Theory of Computation '24
Tutorials

Notations. Let $\Sigma = \{a, b\}$. For $w \in \Sigma^\star$, let $|w|$ denote the length of w. Let $\#_a(w)$ denote the number of as in w and let $\#_b(w)$ denote the number of bs in w.

1. **Tutorial 2**

   Let L_1 be a regular language and L_2 be any language (not necessarily regular) over the same alphabet Σ. Prove that the language $L = \{x \in \Sigma^* \mid x \cdot y \in L_1 \text{ for some } y \in L_2\}$ is regular. Do this by mathematically defining a DFA for L starting from a DFA for L_1 and the language L_2.

2. Design an efficient algorithm that takes as input the descriptions of two DFAs, D_1 and D_2, and determines whether they recognize the same language.

3. Let $x = x[1] \dots x[n]$ and $y = y[1] \dots y[n]$ be strings of equal length over a common alphabet Σ. The strict shuffle of such strings is defined as $\text{sshuffle}(x, y) = x[1]y[1]x[2]y[2] \dots x[n]y[n]$. The strict shuffle of two languages $L_1, L_2 \subseteq \Sigma^*$ is defined as $\text{sshuffle}(L_1, L_2) = \{\text{sshuffle}(x, y) \mid x \in L_1,\, y \in L_2,\, |x| = |y|\}$. Given DFAs D_1, D_2 that recognise L_1, L_2 respectively, explain the construction of a DFA that recognises $\text{sshuffle}(L_1, L_2)$ (and thus, prove that `Reg` is closed under the strict shuffle operation).

4. Given a language L, $\text{Suf}(L)$ be the language defined as follows.

   $$\text{Suf}(L) = \{x \mid x \text{ is a suffix of some } y \in L\}.$$

   Suppose L is a regular language. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes L. Give mathematically rigorous and accurate definitions of an NFA that recognize the language $\text{Suf}(L)$, with a short proof of correctness.

Tutorial 3

5. Prove that the following languages are not regular.

   (a) $\{0^i 1^j \mid i > j\}$

   (b) $\{x \in \{0,1\}^* \mid x = \text{reverse}(x)\}$

   (c) $\{xyx \mid x, y \in \{0,1\}^* \text{ and } |x| > 0,\, |y| > 0\}$

(d) $\{x \in \{0,1\}^* \mid x$ is the binary representation of 3^{n^2}, without leading 0's, for some $n \in \mathbb{N}\}$

(e) $\{0^m 1^n \mid m \neq n\}$ (First try without pumping lemma and then as a challenge, construct a clean proof using the pumping lemma only.)

(f) $\{x \in \{0,1\}^* \mid x$ is the binary representation of $n!$, without leading 0's, for some $n \in \mathbb{N}\}$

6. If A is any language, let $\sqrt{A}$ denote the set of first halves of strings in A, where both the first and second halves are identical

$$\sqrt{A} = \{x \mid xx \in A\}$$

Show that if A is regular, then so is $\sqrt{A}$.

7. If A is any language, let $A_{\frac{1}{3}-\frac{1}{3}}$ denote the set of strings in A with the middle-third removed so that

$$A_{\frac{1}{3}-\frac{1}{3}} = \{xz \mid \text{for some } y,\ |x| = |y| = |z| \text{ and } xyz \in A\}$$

Show that even if A is regular, $A_{\frac{1}{3}-\frac{1}{3}}$ is not necessarily regular.

Tutorial 4

8. If a language $\mathcal{A}$ is regular, it is said to satisfy Pumping Lemma.

(a) Show that the converse of this statement does not hold, i.e. give an example of a language $\mathcal{L}$ such that it satisfies Pumping Lemma but **is not regular**

(b) Prove that your chosen $\mathcal{L}$ is not regular through other properties

(c) Given any irregular language $\mathcal{L}$ on alphabet Σ, show that it can be modified to form an irregular language $\mathcal{L}'$ on alphabet $\Sigma \cup \{\#\}$, that satisfies Pumping Lemma

9. Prove that for any infinite regular language L, there exist two infinite regular languages L_1, L_2 such that $L = L_1 \cup L_2$ and $L_1 \cap L_2 = \emptyset$. Here are two hints.

