

## PSET 4 — 02/06/2023

Prof. Chakrabarti

Student: Amittai Siavava

## Credit Statement

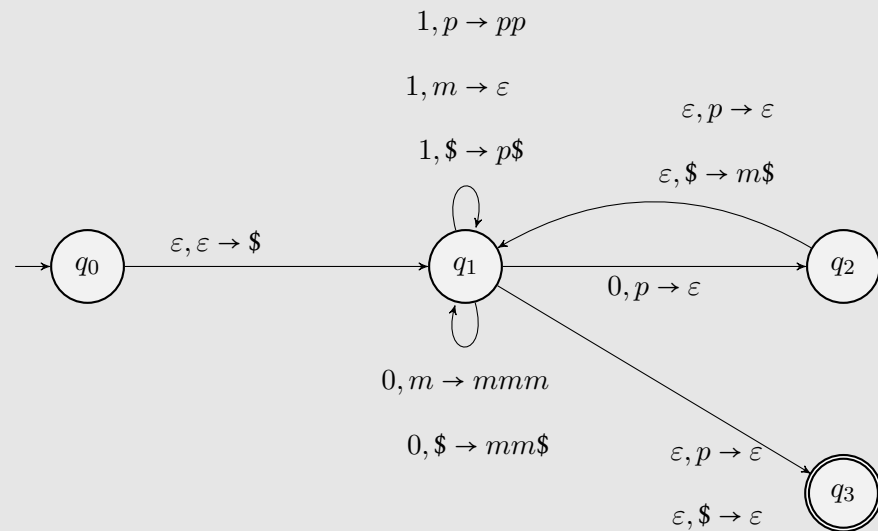
I discussed ideas for this homework assignment with Paul Shin.

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.

Draw a PDA that recognizes the language  $L = \{x \in \{0, 1\}^* : N_1(x) \geq 2N_0(x)\}$ . Give a high-level proof that your PDA works correctly.

FIGURE 1. A PDA recognizing  $L$

**High-Level Idea and Proof of Correctness**

We use have stack variables:  $p$ ,  $m$ , and  $\$$ . Using these, we track the value of  $N_1(x) - 2N_0(x)$  as we read the string.

A  $p$  corresponds to a '+1', an  $m$  corresponds to a '-1', and  $\$$  corresponds to a 0. This is how the PDA works:

- First, we enforce that no stack state can contain both  $p$ 's and  $m$ 's at the same time. We do this by only starting to push  $p$ 's (or  $m$ 's in the alternate case) if the symbol at the top of the stack is  $\$$ , symbolizing a 0.
- We start by pushing a  $\$$  onto the stack, signifying a state of 0.
- Whenever we read a 0, we decrease the stack state by 2. This takes three forms:
  - We can remove two  $p$ 's from the stack.
  - If we only have a single  $p$  at the top of the tack, we remove it and push a single  $m$ .
  - If we have an  $m$  or the zero marker ( $\$$ ) at the top of the stack, we return it and push two more  $m$ 's.
- Whenever we read a 1, we increase the stack state by 1. We do this by:
  - If we have a  $p$  or a  $\$$  at the top of the stack, return it and push another  $p$ .
  - If we have an  $m$  at the top of the stack, remove it.
- Consequently, when we reach the end of the string:
  - If we have a  $\$$  at the top of the stack, that means we have encountered *exactly* twice as many 1's as 0's, so we accept the string.
  - If we have a  $p$  at the top of the stack, that means the number of 0's we have encountered is more than twice the number of 1's we have encountered, so we accept the string.
  - Otherwise, the number of 1's was less than twice the number 0's in the string, so we do not generate a transition to the accepting state.

**Problem 2.**

In class, we wrote a formal construction of a PDA that proves that context-free languages are closed under union.

Give similar constructions for PDAs to prove closure under:

- (a) concatenation.

Let  $L_1$  and  $L_2$  be context-free languages. Take  $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_{01}, F_1)$  to be a PDA that recognizes  $L_1$  and  $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_{02}, F_2)$  to be a PDA that recognizes  $L_2$ .

