one Turing Machine M_{new} to relate the binary coded string (M, w) to the Turing Machine Q.

Let M_{new} translate input string w to one Turing Machine $N(w)$ that only accept w itself, that is

$$L\big(N(w)\big) = \{w\} \tag{3}$$

Note that the way to construct such a w can be just to compare the character of its input one-by-one with a stored copy of w.

Next we apply the property of the Turing Machine Q, which is assumed in the very beginning. That is,

$$L(M) \cap L\big(N(w)\big) = \emptyset \tag{4}$$

By what we derived in (3), we have

$$L(M) \cap \{w\} = \emptyset \tag{5}$$

which is tantamount to the following equation since w is the only element of set $\{w\}$,

$$w \notin L(M) \tag{6}$$

that is,

$$L(M_{new}) = \{(M, w) \mid w \notin L(M)\} \tag{7}$$

which means M_{new} accept the language $\overline{L_u}$, which contradicts our previous statement that there exists no Turing Machine accepting $\overline{L_u}$. Therefore, we should negate the assumption and have

$$L_{ie} \text{ is not recursively enumerable.} \tag{8}$$

5 Boolean Encodings of Graph Properties

Suppose G is an undirected graph of four nodes: 1, 2, 3, and 4. Let x_{ij} , for $1 \leq i < j \leq 4$, be a Boolean variable that we interpret as saying there is an edge between nodes i and j. The expression $x_{12}x_{23}x_{34}x_{14}+x_{13}x_{23}x_{24}x_{14}+x_{13}x_{34}x_{24}x_{12}$, for example, says that the graph G has a Hamilton circuit. In general, a Boolean expression over the x_{ij} variables describes a property of the graph in the sense that a truth assignment to the variables satisfies the expression if and only if it describes a graph having that property. Write expressions for the following properties:

- 1. G contains a clique of size 3 (i.e., a triangle).
- 2. G contains at least one node with no edges.
- 3. G is connected.

5.1 G contains a clique of size 3

Obviously, all graphs that has clique of size three must have one of four possible triangles.

$$x_{12}x_{23}x_{13},\ x_{23}x_{34}x_{24},\ x_{34}x_{14}x_{13},\ x_{14}x_{12}x_{24} \tag{9}$$

Therefore, the following boolean expression would be evaluated to be true if and only if the graph G contains a clique of size 3, that is

$$x_{12}x_{23}x_{13} + x_{23}x_{34}x_{24} + x_{34}x_{14}x_{13} + x_{14}x_{12}x_{24} \tag{10}$$

5.2 G contains at least one node with no edges

Before providing solution to this question, we first introduce one notation $\bar{x}_{ij}$, which means there is no edge from node i to node j.

For all graphs that at least one node has no edges must satisfy one of following boolean term. Each of boolean term shown in the followings represents one possible node with no edge.

$$\bar{x}_{12}\bar{x}_{13}\bar{x}_{14},\ \bar{x}_{23}\bar{x}_{24}\bar{x}_{12},\ \bar{x}_{34}\bar{x}_{23}\bar{x}_{13},\ \bar{x}_{14}\bar{x}_{24}\bar{x}_{34}, \tag{11}$$

Since any graph that has more than one still needs to satisfy one of boolean terms shown above, we have the following boolean expression, which is true if and only if graph G contains at least one node with no edges.

$$\bar{x}_{12}\bar{x}_{13}\bar{x}_{14} + \bar{x}_{23}\bar{x}_{24}\bar{x}_{12} + \bar{x}_{34}\bar{x}_{23}\bar{x}_{13} + \bar{x}_{14}\bar{x}_{24}\bar{x}_{34} \tag{12}$$

5.3 G is connected

Easy to see that all graph with more than 3 edges must be connected and that all graph with less than 3 edges must be non-connected. Thus, we only need to discuss the boolean expression based on those graphs with 3 edges.