(a) Let D be any DFA and q be any one of its states. Prove that the language

$$L_q = \{x \mid x \in \mathcal{L}(D) \text{ and the run of } D \text{ on } x \text{ visits } q \text{ an odd number of times}\}$$

is regular.

(b) Recall the proof of the pumping lemma.

10. Prove using the Myhill-Nerode Theorem that the following languages are non-regular:

(a) $\{x \in \{a,b,c\}^* \mid \#_{ab}(x) = \#_{bc}(x)\}$

(b) $\{0^p \mid p \text{ is prime}\}$

(c) $\{w \in \{(,)\}^* \mid w$ is a balanced string of parantheses$\}$

(d) $\{xy \in \{0,1\}^*\{a,b\}^* \mid x$ is a binary encoding with no leading 0's of $|y|\}$

11. We revisit the problem of showing that the languages

$$L_1 = \{x \in \{0,1\}^\star | x \text{ is a binary representation of } 3^{n^2}\}$$

$$L_2 = \{x \in \{0,1\}^\star | x \text{ is a binary representation of } n!\}$$

are non-regular. We do this in two steps

(a) Show that, if A is an *infinite* regular language, then the set of *lengths* of strings in A contains an infinite Arithmetic Progression.

(b) Use the above to prove that the two given languages are non-regular

Tutorial 5

12. Write context-free grammars for the following languages:

(a) the set of words over the alphabet $\{a, b\}$ containing the same number of occurrences of a and b. Prove rigorously using induction on an appropriate quantity the correctness of your construction. Is your grammar unambiguous?

(b) the set of words over the alphabet $\{a, b\}$ containing twice as many occurrences of a as occurrences of b

(c) the set of words over the alphabet $\{a, b\}$ of even length where the number of occurrences of b in even positions is the same as the number of occurrences of b in odd positions.

13. Write context-free grammars for the following languages

(a) $\{a^i b^j c^k : i \neq j \text{ or } j \neq k\}$

(b) $\{a^i b^j c^k : i + k = j\}$

14. Prove that CFGs are closed under union and concatenation. In other words, given two CFGs generating languages L, K, give CFGs to generate $L \cup K$ and LK.

15. Give algorithmic descriptions of PDAs recognizing the following languages

(a) The set of strings over the alphabet $\{a, b\}$ with more a's than b's

(b) $\{w\#x \mid w^{\mathcal{R}} \text{ is a substring of } x, \text{ where } w, x \in \{0,1\}^\star\}$

(c) $\{x_1\#x_2\#\cdots\#x_k \mid k \geq 1, \text{ each } x_i \in \{a,b\}^\star, \text{ and for some } i, j,\ x_i = x_j^{\mathcal{R}}\}$

Tutorial 6

16. Design a Turing Machine to accept the language $\{a^n b^n c^n | n \geq 0\}$

- **Solution Level 1:** There are two things to keep in mind while designing a Turing Machine for this language:
(a) The number of a's, b's, and c's should match, and
(b) All b's come after all a's, and all c's come after all b's.
If the string violates either of these two conditions, we reject the string. Otherwise we accept the string.

- **Solution Level 2:** To this end, we can do the following:

  (a) Initialize 3 counters (placed after the input string ends) and set their values to 0 - one for each symbol.
  (b) Start reading the string from the leftmost input symbol. If the string is not formatted correctly (all c's follow all b's and all b's follow all a's), reject the string.
  (c) Read the first symbol and increment by one the counter corresponding to a. Move right (and keep incrementing the counter corresponding to a by one for each a symbol encountered subsequently) until we read a b symbol.
  (d) Once we encounter the first b, increment the counter by one corresponding to b. Move right (and keep incrementing the counter corresponding to b by one for each b symbol encountered subsequently) until we encounter the first c symbol.
  (e) Once we encounter the first c, increment the counter corresponding to c by one. Move right (and keep incrementing the counter corresponding to c by one for each c symbol encountered subsequently) until the string ends.
  (f) Compare the three counters and reject if the values of all the counters are not equal. Otherwise accept.