Construct a new PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  as follows:

- $Q = Q_1 \cup Q_2$  (after enforcing that  $Q_1 \cap Q_2 = \emptyset$ )
- $\Sigma = \Sigma_1 \cup \Sigma_2$  (we don't particularly care about equality of  $\Sigma_1$  and  $\Sigma_2$ )
- $\Gamma = \Gamma_1 \cup \Gamma_2$  (we don't particularly care about equality of  $\Gamma_1$  and  $\Gamma_2$ )
- $q_0 = q_{01}$
- $F = F_2$
- $\delta$  is defined as follows:

$$\delta(q, a, \gamma) = \begin{cases} \delta_1(q, a, \gamma) \cup \{q_{02}, \emptyset\} & \text{if } q \in F_1 \text{ and } a = \gamma = \varepsilon. \\ \delta_1(q, a, \gamma) & \text{if } q \in Q_1. \\ \delta_2(q, a, \gamma) & \text{if } q \in Q_2. \end{cases}$$

**Claim 2.1.**  $M$  recognizes  $L_1 \cup L_2$ .

*Proof.* Note that the starting state of  $M$  is  $q_{01}$ , while the accepting states of  $M$  are in  $F_2$ . The only transition that takes  $M$  from a state formerly in  $Q_1$  to a state formerly in  $Q_2$  is when (1) we are at a state  $q \in F_1$  (an accepting state of  $M_1$ ), (2) we read no input (epsilon transition), and (3) we clear the stack.

Since  $M_1$  recognizes  $L_1$ , we know that the computational path of  $M_1$  on all strings in  $L_1$  ends in an accepting state  $q_{f1} \in F_1$ . Likewise, since  $M_2$  recognizes  $L_2$ , we know that the computational path of  $M_2$  on all strings in  $L_2$  ends in an accepting state  $q_{f2} \in F_2$ .

- **Completeness:** If a string  $s$  is in  $L_1 L_2$ , then we can write it as  $s = xy$  for some  $x \in L_1$  and  $y \in L_2$ . Then, the computational path of  $M$  on  $x$  mimics that of  $M_1$  (since it starts at  $q_{01}$  and we use  $\delta_1$  for all states  $q \in Q_1$ ). Therefore,  $M$  has a computational path on  $x$  that ends in an accepting state  $q_{f1} \in F_1$ . Then,  $M$  takes the epsilon transition to  $q_{02}$ . In processing  $y$ ,  $M$  starts at  $q_{02}$  and uses  $\delta_2$  for all states  $q \in Q_2$ , so it has some computational path from  $q_{02}$  to  $q_{f2} \in F_2$ . Putting these two

paths together and the middle epsilon transition, we get a computational path of  $M$  from  $q_{01}$  to  $q_{f2} \in F_2$ , which is an accepting state of  $M$ . Therefore,  $M$  accepts the string.

- **Soundness:** Let  $s$  be a string accepted by  $M$ . Then there must exist some computational path  $p_1$  of  $M$  on  $s$ , taking  $M$  from  $q_{01}$  to a state in  $F_1$ , followed by the epsilon transition to  $q_{02}$ , and some computational path  $p_2$  of  $M$  from  $q_{02}$  to a state in  $F_2$ . By definition of  $M$ ,  $p_1$  corresponds to an accepting computational path of  $M_1$  and  $p_2$  corresponds to an accepting computational path of  $M_2$ . Meaning that  $M_1$  accepts some prefix  $x$  of  $s$  and  $M_2$  accepts some suffix  $y$  of  $s$ , and  $x$  and  $y$  form the entire string  $s$ , so  $s = xy$ ,  $x \in L_1$ ,  $y \in L_2$ . Therefore, any such  $s$  accepted by  $M$  is in  $L_1 L_2$ . □

(b) Kleene star.

Let  $L$  be a context-free language. Take  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  to be a PDA that recognizes  $L$ .

Construct a new PDA  $M_2 = (Q \cup q_{start}, \Sigma, \Gamma, \delta_2, q_0, F \cup q_{start})$  where:

- $q_{start} \notin Q$
- $\delta_2$  is defined as follows:

$$\delta_2(q, a, \gamma) = \begin{cases} \delta(q, a, \gamma) \cup \{(q_0, \emptyset)\} & \text{if } q \in F \text{ and } a = \gamma = \varepsilon. \\ \{(q_0, \emptyset)\} & \text{if } q = q_{start} \text{ and } a = \gamma = \varepsilon. \\ \delta(q, a, \gamma) & \text{otherwise.} \end{cases}$$

**Claim 2.2.**  $M_2$  recognizes  $L^*$ .

*Proof.* Note that:

