Introduction to the Theory of Computation

Final Exam Sample Solutions

Problem 1 (20 pts). Consider $f(n) = e^{-n}, g(n) = \sin n + 2$.

(a) (10 pts) Use the definition of Small-O to check whether $f(n) = o(g(n))$ or not. Do NOT directly calculate the limit. That is, you need to consider/apply the definition of limit.

(b) (10 pts) Use the definition of Big-O to check whether $f(n) = O(g(n))$ or not.

Solution.

(a) Yes.

Given real number $c > 0$, by letting $n_0 = \max(1, \lfloor -\ln c \rfloor + 1)$, we have

$$e^{n_0} > e^{-\ln c} = \frac{1}{c},$$

where the inequality is from monotonicity and the fact $x - 1 < \lfloor x \rfloor$ for any real x. Therefore

$$e^{-n} \leq e^{-n_0} < c \leq c(\sin n + 2),\ \forall n \geq n_0,$$

where the last inequality follows from $\sin n \geq -1$.

(b) Yes. Note that f is decreasing, and $f(0) = e^0 = 1$, so $f(n) \leq 1$ for $n \geq 1$. On the other hand, since $\sin n \geq -1$, $\forall n$, we have $g(n) \geq 1$, $\forall n$. Take $n_0 = 1, c = 1$, then $f(n) \leq 1 \leq 1 \cdot g(n)$, $\forall n \geq 1$.

Common mistake: For (a) you must give n_0, and for (b) you must give c and n_0. If not, it's very likely you don't get any point.

Problem 2 (10 pts). Suppose $f(n) = 2^{O(n^3)}$. Is $f(n)^3 = 2^{O(n^3)}$? Please provide detailed explanation.

Solution. Yes. By definition,

$$\exists c, n_0 \in \mathbb{N} \text{ s.t. } \forall n \geq n_0, f(n) \leq 2^{cn^3}.$$

Take $n_0' = n_0, c' = 3c$. Then

$$\forall n \geq n_0', f(n)^3 \leq (2^{cn^3})^3 = 2^{3cn^3} = 2^{c'n^3},$$

i.e. $f(n)^3 = 2^{O(n^3)}$.

Problem 3 (20 pts). Consider the language $\{w\#w \mid w \in \{0,1\}^*\}$.

(a) (15 pts) Design a 2-tape TM with no more than 6 states (including q_{reject}) to recognize the language. Note that to simplify the figure, you don't need to draw q_{reject} and the transitions going to it. The strategy should be to

- copy the first w to the second tape
- move head of the second tape to the beginning
- compare contents in the two tapes

We require that $\Gamma = \{0, 1, \#, \sqcup\}$. Note that

- For the two-tape TM, we have

$$\delta : Q \times \Gamma^2 \to Q \times \Gamma^2 \times \{\text{L, R, S}\}^2$$

- There is no $\sqcup$ before the input

(b) (5 pts) Simulate

$$01\#01$$

by using the machine obtained in (a) and the standard TM in Example 3.9 of textbook (see Figure 1). Compare the number of steps needed by the two machines. You need to show every step of the simulation.

