CS 39: Theory of Computation **Winter '23**

PSET 5 — 02/20/2023

Prof. Chakrabarti ***Student: Amittai Siavava***

Credit Statement

I discussed ideas for this homework assignment with Paul Shin.

I also referred to the following books:

(a) **Introduction to the Theory of Computation** by **Michael Sipser**.

(b) **A Mathematical Introduction to Logic** by **Herbert Enderton**.

Problem 1.

For each of the following languages, say whether or not it is a CFL and prove your answer, either by designing an appropriate CFG or PDA or by using closure properties and/or the pumping lemma. If designing a CFG/PDA, please explain your construction in brief so the grader can understand your design.

Lemma 1.1. *(Pumping Lemma for CFLs)*

If a language is a CFL, then the language has a pumping length p such that any string s in the language that has length at least *p can be written as $s = uvwxy$ where*

(i) $|vwx| \leq p$.

(ii) $|vx| > 0$.

(iii) $uv^k wx^k y \in L_1$ *for all* $k \geq 0$.

(a) $L_1 = \{a^n b^n c^m : n \leq m \leq 2n\}$

The language is not a CFL.

Suppose it is, then take p to be the pumping length and $s = a^p b^p c^{2p}$, then s is a string in the language.

Since $|vwx| \leq p$, vwx must not span three distinct letters. In particular, it must be of one of the following forms:

(i) $vwx = a^i b^j$ for some $i, j \geq 0, 1 < i + j \leq p$.

Since $|vx| > 0$, at least one of v and x must be nonempty, so vx contains some number of a's and some number of b's (but no c's). There are two scenarios here.

- If vx contains the same number of a's as the number of b's, then pumping down the string does not break the equality of a's and b's, but ir reduces the number of a's and the number of b's by some non-zero l_1 and l_2, respectively. So $a^{p-l_1} b^{p-l_2} c^2 p \in L_1$, suggesting that $2p \leq 2p - (l_1 + l_2)$, which can only happen since $l_1 + l_2 > 0$ (following from $|vx| > 0$).
- If vx contains more a's than b's, then pumping down the string breaks the equality of a's and b's the resulting string cannot be in L_1. The same scenario applies for when vx contains more b's than a's.

(ii) $vwx = b^i c^j$ for some $i, j \geq 0, 1 < i + j \leq p$.

If $i > 0$ (so vwx starts with a sequence of b's).

- If v is nonempty, then v contains a number of b's so pumping down the string reduces the number of b's but does not reduce the number of a's, breaking the equality of the number of a's and the number of b's in the string. The resulting string must not be in L_1, contradicting the pumping lemma.
- If v is empty, then x must be nonempty (since $|vx| > 0$). In particular, x must contain a number of c's. It may also contain a number of b's. If x contains only c's (say, $x = c^l$ for some $0 < l \leq j$), pumping up the string tells us that $a^p b^p c^{2p+l}$ is in L_1, but that implies that $2p + l \leq 2p$, so $l = 0$, but if $l = |x| = 0$ and $|v| = 0$ then we would have that $|vx| = 0$, contradicting the pumping lemma. If x contains both c's and b's, the pumping up will increase the number of b's in the string, breaking the equality between the number of b's and the number of a's. So the resulting string must not be in L_1, contradicting the pumping lemma.

If $i = 0$, then $j > 0$ and, since $|vx| > 0$, $vx = c^l$ for some $0 < l < j$. Pumping up the string increases the number of c's in the string by some amount l. For the string to still be a valid member of L_1, then $2p + l \leq 2p$, implying that $l = 0$ so $|vx| = 0$, contradicting the pumping lemma.

(b) $\left\{a^n b^{n^2} : n \geq 0\right\}$

The language is not a CFL.

Suppose it is, then, following the pumping lemma (1.1), take p to be the pumping length and $s = a^p b^{p^2}$ to be a string in the language.

Write $s = uvwxy$, the pumping lemma tells us that $|vwx| \leq 0$, $|vx| > 0$, and $uv^k wx^k y \in L_1$ for all $k \geq 0$.

There are two scenarios;

(i) $vwx = a^i$ for some $0 < i \leq p$. Since $|vx| > 0$, $vx\ vx = a^l$ for some $0 < l < i$. Pumping down tells us that $a^{p-l} b^{p^2} \in L_1$, so $p^1 = (p-l)^2$, which implies that $l = 0$, but that contradicts the pumping lemma's condition that $|vx| > 0$.