  We can add a little bit of technical detail to this. The counters are tricky to implement in an actual Turing Machine, because we usually assume that the Tape alphabet is unary. Consider the following Tape alphabet: $\Gamma = \{\$, \sqcup, \alpha, \beta, \gamma\}$.

  (a) Every time we encounter an a, we move past the $\sqcup$ symbol and write an α symbol to the tape. Then we move back to one position right of the last recorded position.
  (b) Every time we encounter a b symbol, we move past the $\sqcup$ symbol and all the α symbols, and add **two** β symbols to the tape. Then we move back to one position right of the last recorded position.
  (c) Every time we encounter a c symbol, we move past the $\sqcup$ symbol and all the α and β symbols, and add one γ symbol to the tape. Then we move back to one position right of the last recorded position.
  (d) Once we have finished reading the input, we start scratching off one β symbol every time we read one α symbol. Once all the α symbols (and an equivalent number of β symbols) are scratched off, we start scratching off one γ symbol every time we read one β symbol.
  (e) If the string is of the form $\{a^i b^j c^k | i \neq j \neq k\}$, then we will encounter one of the following error scenarios:
     - There are α symbols left to scratch but no more β symbols.
     - There are β symbols left to scratch but no more γ symbols.
     - All α and β symbols are scratched, but there are γ symbols left to scratch.

     We reject the input if any of these scenarios occur. Otherwise we accept (given that the string follows the format of the input).

17. Describe a Turing Machine to recognize the language $\{ww \mid w \in \{0,1\}^*\}$

    Solution: Given a string $x = x_1 x_2 \cdots x_\ell \in \{0,1\}^*$, $x_i \in \{0,1\}\ \forall\ 1 \leq i \leq \ell$ for some $\ell \in \mathbb{N}$ to check whether it is of the form ww for some $w \in \{0,1\}^*$, we are done if and only if all the following conditions hold:

(a) $x_i \in \{0,1\} \; \forall \; 1 \le i \le \ell$

(b) $|x|$ is even

(c) $x_i = x_{\frac{\ell}{2}+i} \; \forall \; 1 \le i \le \frac{\ell}{2}$

- **High-Level Description:**
  Based on this, the following simple algorithm suffices:

Algorithm Pseudocode to recognize $\{ww \mid w \in \{0,1\}^*\}$

```
Input: $x \in \{0,1\}^*$
Output: ACCEPT or REJECT
1: if $|x|$ is odd or contains a character other than 0 and 1 then REJECT
2: Locate the midpoint position $k$ of $x$ ($k = \frac{\ell}{2}$)
3: while $1 \le i \le \frac{\ell}{2}$ do
4:     if $x_i \ne x_{k+i}$ then REJECT
5: ACCEPT
```

- **Implementation-Level Description:**
  When we make the jump to implementation-level, we have to convert every step of the pseudocode into a subroutine that implements the functionality of that step **in terms of a tape of unbounded length and markings on said tape**:

  Step 1 can be directly implemented as a subroutine, since the language of even length strings in $\{0,1\}^*$ is a regular language $(\{0,1\} \cdot \{0,1\})^*$

  In **Step 2**, i.e. to locate the midpoint of x, we need to reach a position k in x such that it is equidistant from both ends of x. If we were to alternatively scratch off the leftmost and rightmost symbol of x and replace with #, we are guaranteed to reach the midpoint position k after multiple rounds (since $|x|$ is even).