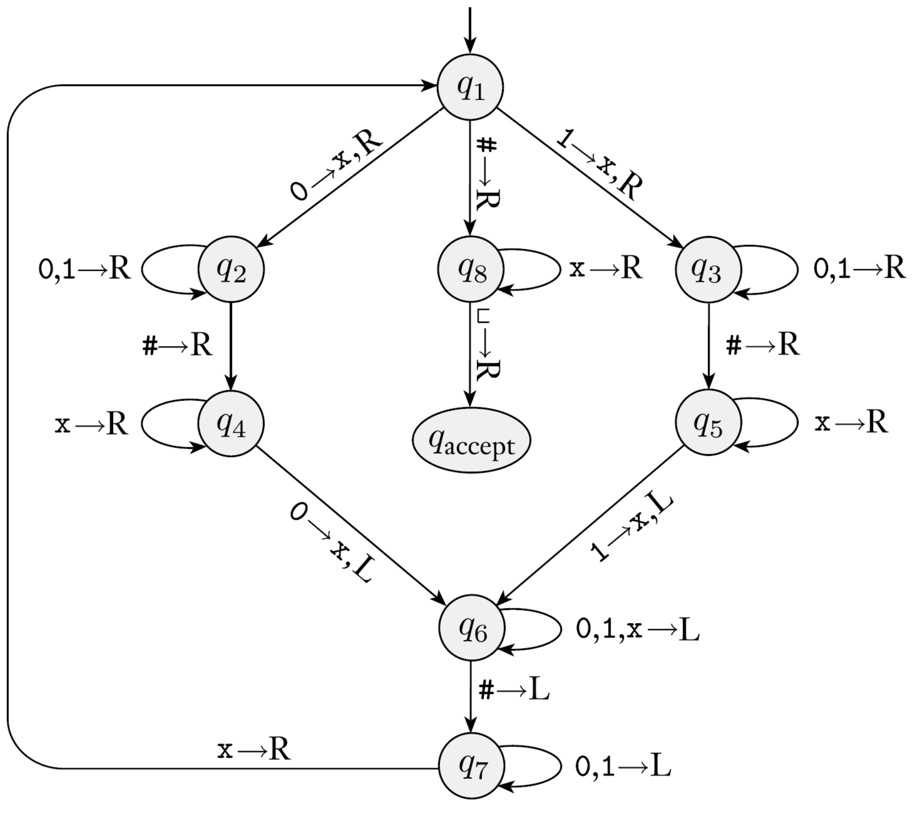
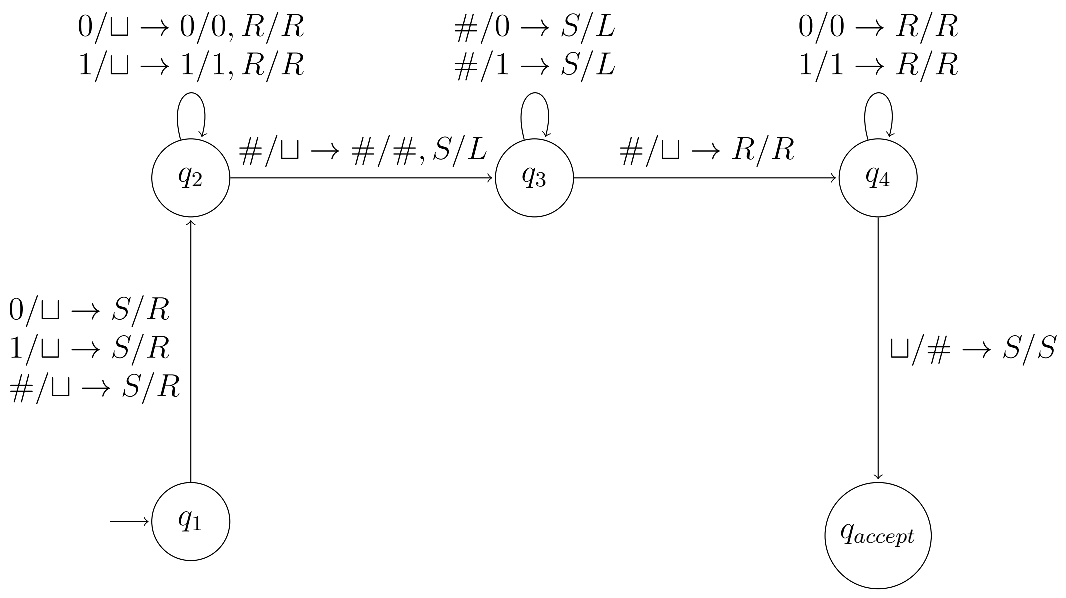


Figure 1: State diagram for the Turing machine in Example 3.9

Solution.

(a) We describe the TM with the following state diagram



The idea is described as follows.

- At q_1, we move the head of the second tape to the right for one step (the transition to q_2). By doing so, there will be a leading $\sqcup$ to indicate the beginning of the second tape.
- At q_2, we repeatedly copy the contents on the first tape to the second tape, and stop copying until the # is met (the transition to q_3).
- At q_3, we repeatedly move head of the second tape until reaching the beginning $\sqcup$ (the transition to q_4).
- At q_4, we repeatedly check if the right half of the input string on the first tape equals the copied left half on the second tape. We accept if the two heads meet the end of strings at the same time (the transition to q_{accept}).

