# Theory Assignment II

Automata Theory Monsoon 2024, IIIT Hyderabad

October 1, 2024

Total Marks: 40 points Due date: **16/09/24 11:59 pm**

**General Instructions:** All symbols have the usual meanings (example: $\mathbb{R}$ is the set of reals, $\mathbb{N}$ the set of natural numbers, and so on). FSM stands for finite state machine. DFA stands for deterministic finite automata. NFA stands for non-deterministic finite automata. $a^*$ is the Kleene Star operation.

---

1. [3 points] We will show that $D$ is context-free by constructing a context-free grammar for it.

   First, let's analyze the structure of strings in $D$:

   - Any string $z \in D$ must be of even length, with its two halves differing in at least one bit.
   - This means $z$ can be written as $z = t0yv1w$ or $z = t1yv0w$ where $|t| = |v|$ and $|y| = |w|$.
   - Equivalently, we can say $z = t0vy1w$ or $z = t1vy0w$ where $|t| = |v|$ and $|y| = |w|$.

   Given this structure, we can construct the following context-free grammar for $D$:

   $$\begin{aligned} S &\to S0S1 \mid S1S0 \\ S0 &\to XS0X \mid 0 \\ S1 &\to XS1X \mid 1 \\ X &\to 0 \mid 1 \end{aligned}$$

   This grammar generates strings in $D$ as follows:

   - The start symbol $S$ ensures that there's at least one pair of different bits at corresponding positions in the two halves of the string.
   - $S0$ and $S1$ generate matching substrings with a 0 or 1 in the middle, respectively.
   - $X$ can generate any bit, allowing for matching substrings on either side of the center bit in $S0$ and $S1$.

   This grammar generates all strings in $D$ and only strings in $D$. Therefore, $D$ is a context-free language.

2. [6 points] 1. Assume, for the sake of contradiction, that $L = \{a^n b^m \mid m = n^2\}$ is context-free. Let $p$ be the pumping length given by the pumping lemma for context-free languages. Consider the string $s = a^p b^{p^2}$, which is clearly in $L$. According to the pumping lemma, $s$ can be split into $s = uvxyz$ such that:

1. $uv^ixy^iz \in L$ for all $i \geq 0$, 2. $|vxy| \leq p$, 3. $|vy| > 0$.

Since $|vxy| \leq p$ and $|v| + |y| > 0$, $v$ and $y$ can only consist of $a$'s or $b$'s or a mix of both. We will consider the possible cases for the substrings $v$ and $y$:

1. $v$ and $y$ consist only of $a$'s: - In this case, $v$ and $y$ are substrings of $a^p$. Let $v = a^j$ and $y = a^k$ with $j + k > 0$. - Pumping $s$ would result in $uv^2xy^2z = a^{p+j+k}b^{p^2}$, which is not in $L$ since the number of $b$'s is still $p^2$, not $(p + j + k)^2$.

2. $v$ and $y$ consist only of $b$'s: - In this case, $v$ and $y$ are substrings of $b^{p^2}$. Let $v = b^j$ and $y = b^k$ with $j + k > 0$. - Pumping $s$ would result in $uv^2xy^2z = a^pb^{p^2+j+k}$, which is not in $L$ since the number of $b$'s is no longer $(p + j)^2$.

3. $v$ contains $a$'s and $y$ contains $b$'s: - In this case, let $v = a^j$ and $y = b^k$ with $j, k > 0$. - Pumping $s$ would result in $uv^2xy^2z = a^{p+j}b^{p^2+k}$. For this string to be in $L$, we would require $((p + j)^2 = p^2 + k) \implies j^2 + 2p \cdot j = k$. Now, $2p \cdot j > 2p > p \implies j^2 + 2p \cdot j > p \implies k > p$. But clearly $k < p$ cause $|vxy| \leq p$ leading to a contradiction.

In all cases, pumping $s$ results in strings that are not in $L$. Hence, we have reached a contradiction, implying that $L = \{a^nb^m \mid m = n^2\}$ is not context-free.

2. Assume that $L$ is context-free. Then by the pumping lemma for context-free languages, there must be a pumping length $p$ such that if $s$ is a string in the language with $|s| \geq p$, then $s$ satisfies the conditions of the pumping lemma.