Specifically, simplified boolean expression for all connected graphs are presented as follows,

$$\begin{aligned} & x_{12}x_{13}x_{24} + x_{23}x_{13}x_{24} + x_{34}x_{13}x_{24} + x_{14}x_{13}x_{24} \\ & + x_{12}x_{14}x_{13} + x_{23}x_{34}x_{13} + x_{23}x_{14}x_{13} + x_{12}x_{34}x_{13} \\ & + x_{34}x_{14}x_{24} + x_{12}x_{14}x_{24} + x_{23}x_{14}x_{24} + x_{12}x_{34}x_{24} \\ & + x_{23}x_{34}x_{14} + x_{12}x_{34}x_{14} + x_{12}x_{23}x_{14} + x_{12}x_{23}x_{34} \end{aligned} \tag{13}$$

6 Proving $\mathcal{NP}$-Completeness

We know that the Node Cover problem is $\mathcal{NP}$-complete. Show that the following Dominating Set problem is $\mathcal{NP}$-complete: Given a graph G and an integer k, does there exist a subset S of at most k nodes of G such that each node is either in S or adjacent to a node of S?

6.1 Dominating problem is in $\mathcal{NP}$

To prove the the Dominating Set Problem is in $\mathcal{NP}$, we can show one non-deterministic Turing Machine to achieve the task of figuring out whether there is a dominating set with at most k nodes in given graph G.

The non-deterministic Turing Machine works as follows. Non-deterministically choose ("guess") k nodes from given graph G and form nodes set S. The time complexity for this step is $O(klog(n))$. Then check whether all the nodes are in chosen set S or adjacent to nodes set S. This step will cost $O(n^3k)$ in worst case. Once we find one node that is neither in set S, nor adjacent to node set S, we reject the given graph and k. If the dominating set exists, there should be one execution path by which we successfully "guessed" the dominating set and accept. Since the whole algorithm runs in polynomial time, so dominating problem is in $\mathcal{NP}$.

6.2 Node Cover Problem reduction

Now we describe the reduction from the Node Cover Problem to Dominating Set Problem.

Given a graph G, we can replace each edge in G by one triangle to create graph G'. Formally, new graph $G' = (V', E')$, where the set of nodes $V' = V \cup V_e$ and the set of edges $E' = E \cup E_e$. Specifically, $E_e = \{(v_{e_i}, v_k), (v_{e_i}, v_l) | e_i = (v_k, v_l) \in E\}$, and $V_e = \{v_{e_i}, e_i \in E\}$.

Next we first claim and then prove that

$$\text{graph } G \text{ has cover set of size } K \textbf{ if and only if } \text{the graph } G' \text{ has dominating set of size } k \qquad (14)$$

(If part) If G has a node cover S of size K, then the same set of nodes form a dominating set in G'. This is because for each edge (u, v) in graph G, either u or v must be in the dominating cover consisting of those K nodes, otherwise the K-node-cover property of graph G does not hold. And the transformation by replacing edge with triangle will not introduce any inadjacency. That is, for each edge, whatever u or v is in the cover set, newly introduced node x is always adjacent to the node that is in cover set.

(Only-if part) Suppose that there is a dominating set of size K in graph G'. We can specify that dominating set only pick up nodes from the set V. If v_{e_i} was picked, we can replace it by v_k or v_l without increasing its size and changing its nature of being a dominating set. In this manner, for each edge e_i, v_{e_i} must have neighbour, either v_k or v_l in the dominating set. Then, the node v_{e_i}, edge (v_{e_i}, v_l) and edge (v_{e_i}, v_k) can be removed to derive the original graph G with a node cover of size K. The resulted graph has a node cover of size k because we have claimed that, for each triangles, only one edge (v_k, v_l) was reserved and either v_k or v_l is in the dominating set. Thus, the dominating set in G' is the node cover set in graph G.

In summary, since the Dominating Set Problem is in $\mathcal{NP}$ class and the Node Cover Problem, which has been proven to be $\mathcal{NP}$-complete, can be reducible to Dominating Set Problem, it can be concluded that Dominating Set Problem is $\mathcal{NP}$-complete.