  But in the process of finding the midpoint position, we ended up scratching the entire string x and we now have nothing to compare. So rather than replacing each 0 and 1 with #, how about replacing 0 with α and 1 with β respectively? It works out since the original string is now preserved albeit in a different alphabet and we can verify that we have reached the midpoint position when there are no more 0s and 1s in the tape.

  Looping as described in **Step 3** will require us to switch between i^{th} position and $(k+i)^{\text{th}}$ position for some $1 \le i \le \frac{\ell}{2}$ but the problem our finite control will forget the value of k, the moment we move even a single step away from the current position.

  To solve this, we will just move our tape head from the k^{th} position to the beginning and reverse the marking we had done initially, replace each α with 0 and β with 1. We now are at the starting position with a modified input $x = cv$, where $c \in \{0,1\}^*$ and $v \in \{\alpha,\beta\}^*$, so a TM can easily see where the first half of x ends and the second half begins.

  Note: Alternatively, we can directly start the comparison step in **Step 4** but read from the second half first and then verifying characters in the first half iteratively to get a shorter solution. Although my solution is not that efficient, it contains a lot of sanity checks since **the goal is not to give a fast or efficient algorithm (unlike DSA, ADA and/or MAD) but to give a logically valid algorithm, without any kind of loopholes**

To perform **Step 4**, we can just scratch out the leftmost character in c with # and if it was 0, we go over to the leftmost character in v and read it. If it is β, directly reject directly or scratch with \$. Since the markers are different, there is no issue in repeating this process as long as the entire string gets scratched out with # and \$ (since $|c| = |v|$).

Finally, **Step 5** says that if we have scratched out the entire string, we simply accept and since we have explicitly passed all the necessary and sufficient conditions to verify whether $x \in \{ww \mid w \in \{0,1\}^*\}$

Algorithm Turing Machine to recognize $\{ww \mid w \in \{0,1\}^*\}$

Input: Infinite tape containing $x \in \{0,1\}^*$ along with a tape head
Output: ACCEPT or REJECT

```
1: if $x \notin (\{0,1\} \cdot \{0,1\})^*$ then REJECT
2: while the universe exists (always true) do
3:     if tape does not contain any more 0 s and 1 s then exit from loop
4:     if 0 is read then write $\alpha$
5:     else write $\beta$
6:     Move to the rightmost occurrence of a 0 or 1
7:     if 0 is read then write $\alpha$
8:     else write $\beta$
9:     Move to the leftmost occurrence of a 0 or 1
10: while moving left to the beginning of tape do
11:     if $\alpha$ is read then write 0
12:     else write 1
13: while the universe exists (always true) do
14:     if tape does not contain any character from $\{0, 1, \alpha, \beta\}$ then exit from loop
15:     if 0 is read then write #
16:         Move to the leftmost occurrence of $\alpha$ or $\beta$
17:         if $\beta$ is read then REJECT
18:         else write $
19:     else write #
20:         Move to the leftmost occurrence of $\alpha$ or $\beta$
21:         if $\alpha$ is read then REJECT
22:         else write $
23:     Move to the leftmost occurrence of a 0 or 1
24: ACCEPT, regardless of configuration
```

Assumption: Since the leftmost occurrence and rightmost occurrence of any character is unique and can be easily found, we just assume that any subroutine that scans the whole tape as stated in **Step 1**, **Step 3** and **Step 14** will start and end with the tape head at the same position.

I also assume that there is a blank space character on both the left and right sides of the input, because we can always copy the input again leaving one space gap. The exact details is an exercise to the reader.