   Let $s = 0^p1^p0^p1^p$. Clearly $|s| \geq p$, as required by the pumping lemma. Now, according to the pumping lemma, $s = uvxyz$ with $|vxy| \leq p$. This means there are three cases that describe $vxy$:

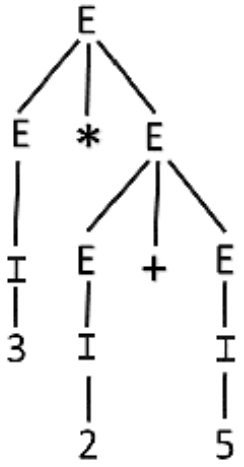
   (a) $vxy$ is comprised of all 0s and is contained entirely within either the first or second string of 0s. Since $|vy| > 0$, then either $v$ or $y$ must contain at least one 0. Now consider $uv^0xy^0z$. This forces either the first or the second string of 0s to have at least one fewer 0s than the other. Thus $uv^0xy^0z \notin L$, which is a contradiction of the pumping lemma.

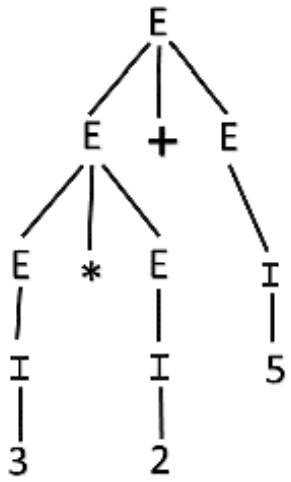
   (b) $vxy$ is comprised of all 1s and is contained entirely within either the first or the second string of 1s. By the same reasoning in 1, we can see that a contradiction is derived.

   (c) $vxy$ is comprised of a mix of 0s and 1s. This really describes two cases, where $vxy$ is a string of 0s followed by a string of 1s or $vxy$ is a string of 1s followed by a string of 0s. We take the first case to be representative. In this case $vxy$ either straddles the first 0-1 division or it straddles the second 0-1 division. Again, because $|vxy| \leq p$, it follows that pumping either up or down will only affect the substrings immediately adjacent to the division that is straddled. The other two substrings will be unaffected. Thus the length of the straddled substrings will be changed by pumping while the length of the other two will not be. Thus the result of pumping will result in a string that is not in the language, and a contradiction is again derived.

   Since for every case, $s$ cannot be pumped, we have a contradiction with the pumping lemma. Therefore our original assumption was false and $L$ is not context-free.

3. [2 points]




4. [4 points] 1. First add a new start symbol $S_0$ and the rule $S_0 \to A$:

$$\begin{aligned} S_0 &\to A \\ A &\to BAB \mid B \mid \epsilon \\ B &\to 00 \mid \epsilon \end{aligned}$$

2. Next eliminate the $\epsilon$-rule $B \to \epsilon$, resulting in new rules corresponding to $A \to BAB$:

$$\begin{aligned} S_0 &\to A \\ A &\to BAB \mid BA \mid AB \mid A \mid B \mid \epsilon \\ B &\to 00 \end{aligned}$$

3. Now eliminate the redundant rule $A \to A$ and the $\epsilon$-rule $A \to \epsilon$:

$$\begin{aligned} S_0 &\to A \mid \epsilon \\ A &\to BAB \mid BA \mid AB \mid BB \mid B \\ B &\to 00 \end{aligned}$$

4. Now remove the unit rule $A \to B$:

$$\begin{aligned} S_0 &\to A \mid \epsilon \\ A &\to BAB \mid BA \mid AB \mid BB \mid 00 \\ B &\to 00 \end{aligned}$$

5. Then remove the unit rule $S_0 \to A$:

$$\begin{aligned} S_0 &\to BAB \mid BA \mid AB \mid BB \mid 00 \mid \epsilon \\ A &\to BAB \mid BA \mid AB \mid BB \mid 00 \\ B &\to 00 \end{aligned}$$

6. Finally convert the 00 and $BAB$ rules:

$$\begin{aligned} S_0 &\to BA_1 \mid BA \mid AB \mid BB \mid N_0N_0 \mid \epsilon \\ A &\to BA_1 \mid BA \mid AB \mid BB \mid N_0N_0 \\ A_1 &\to AB \\ B &\to N_0N_0 \\ N_0 &\to 0 \end{aligned}$$

This grammar satisfies all the requirements for Chomsky Normal Form.