(ii) $vwx = a^i b^j$ for some $0 < i$, $0 < j$, and $i + j \leq p$. Since $|vx| > 0$, $vx = a^{l_1} b^{l_2}$ with the condition that $0 < l_1 + l_2 < i + j$

- If $l_1 = 0$, then pumping up the string increases the number of b's but does not change the number of a's, breaking the condition that the number of b's equal to the square of the number of a's.
- If $l_1 > 0$ and $l_2 = 0$, then pumping up the string increases the number of a's but does not change the number of b's, breaking the condition that the number of b's equal to the square of the number of a's.
- If $l_1 > 0$ and $l_2 > 0$, then pumping up the string tells us that $a^{p+l_1} b^{p^2+l_2} \in L_1$, so $p^2 + l_2 = (p + l_1)^2$.

  By expanding the equation, we see that:

$$
\begin{aligned}
p^2 + l_2 &= (p + l_1)^2 \\
&= p^2 + 2pl_1 + l_1^2 \\
\therefore l_2 &= 2pl_1 + l_1^2 \\
\therefore l_2 &> p \qquad \text{since } l_1 \text{ (string length) must be positive!} \\
\therefore l_1 + l_2 &= |vx| > p \qquad \text{(contradiction)}
\end{aligned}
$$

Problem 2.

Do the same for each of the following languages:

(a) $\{b_i \# b_{i+1} : i \geq 1\}$, where b is the binary representation of i with no leading 0's.

(b) $(a \cup b)^* - \{(a^n b^n)^n : n \geq 1\}$

Problem 3.

Formally describe a two-tape deterministic TM that decides the language $w \in \{0,1\}^* : w = w^R$. Provide a diagram, plus explanations and comments, so that your grader can understand how your TM works. You may assume that in a particular move one (or both) of the heads is allowed to remain stationary. Thus, the transition function for such a TM would look like

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

Indicate $\delta(q, a, b) = (r, c, d, L, S)$ by drawing an arrow from q to r and labeling it "$(a, b) \to (c, d), (L, S)$." Appreciate the ease of programming with two tapes for this language.

Strategy: Let tape A and tape B be the two tapes in the TM. Note that at the start, the input string is on tape A (plus infinite blanks on the right end), and tape B is blank.

(i) Shift the input string on tape A to the right by 1, and insert a $\#$ symbol at the left end of tape A to mark the beginning. We also insert a $\#$ at the left-most position on tape B to mark the beginning.

(ii) Move the head on tape A to the right end of the string and the head on tape B to the left second position (just after the $\#$). Traverse backwards on tape A and forwards on tape B, copying the string (in reverse) to tape B.

(iii) Move both heads to beginning of each tape. Traverse forwards on tape A and tape B, comparing the corresponding elements of tape A and tape B. If they are not equal, then $w \neq w^R$. Reject the input string. If they are equal, then $w = w^R$. Accept the input string.

Note: In the turing machine diagram, when we read two defined symbols (x, y) during a transition, we specifically mean that $x \neq y$ (otherwise, we denote the reading as (x, x) to specify that the readings from the two tapes are equivalent). When we read $(*, *)$ during a transition, we mean the two readings may be be equal or unequal, or any of the readings may be undefined (presumably, the transition is agnostic to the readings). However, if we do write the symbol back to the tape, it's the exact symbol that we read from the tape.