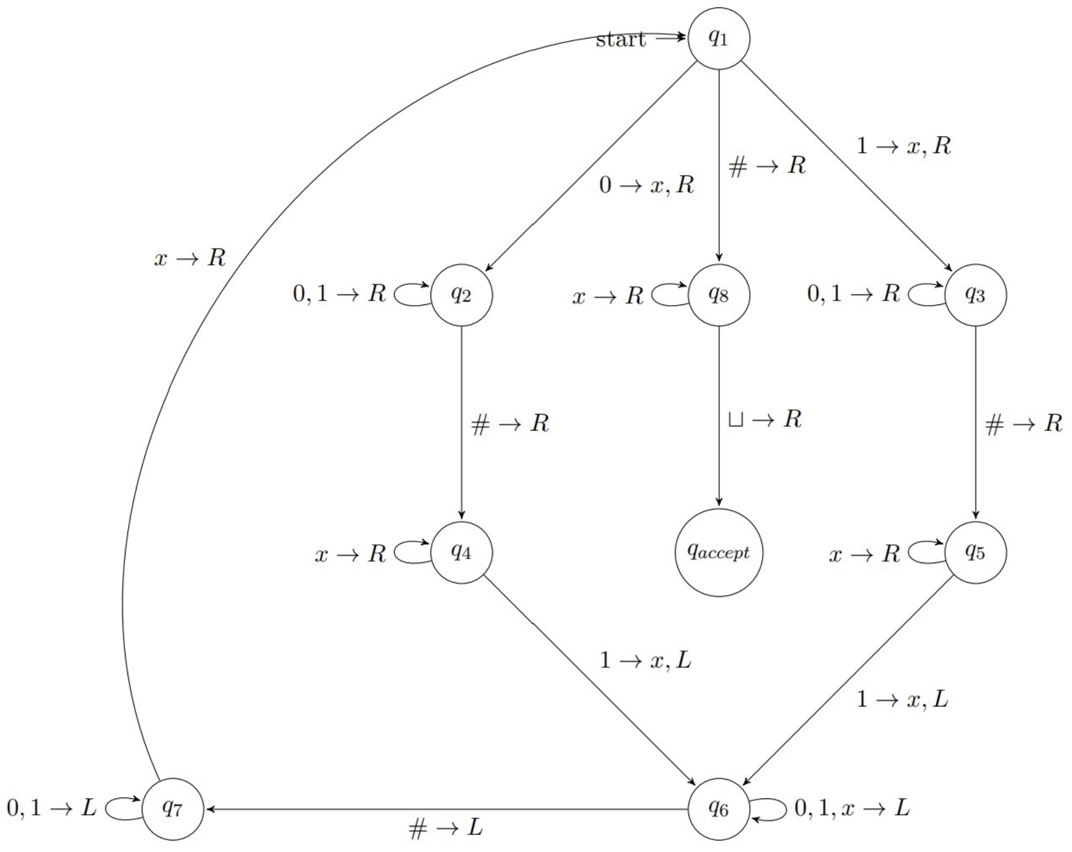
Tutorial 7

18. On the following page is the state diagram of a Turing machine using input alphabet $\Sigma = \{0, 1, \#\}$ and tape alphabet $\Gamma = \{0, 1, \#, x, \sqcup\}$. The notation "$a \to R$" is shorthand for "$a \to a, R$" The reject state and transitions to the reject state are not shown. Whenever the TM encounters a character for which there is no explicit transition that means that the TM goes to the reject state. Use the convention that the head moves right in each of these transitions to the reject state.

    (a) Give the sequences of configurations that this TM M enters when given as input strings (i) 01#11, (ii) 01#10, and (iii) 0##0.

    (b) Give an implementation-level description of the Turing machine described by this state diagram.

    (c) What is the language decided by M? That is, what is the set of strings that lead the TM M to halt and accept?



19. Prove the any finite state automaton with **two stacks** is equivalent to a Turing machine

20. Given two deterministic Turing Machines for decidable languages L_1 and L_2, construct a **deterministic** TM decider for

    (a) $L_1 \cdot L_2$

    (b) $\{L_1\}^\star$

21. A Turing machine with stay put instead of left is similar to an ordinary Turing machine, but the transition function has the form

$$\delta : Q \times T \to Q \times T \times \{R, S\}$$

At each point the machine can move its head right or let it stay in the same position. Show that this Turing machine variant is not equivalent to the usual version. (Hint: Show that these machines only recognize regular languages).