Common mistake: If you have a configuration

$$q0\cdots$$

and a transition

$$\delta(q,0) = (0,L),$$

then the next configuration should still be

$$q0\cdots$$

rather than

$$q\sqcup 0\cdots.$$

(b) We give the sequence of running 01#01 on the two-tape TM as follows.

$$\text{Step } 0 \rightarrow q_1 \begin{array}{lllll} 0 & 1 & \# & 0 & 1 \\ \sqcup \end{array}$$

$$\text{Step } 1 \rightarrow \begin{array}{llllll} q_2 & 0 & 1 & \# & 0 & 1 \\ \sqcup & q_2 & \sqcup \end{array}$$

$$\text{Step } 2 \rightarrow \begin{array}{llllll} 0 & q_2 & 1 & \# & 0 & 1 \\ \sqcup & 0 & q_2 & \sqcup \end{array}$$

$$\text{Step } 3 \rightarrow \begin{array}{llllll} 0 & 1 & q_2 & \# & 0 & 1 \\ \sqcup & 0 & 1 & q_2 & \sqcup \end{array}$$

$$\text{Step } 4 \rightarrow \begin{array}{llllll} 0 & 1 & q_3 & \# & 0 & 1 \\ \sqcup & 0 & q_3 & 1 & \# \end{array}$$

$$\text{Step } 5 \rightarrow \begin{array}{llllll} 0 & 1 & q_3 & \# & 0 & 1 \\ \sqcup & q_3 & 0 & 1 & \# \end{array}$$

$$\text{Step } 6 \rightarrow \begin{array}{llllll} 0 & 1 & q_3 & \# & 0 & 1 \\ q_3 & \sqcup & 0 & 1 & \# \end{array}$$

$$\text{Step } 7 \rightarrow \begin{array}{llllll} 0 & 1 & \# & q_4 & 0 & 1 \\ \sqcup & q_4 & 0 & 1 & \# \end{array}$$

$$\text{Step } 8 \rightarrow \begin{array}{llllll} 0 & 1 & \# & 0 & q_4 & 1 \\ \sqcup & 0 & q_4 & 1 & \# \end{array}$$

$$\text{Step } 9 \rightarrow \begin{array}{lllllll} 0 & 1 & \# & 0 & 1 & q_4 & \sqcup \\ \sqcup & 0 & 1 & q_4 & \# \end{array}$$

$$\text{Step } 10 \rightarrow \begin{array}{lllllll} 0 & 1 & \# & 0 & 1 & q_{accept} & \sqcup \\ \sqcup & 0 & 1 & q_{accept} & \# \end{array}$$

The sequence of running on the Turing machine in example 3.9 is shown as follows.

$$\begin{array}{l}
q_1 01\#01 \\
\mathrm{x} q_2 1\#01 \\
\mathrm{x}1 q_2 \#01 \\
\mathrm{x}1\# q_4 01 \\
\mathrm{x}1 q_6 \#\mathrm{x}1 \\
\mathrm{x} q_7 1\#\mathrm{x}1 \\
q_7 \mathrm{x}1\#\mathrm{x}1 \\
\mathrm{x} q_1 1\#\mathrm{x}1 \\
\mathrm{xx} q_3 \#\mathrm{x}1 \\
\mathrm{xx}\# q_5 \mathrm{x}1 \\
\mathrm{xx}\#\mathrm{x} q_5 1 \\
\mathrm{xx}\# q_6 \mathrm{xx} \\
\mathrm{xx} q_6 \#\mathrm{xx} \\
\mathrm{x} q_7 \mathrm{x}\#\mathrm{xx} \\
\mathrm{xx} q_1 \#\mathrm{xx} \\
\mathrm{xx}\# q_8 \mathrm{xx} \\
\mathrm{xx}\#\mathrm{x} q_8 \mathrm{x} \\
\mathrm{xx}\#\mathrm{xx} q_8 \sqcup \\
\mathrm{xx}\#\mathrm{xx} \sqcup q_{\text{accept}}
\end{array}$$

The former takes 10 steps, while the latter takes 18 steps.