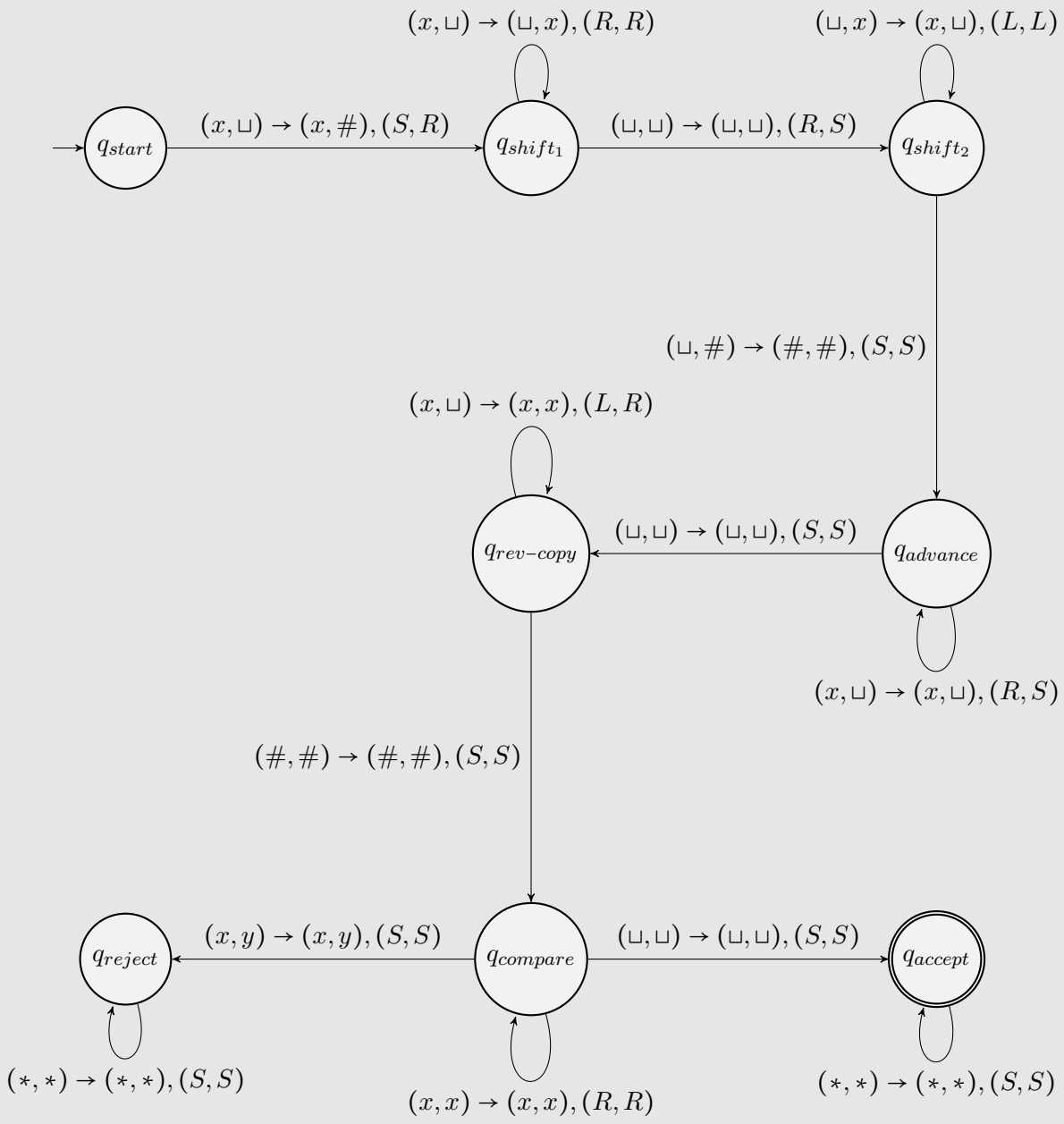

Figure 1. Turing Machine to decide $L = \left\{w \in \{0, 1\}^* : w = w^R\right\}$

Problem 4.

Formally describe a two-tape NDTM for the language $\{ww : w \in \{0,1\}^*\}$. Again, provide a diagram, plus explanations and comments.

Appreciate the ease of programming resulting from nondeterminism and the availability of two tapes.

Strategy: Let tape A and tape B be the two tapes in the TM. Note that at the start, the input string is on tape A (plus infinite blanks on the right end), and tape B is blank.

(i) Add a left-end marker (#) to tape B and move the head on tape B one step right.

(ii) Traverse tape A from left to right, copying the contents of tape A onto tape B.

(iii) Non-deterministically, stop copying the contents of tape A onto tape B, move the head on tape B to the left-most end (but maintain the head on tape A at its current position) and compare the remaining contents on tape A with those copied onto tape B.

(iv) Accept if there is such a computational path that copies some (possibly empty) part of the input string onto tape B, compares it with the remaining part of the string on tape A, and successfully reaches the end of the string.

Note: In the turing machine diagram, when we read two defined symbols (x, y) during a transition, we specifically mean that $x \neq y$ (otherwise, we denote the reading as (x, x) to specify that the readings from the two tapes are equivalent). When we read $(*, *)$ during a transition, we mean the two readings may be be equal or unequal, or any of the readings may be undefined (presumably, the transition is agnostic to the readings). However, if we do write the symbol back to the tape, it's the exact symbol that we read from the tape.