5. [5 points] PDA equipped with two stacks is computationally more powerful than a PDA with a single stack.
Proof: To demonstrate that a Pushdown Automaton (PDA) with two stacks ($P^2$) is computationally more powerful than a PDA with a single stack ($P^1$), we first note that any language recognizable by $P^1$ can also be recognized by $P^2$. This is because a $P^2$ can simulate a $P^1$ by using one of its stacks as a replacement for the single stack of the $P^1$.

To show that $P^2$ is strictly more powerful than $P^1$, we need to identify a language $L$ that can be recognized by $P^2$ but not by $P^1$. Since the class of Context-Free Languages (CFLs) is exactly the set of languages recognizable by $P^1$, we need to find a language $L$ that is not context-free but can be recognized by $P^2$.

Consider the language

$$L = \{w\#w \mid w \in \{0,1\}^*\}.$$

It is known from the pumping lemma for context-free languages that $L$ is not context-free. We will now construct a $P^2$ that recognizes $L$.

**Construction of $P^2$ for recognizing $L$:**

Let $P_1^2$ be a PDA with two stacks $S_1$ and $S_2$. The operation of $P_1^2$ on an input string $x$ is as follows:

(i) Push each symbol read from the input into stack $S_1$ until the symbol '#' is encountered.

(ii) After reading '#', switch to stack $S_2$ and push the remaining symbols from the input into $S_2$ until the end of the input is reached.

(iii) Simultaneously pop the top symbols from $S_1$ and $S_2$. If at any point the symbols $e_1$ from $S_1$ and $e_2$ from $S_2$ are not equal, *reject* the input.

(iv) Continue the matching process until both stacks $S_1$ and $S_2$ are empty. If both stacks reach their bottom simultaneously, *accept* the input. If either stack is not empty when the other is empty, *reject* the input.

This construction shows that $P_1^2$ can recognize the language $L$, which is not context-free and hence not recognizable by a single-stack PDA ($P^1$). Therefore, we have demonstrated that $P^2$ is more powerful than $P^1$ in terms of the languages they can recognize.

Further it can be proved that "a PDA with two stacks is equivalent in power to a Turing machine".

6. [6 points] We will use a 3-tape TM. The intuition is to first check whether the input $x$ is even (the last digit is 0) or odd (the last digit is 1). If the former, we replace the terminating 0 with a blank (dividing by 2 in binary) and halt. If the latter, we duplicate the input on a second tape and add a terminating 0 to the original (multiplying by 2 in binary). We now have $2x$ on the first tape and $x$

on the second tape. We then add a leading zero to the first tape and two leading zeroes to the second so that we can sum the newly aligned tapes to get $3x$, to which we finally add 1. The third tape is used to keep track of carry bits in the addition step.

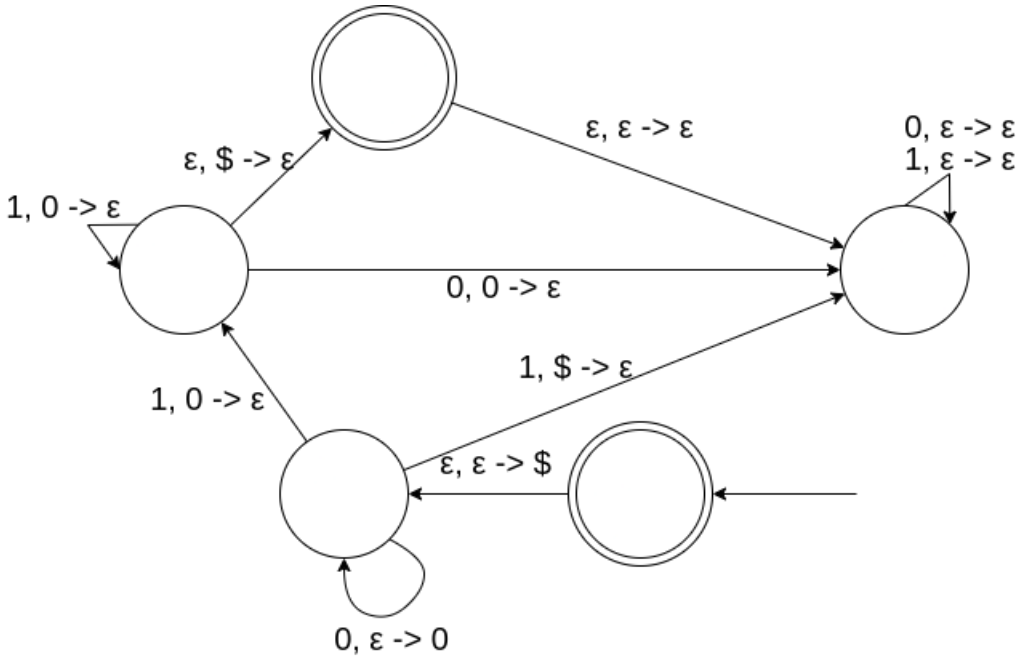
More specifically, construct machine $M$ as follows: On input $\#\langle x \rangle$, where $\langle x \rangle$ is a binary string:

1. Make sure the first character is a #, then scan the remaining binary number on the input tape from left to right until the last digit is reached. If it is a 0 (i.e., the number is even), replace the 0 with a blank and halt. (If it is a 1, go on.)
2. Write "#00" to the second tape, then copy the contents of the first tape onto the second tape after the '#00".
3. Shift the contents of the first tape after the # over to the right by 1. Make the new first symbol after the # on the first tape a "0".
4. Replace the first blank on the first tape with a 0.
5. Write a # to the third tape for each symbol on the first tape, then replace the rightmost # with a 0.
6. Add the contents of the second tape to the first tape by repeating the following two steps until all the digits of the second tape are crossed off:
   (a) Starting with the rightmost digit on the second tape that has not been crossed off, add it and the digits from the other two tapes in the corresponding place (i.e. the same number of spaces to the left of the first blank on their tape), write the result to that place on the first tape, and write any carry to the third tape:
      i. If all of the three digits are 0, write 0 to that place on the first tape and replace the rightmost # on the third tape with a 0.
      ii. If only one of them is 1, write 1 to that place on the first tape and replace the rightmost # on the third tape with a 0.
      iii. If only two of them are 1, write 0 to that place on the first tape and replace the rightmost # on the third tape with a 1.
      iv. If all three of them are 1, write 1 to that place on the first tape and replace the rightmost # on the third tape with a 1.
   (b) Cross off the rightmost digit on the second tape that has not been crossed off.
7. Add 1 to the number on the first tape by starting with the rightmost digit and checking if it is 0. If it is 0, change it to 1 and go to the final step. If it is 1, change it to 0, proceed to the next rightmost digit and repeat this step.
8. If the first symbol after the # on the first tape is a 0, delete it and shift the remainder of the tape one to the left. Halt.

7. [4 points] Let $R_1$ and $R_2$ be two recursive languages, and $RE_1$ and $RE_2$ be two recursively enumerable languages. The classes the following languages belong to are:

   1. $R_1 \cup R_2$ is **Recursive**.
   2. $RE_1 \cup RE_2$ is **Recursively Enumerable**.
   3. $R_1 \cup RE_2$ is **Recursively Enumerable**.
   4. $R_1 \cap RE_2$ is **Recursively Enumerable**.

8. [3 points] Start with two strips - Input and Working
From the input strip, keep moving right and adding the character to the working strip. After each addition, call $M_1$ on it. Once $M_1$ recognises the input, insert a character after the last character read from the Input strip (ensure the character does not belong to either language) and clear the working strip.
Do the same routine from the current position on Input strip, while calling $M_2$ and then mark its end once $M_2$ recognises an input.
Now go back to the start of the Input strip and use the marking characters to identify $s_1$ and $s_2$ separately and add them to the Working strip.




9. [3 points]

10. [4 points] Consider the context-free grammar $C = (V, \Sigma, R, S)$ where:

- $V = \{A, B\}$ (the set of variables),
- $\Sigma = \{0, 1\}$ (the set of terminals),
- $S = A$ (the start symbol),
- $R$ is the set of production rules, given by:

$$A \to 0A0 \mid 0B1 \mid 1A0 \mid 1B1$$
$$B \to 0B0 \mid 0B1 \mid 1B0 \mid 1B1 \mid 0 \mid 1 \mid \epsilon$$

The way we construct strings in this grammar is by always adding either a 0 or a 1 to either side, so we keep track of the midway point of our string. We can terminate if and only if we derive the $B$ variable somewhere in our derivation.

We derive the $B$ variable somewhere in our derivation if and only if we add a 1 to the second half of our string. We can then continue to build our string however we want and end up picking either a 0, 1, or $\epsilon$ as our middle symbol