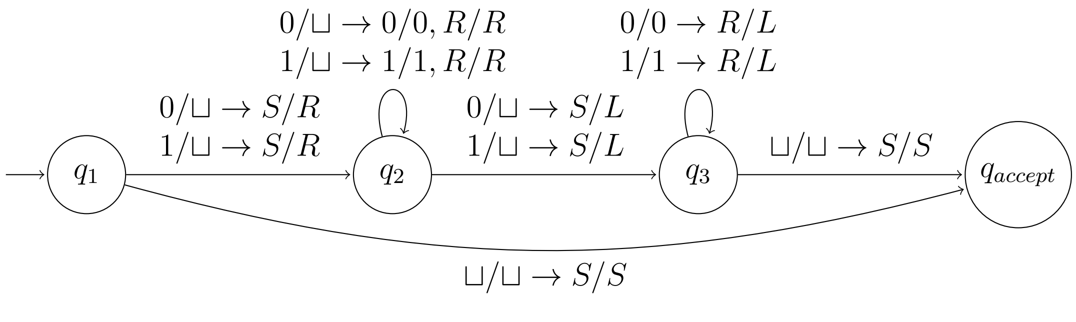
Problem 4 (20 pts). Consider the language $\{ww^R \mid w \in \{0,1\}^*\}$, where w^R is the reverse of w.

(a) (15 pts) Give a two-tape non-deterministic TM with no more than 5 states (including q_{reject}) to recognize this language. Note that here we allow S (stay moves) to make the problem easier.

(b) (5 pts) Simulate the path (not the entire tree) that leads to the acceptance of 0110.

Solution.

(a) See the following NTM.



The idea is described as follows.

- At q_1, we either accept ϵ (the transition to q_{accept}), or move the head of the second tape to the right (the transition to q_2).
- At q_2, we repeatedly copy the contents on the first tape to the second tape, and non-deterministically guess the middle of the input string and stop copying (the transition to q_3).
- At q_3, we repeatedly check if the right half of the input string on the first tape (reading left to right) equals the copied left half on the second tape (reading right to left). This is true if and only if the input is in the form ww^R. So we accept if the two heads meet $\sqcup$s at the same time (the transition to q_{accept}).

(b) See the following sequence.

$$\text{Step } 0 \rightarrow q_1 \begin{array}{llll} 0 & 1 & 1 & 0 \\ \sqcup & & & \end{array}$$

$$\text{Step } 1 \rightarrow \begin{array}{lllll} q_2 & 0 & 1 & 1 & 0 \\ \sqcup & q_2 & \sqcup & & \end{array}$$

$$\text{Step } 2 \rightarrow \begin{array}{lllll} 0 & q_2 & 1 & 1 & 0 \\ \sqcup & 0 & q_2 & \sqcup & \end{array}$$

$$\text{Step } 3 \rightarrow \begin{array}{lllll} 0 & 1 & q_2 & 1 & 0 \\ \sqcup & 0 & 1 & q_2 & \sqcup \end{array}$$

$$\text{Step } 4 \rightarrow \begin{array}{lllll} 0 & 1 & q_3 & 1 & 0 \\ \sqcup & 0 & q_3 & 1 & \sqcup \end{array}$$

$$\text{Step } 5 \rightarrow \begin{array}{lllll} 0 & 1 & 1 & q_3 & 0 \\ \sqcup & q_3 & 0 & 1 & \sqcup \end{array}$$

$$\text{Step } 6 \rightarrow \begin{array}{llllll} 0 & 1 & 1 & 0 & q_3 & \sqcup \\ q_3 & \sqcup & 0 & 1 & \sqcup & \end{array}$$

$$\text{Step } 7 \rightarrow \begin{array}{llllll} 0 & 1 & 1 & 0 & q_{\text{accept}} & \sqcup \\ q_{\text{accept}} & \sqcup & 0 & 1 & \sqcup & \end{array}$$

Common mistake: You cannot assume a $\sqcup$ before the input.

Problem 5 (30 pts). A CNF $\langle V, \Sigma, R, S\rangle$ satisfies that R contains only the following types of rules,

$$\begin{aligned} \text{Type-V} &: A \rightarrow BC \\ \text{Type-T} &: A \rightarrow a \\ \text{Type-E} &: S \rightarrow \epsilon \end{aligned}$$

where $a \in \Sigma$, $A, B, C \in V$, and $B, C \neq S$. Let

$$I_{CNF} = \{\langle G\rangle \mid G \text{ is a CNF and } |L(G)| = \infty\}$$

be the set of CNFs that generates an infinite number of strings, where $|L(G)|$ is the number of strings in the language $L(G)$ of G. Various ways may be possible to prove that I_{CNF} is decidable, but we would like you to answer the following sub-problems for a step-by-step proof.