**Completeness:** If a string  $s$  is in  $L^*$ , then, either:

- (i)  $s = \varepsilon$ . Since  $q_{start}$  is an accepting state of  $M_2$ ,  $M_2$  accepts  $s$ .
- (ii)  $s = x_1 x_2 \cdots x_n$ , with all  $x_i \in L$ . Then, for each  $x_i$ , there exists some computational path that takes  $M_2$  from  $(q_0, \emptyset)$  to some  $(q_f, \Gamma_0)$  where  $q_f \in F$  and  $\Gamma_0 \in \Gamma^*$ . Since  $(q_0, \emptyset)$  is in the set of possible next states for epsilon transitions on all accepting states (as defined in case 1 of  $\delta_2$ ), we have a connecting path from any such  $(q_f, \Gamma_0)$  to  $(q_0, \emptyset)$  between any  $x_i$  and  $x_{i+1}$ . Therefore,  $M_2$  has some computational path that:
  - (i) starts at  $(q_{start}, \emptyset)$ ,
  - (ii) Advances to some accepting state  $(q_f, \Gamma_0)$  after reading  $x_1$ ,
  - (iii) Takes an  $\varepsilon$ -transition back to  $(q_0, \emptyset)$ ,
  - (iv) Advances to some accepting state  $(q_{f2}, \Gamma_1)$  after reading  $x_2$ ,

- (v) Takes an  $\varepsilon$ -transition back to  $(q_0, \emptyset)$  again,
- (vi) Repeats the process for  $x_3, \dots, x_n$ , and
- (vii) Is in some accepting state after finishing reading  $x_n$  (but not taking the  $\varepsilon$ -transition back to  $(q_0, \emptyset)$ ).

Therefore,  $M_2$  has a computational that accepts  $s$ , so  $M_2$  accepts  $s$ .

**Soundness:** If  $s$  is accepted by  $M_2$ , then, either:

- (i)  $s = \varepsilon$ , since  $q_{start}$  is an accepting state. Since  $\varepsilon \in L^*$  for any language, then  $s$  is a valid string in  $L^*$ .
- (ii) Otherwise, we claim that  $s = x_1x_2\cdots x_n$ , with all  $x_i \in L$ .

$M_2$  mimics the transitions of  $M$ , only adding a new start state and  $\varepsilon$ -transitions from accepting states to the old start state. Therefore, if  $s$  is accepted by  $M_2$ , then a suffix  $x_1$  of  $s$  (which might be the whole string) takes  $M_2$  from  $(q_0, \emptyset)$  to some accepting state  $(q_f, \Gamma_0)$  where  $q_f \in F$ . If the suffix is NOT the whole string, write the whole string  $s$  as  $s = px_1$ , then if we erase the suffix  $x_1$  then the prefix  $p$  must also end up in some accepting state of  $M_2$ . Continuing in the same way, we can extract a suffix  $x_2$  of  $p$ , and so on up to some  $x_n$  where we remain with the empty string. Therefore, we can write  $s = x_nx_{n-1}\cdots x_1$  where all  $x_i \in L$ . But this is exactly identical to writing  $s = x_1x_2\cdots x_n$  required for  $s$  to be in  $L^*$ , only that the numbering of  $x$  is reversed. Therefore, any such accepted string  $s$  must be in  $L^*$ .

□

**Problem 3.**

Give an alternate proof, using CFGs alone (no PDAs), to prove that context-free grammars are closed under:

(a) union.

Let  $G_1 = (V_1, \Sigma_1, R_1, S_1)$  and  $G_2 = (V_2, \Sigma_2, R_2, S_2)$  such that  $G_1$  and  $G_2$  generate  $L_1$  and  $L_2$ , respectively. Define  $G = (V, \Sigma, R, S)$  as follows:

- $V = V_1 \cup V_2 \cup \{S\}$ , where  $S \notin V_1 \cup V_2$  and  $V_1 \cap V_2 = \emptyset$ .
- $\Sigma = \Sigma_1 \cup \Sigma_2$
- $R = R_1 \cup R_2 \cup \{(S, S_1), (S, S_2)\}$

**Claim 3.1.**  $G$  generates  $L_1 \cup L_2$ .

*Proof.* We show that  $G$  is a CFG that generates  $L_1 \cup L_2$ .