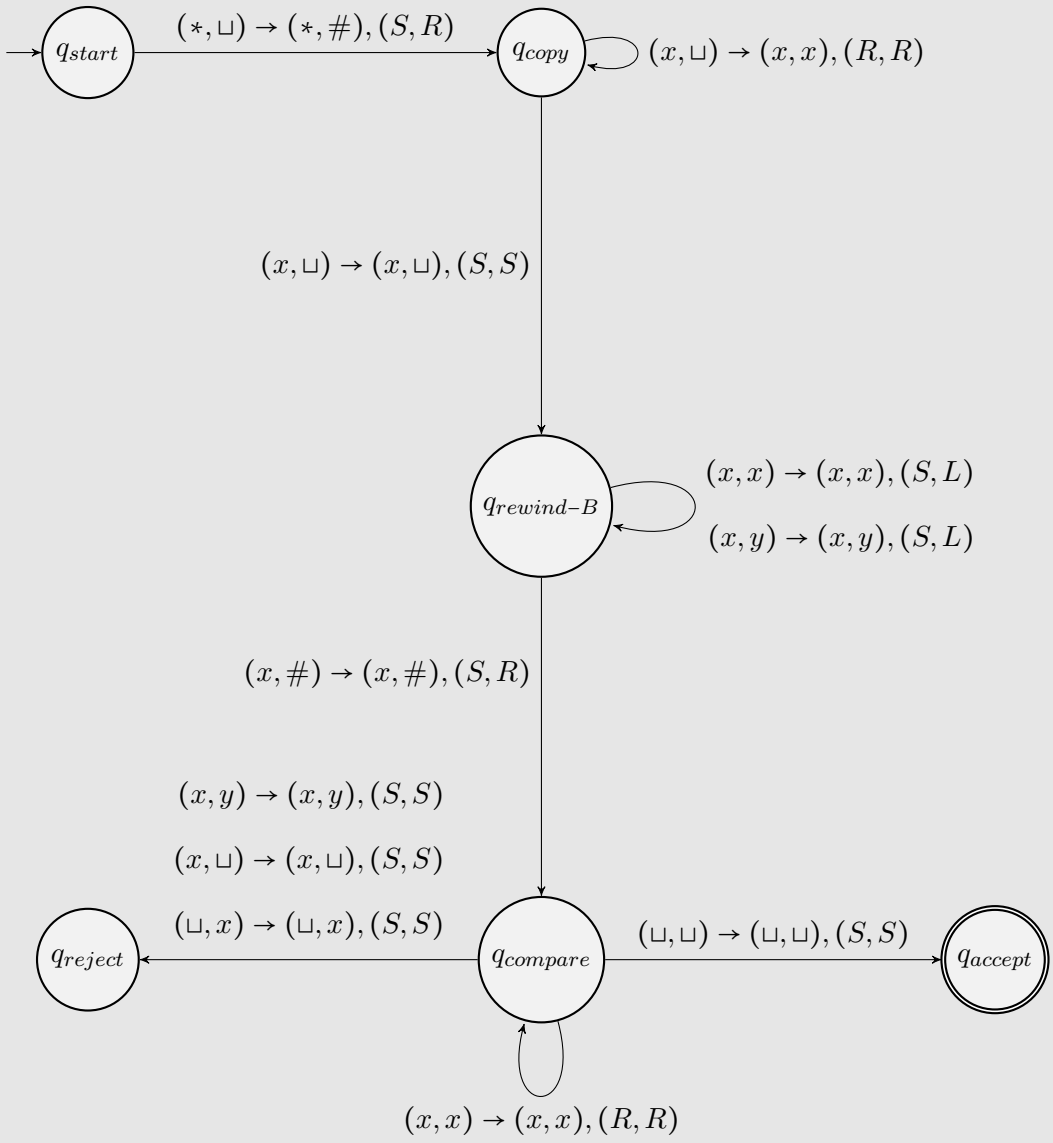

Figure 2. Turing Machine to decide $L = \{ww : w \in \{0, 1\}^*\}$

Problem 5.

Show that a k-tape TM M can be simulated by a single-tape TM M' in such a way that a computation that takes time t (i.e., t steps of one configuration yielding another) on M takes time $O(t^2)$ on M'. The big-O notation may hide a constant that depends on k. You may assume that $t \geq |x|$, where x is the input to M.

Problem 6.

Consider the following CFG:

$$S \to 1S00 \mid 00S1 \mid SS \mid 0S1S0 \mid \varepsilon$$

(a) Give a simple description of the language it generates using set-builder notation.

$$L = \left\{x \in \{0,1\}^* : N_0(x) = 2N_1(x)\right\}$$

(b) Now for the hard and fun part: prove the correctness of your answer.