(a) (5 pts) Consider a CNF $G = \langle V, \Sigma, R, S\rangle$. If now we consider only type-V rules, then the *partial parse trees* (not involving type-T and type-E rules) are binary trees where every node is a variable. For example, let

$$G_a = \langle \{S, A, B\}, \{a\}, R, S\rangle,$$

where R contains the following rules.

$$\begin{aligned} S &\to AB \mid \epsilon \\ A &\to BB \mid a \\ B &\to AA \end{aligned}$$

An example of partial parse trees is shown below.

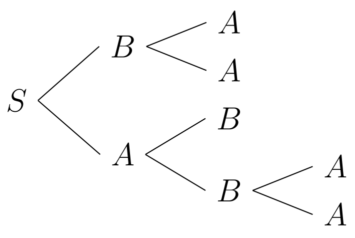
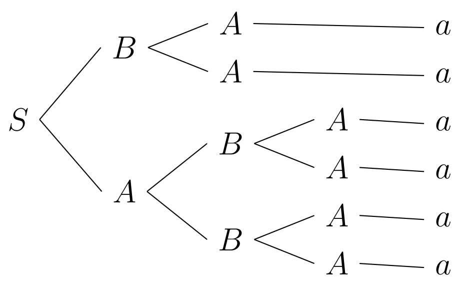


Figure 2: An example of partial parse tree

We say a partial parse tree *generates* a string w if it can be extended (with all rules in R) to a parse tree for w. For instance, the above partial parse tree can be extended to the following parse tree generating $aaaaaa$. (Note that we say "if it can be." That is, a partial parse tree may not be able to generate any string; see sub-problem (b).)



We define the *height* of a partial parse tree to be the maximum number of edges in paths starting from the root. Note that the height of the example in Figure 2 is 3.

By the similar reasoning of pumping lemma, if some non-S variable appears at least twice in a path from root, then the looping part between the two occurrences can be repeated as many times as we want, and therefore the partial parse tree can be extended to as large height as we want, and vice versa. For example, the partial parse tree shown above contains paths $S - A - B - A$ where A appears twice, and indeed $\langle G_a\rangle \in I_{CNF}$.

- Show that any partial parse tree contains an odd number of nodes.
- For any given CNF $\langle V, \Sigma, R, S\rangle$, find an odd value z so that for any of its partial parse tree, if

$$\#\text{nodes} \geq z,$$

then some non-S variable must appear at least twice in a path from the root of the partial parse tree.

Hint: You may directly use the fact that if the height of a binary tree is no more than h, then

$$\#\text{nodes} \leq 2^{h+1} - 1. \tag{1}$$

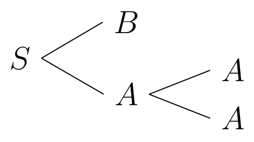
(b) (5 pts) We are about to argue that partial parse trees generate long strings if the height is large, but there is one subtle issue that needs consideration. Consider

$$G_b = \langle \{S, A, B\}, \{b\}, R, S \rangle,$$

where R contains the following rules.

$$\begin{aligned} &S \to AB \mid b \\ &A \to AA \\ &B \to b \end{aligned}$$

G_b can have a partial parse tree of height as large as we want (by repeatedly applying the rule $A \to AA$), but $L(G_b) = \{b\}$! This is because A is *non-deriving*, i.e. A cannot derive any string of literals. For example, the following partial parse tree cannot generate any string.



Therefore we would like to remove all the non-deriving variables before our argument. Assume we already have a decider D for $E_{CNF} = \{\langle G \rangle \mid G \text{ is a CNF and } L(G) = \emptyset\}$. Use D to design an algorithm that finds out all non-deriving variables of a given CNF.

(c) (5 pts) Given a CNF G, we can apply the algorithm in (b) to find out all the non-deriving variables, remove them from V, and remove all the rules involving any of them, getting a new CNF G'. It's clear that $L(G) = L(G')$. Because obtaining G' from G is a decidable procedure, we can now assume that the CNF G considered has no non-deriving variable.