We shall prove this in two parts. First, we show by induction on the form of the string that the condition holds for all the derivations of S, then we show that every string can be written as one of the derivations of S.

Claim 6.1. $N_0(S) = 2N_1(S)$ ***for all derivations of*** S**.**

Proof. We shall show this by induction on the form of S.

Base Case: $S \Rightarrow \varepsilon$

In this case, $N_0(S) = 0$ and $N_1(S) = 0$, so the condition $N_1(S) = 2N_0(S)$ holds.

Inductive Step:

(i) $S \Rightarrow 1S00$

Assuming $N_0(S) = 2N_1(S)$ for the smaller case, let $x = N_1(S)$.

Then $N_0(S) = N_0(S) + 2 = 2x + 2 = 2(x + 1)$, and $N_1(S) = N_1(S) + 1 = x + 1$.

Therefore, $N_0(S) = 2N_1(S)$ and the condition holds.

(ii) $S \Rightarrow 00S1$

Assuming $N_0(S) = 2N_1(S)$ for the smaller case, proceeding as in case (i) above we also see that $N_0(S) = 2N_1(S)$.

(iii) $S \Rightarrow SS$

Assuming that $N_0(S) = 2N_1(S)$ and $N_0(S) = 2N_1(S)$ for the two smaller cases,
let $x = N_1(S)$ and $y = N_1(S)$.

We see that $N_0(S) = N_0(S)+N_0(S) = 2x+2y = 2(x+y)$, and $N_1(S) = N_1(S)+N_1(S) = x+y$.

Therefore, $N_0(S) = 2N_1(S)$ and the condition holds.

(iv) $S \Rightarrow 0S1S0$

Assuming that $N_0(S) = 2N_1(S)$ and $N_0(S) = 2N_1(S)$ for the two smaller cases, let $x = N_1(S)$ and $y = N_1(S)$.

Then $N_0(S) = N_0(S) + N_0(S) + 2 = 2x + 2y + 2 = 2(x + y + 1)$, and $N_1(S) = N_1(S) + N_1(S) + 1 = x + y + 1$.

Therefore, $N_0(S) = 2N_1(S)$ and the condition holds.

□

Claim 6.2. *Every string $x \in L$ can be expressed as a derivation of S.*

Proof. First, let's define a useful metric. For any string x, let $f(x) = N_0(x) - 2N_1(x)$. This means that $f(x) = 0$ if and only if $x \in L$.

Now let's look at the strings in L and see how the metric changes over the length of the strings. Let x be an arbitrary string in L. There are two general cases:

(i) x starts and ends with the same symbol. Write $x = x_1x_2 \dots x_n$ so that $x_1 = x_n$. Since $x \in L$, $f(x) = 0$.

- If $x_1 = 0$ and $x_n = 0$, that means $f(x_1) = 1$ and $f(x_1x_2 \dots x_{n-1}) = -1$. For the metric to move from 1 to -1, we must have crossed the zero line at some point. However, since the function decreases by 2, there are 2 possible scenarios:
  - If the metric equals 0 at some point, we can split the string as in the previous case.
  - If the metric does not equal 0 at any point, then it must transition from 1 to -1 at the point where it crossed the zero line (say, k). Therefore, ignoring the 0 causing the decrease and the zeros at the beginning and the end of the string, we have that $f(x_2 \dots x_{k-1}) = 0$ and $f(x_{k+1} \dots x_{n-1}) = 0$. Therefore, we can write the string as $s = 0S1S0$.
- If $x_1 = 1$ and $x_n = 1$, that means $f(x_1) = -2$ and $f(x_1x_2 \dots x_{n-1}) = 2$. For the metric to move from -2 to 2, we must have crossed the zero line. However, the metric always increases by 1 so there must exist some j between 1 and $n-1$ so that $f(x_1x_2 \dots x_j) = 0$. We can therefore split the string $x_1x_2 \dots x_n$ into $x_1x_2 \dots x_j$ and $x_jx_{j+1} \dots x_n$, with each of the substrings being a member of L.

(ii) x starts and ends with differing symbols.

Write $x = x_1x_2 \dots x_n$ so that $x_1 \neq x_n$. Suppose $x_0 = 0$ and $x_n = 1$. Then $f(x_1) = 1$ and $f(x_{n-1}) = 2$. if $x_2 = 1$, we can split the sentence into two! Therefore, x_2 must be 0 so we can write the sentence as $00S1$. The same argument applies for when 0 and 1 are interchanged.

□