Show that for a CNF without non-deriving variables, any partial parse tree of height $h > 0$ can generate at least one string, and that the strings w it generates satisfy $|w| \geq h + 1$.

(d) (10 pts) Wrapping up our previous results, given a CNF G without non-deriving variables (by (b)), if there are enough nodes in any partial parse tree of G, by (a) the partial parse tree can be extended to as large height as we want, and therefore by (c) it can generate strings as long as we want.

Use the above properties to derive an algorithm that decides I_{CNF}.

(e) (5 pts) Run your algorithm in (d) on $\langle G_a \rangle$ in (a) and $\langle G_b \rangle$ in (b). You don't need to provide details of running the algorithm in (b).

Solution.

(a) Because the smallest partial parse tree contains only the root S, and each type-V rule applied adds two nodes to the tree, the number of nodes is always odd.

From (1), if

$$\#\text{nodes} \geq 2^{n+1},$$

then

$$\text{height} \geq n+1.$$

That is, there is a path P from root that contains at least $n+2$ nodes. Take $n = |V| - 1$. The path now contains at least $|V| + 1$ nodes. Because S can only appear at the root, P contains at least $|V|$ non-S variables. There are only $|V| - 1$ non-S variables, so by pigeonhole principle some non-S variable must appear at least twice in the path P.

But $|V| > 0$ implies $2^{|V|}$ is even. Therefore we consider trees with at least $2^{|V|} + 1$ nodes.

(b) The following algorithm finds out all the non-deriving variables because A is non-deriving if and only if $G_A = \langle V, \Sigma, R, A\rangle$ cannot generate any string of literals, in which case G_A is accepted by D.

```
Input: CNF $\langle V, \Sigma, R, S\rangle$
Let $V_N = \emptyset$;
for $A \in V$ do
    Let $G_A = \langle V, \Sigma, R, A\rangle$;
    if $D$ accepts $\langle G_A\rangle$ then add $A$ to $V_N$;
end
return $V_N$;
```

Remark. The decider for E_{CNF} in the textbook actually finds out all non-deriving variables during the process. Thus instead of calling it $|V|$ times, in fact one call is enough.

(c) Because no variable is non-deriving, we can extend every variable at the leaves of the partial parse tree to a string of literals, and this gives a string generated by the partial parse tree.

Because a partial parse tree involves only type-V rules, applying one rule adds two children to the tree, which increases the number of leaf nodes (of the current tree) by one. A partial parse tree of height h contains a path of h edges, corresponding to h applications of rules. Because the height-0 tree with only the root S has one leaf, for the partial parse tree considered, the number of leaves is at least $h+1$. Since $h > 0$, by the definition of CNF, each leaf variable of the partial parse tree is not S, and each leaf variable cannot yield ϵ. So the leaf variables must derive non-empty strings of literals, and thus the generated string is of length at least $h+1$.

Common mistake: Some forget to answer the question that any partial parse tree of height $h > 0$ can generate at least one string.

(d) The algorithm works as follows.

```
Input: CNF $\langle V, \Sigma, R, S\rangle$
Remove non-deriving variables found by the algorithm in (b), getting CNF $\langle V', \Sigma, R', S'\rangle$;
for each partial parse tree $T$ with at most $2^{|V'|} + 1$ nodes do
    if the height of $T$ is at least $|V'|$ then accept;
end
reject;
```

The algorithm always halts because the number of partial parse trees with at most $2^{|V'|} + 1$ nodes is finite.

Common mistake: According to the problem statement, you need to use "the above properties" to derive the algorithm.

(e)

- $\langle G_a \rangle$
  There is no non-deriving variable. So $G' = G$, $V' = V$, and $|V'| = |V| = 3$. The partial parse tree provided in (a) has $9 = 2^3 + 1$ nodes, and its height is $3 = |V'|$, so $\langle G_a \rangle$ is accepted.
- $\langle G_b \rangle$
  A is non-deriving, so after removing A and rules involving A we obtain $G'_b = \langle \{S, B\}, \{b\}, R', S \rangle$, where R' contains the following rules.

  $$S \to b$$
  $$B \to b$$

  Now $|V'| = 2$, and the only partial parse tree with no more than $2^2 + 1 = 5$ nodes is the height-0 tree with only the root S. So $\langle G_b \rangle$ is rejected.