(i) **Completeness:** Let  $w$  be a string in  $L_1 \cup L_2$ . This means that, either:

- $w \in L_1$ , so there exists some derivation  $S_1 \Rightarrow^* w$  from  $G_1$ , or
- $w \in L_2$ , so there exists some derivation  $S_2 \Rightarrow^* w$  from  $G_2$ .

Note that  $G$  is defined such that  $V_1 \subset V$  and  $V_2 \subset V$ . Likewise,  $R_1 \subset R$  and  $R_2 \subset R$ . Therefore, any such derivation can be deduced in  $G$  *starting from the relevant symbol, of either  $S_1$  or  $S_2$* . However, the start symbol in  $G$  is  $S$ , so a derivation  $S \Rightarrow^* S_1$  or  $S \Rightarrow^* S_2$  is needed to be able to derive strings from  $L_1$  or  $L_2$  respectively. Since the definition of  $G$  adds two new rules,  $(S, S_1)$  and  $(S, S_2)$ , the derivation  $S \Rightarrow^* S_1$  and  $S \Rightarrow^* S_2$  are possible. So any string that can be generated by  $G_1$  can also be generated by  $G$ , and any string that can be generated by  $G_2$  can also be generated by  $G$ , meaning  $G$  can generate any string in  $L_1 \cup L_2$ .

(ii) **Soundness:** If a string is generated by  $G$ , we claim that it is in  $L_1 \cup L_2$ . Note that  $G$  has a single start symbol,  $S$ , and the only rules including  $S$  are  $(S, S_1)$  and  $(S, S_2)$ . This means from  $S$  we can only derive *either  $S_1$  or  $S_2$ , but not both, and not any other symbol*. Since  $S_1 \in V_1$  and  $S_2 \in V_2$  and we defined  $V_1$  and  $V_2$  to be disjoint, the only strings that can be generated from  $S_1$  must be in  $L_1$  (using the rules in  $R_1$ ) and the only strings that can be generated from  $S_2$  must be in  $L_2$  (using the rules in  $R_2$ ). Therefore, any string that can be generated by  $G$  must be either in  $L_1$  or in  $L_2$ , meaning any string  $G$  generates is in  $L_1 \cup L_2$ .

□

(b) concatenation.

Let  $G_1 = (V_1, \Sigma_1, R_1, S_1)$  and  $G_2 = (V_2, \Sigma_2, R_2, S_2)$  such that  $G_1$  and  $G_2$  generate  $L_1$  and  $L_2$ , respectively. Define  $G = (V, \Sigma, R, S)$  as follows:

- $V = V_1 \cup V_2 \cup \{S\}$ , where  $S \notin V_1 \cup V_2$  and  $V_1 \cap V_2 = \emptyset$ .
- $\Sigma = \Sigma_1 \cup \Sigma_2$
- $R = R_1 \cup R_2 \cup \{(S, S_1S_2)\}$

**Claim 3.2.**  $G$  generates  $L_1L_2$ .

*Proof.* We show that  $G$  is a CFG that generates  $L_1L_2$ .

- (i) **Completeness:** Let  $w$  be a string in  $L_1L_2$ . This means that, for some  $u \in L_1$  and  $v \in L_2$ ,  $w = uv$ . Therefore, there exists some derivation  $S_1 \Rightarrow^* u$  in  $G_1$  and some derivation  $S_2 \Rightarrow^* v$  from  $G_2$ . Since  $V_1 \subset V$ ,  $V_2 \subset V$ ,  $R_1 \subset R$ , and  $R_2 \subset R$ , these derivations are also possible in  $G$  *starting from the relevant symbol, of either  $S_1$  or  $S_2$* . But the start symbol in  $G$  is  $S$ , so a derivation  $S \Rightarrow^* S_1S_2$  is needed to be able to derive strings from  $L_1L_2$ . The definition of  $G$  adds this rule,  $(S, S_1S_2)$ , so the derivation  $S \Rightarrow^* S_1S_2$  is possible. Therefore, any string in  $L_1L_2$  can be generated by  $G$ .
- (ii) **Soundness:** If a string is generated by  $G$ , we claim that it is in  $L_1L_2$ .  $G$  has a single start symbol,  $S$ , and the only rule from  $S$  is  $(S, S_1S_2)$ . This means from  $S$  we can only derive  $S_1S_2$ . Since  $S_1 \in V_1$  and  $S_2 \in V_2$  and we defined  $V_1$  and  $V_2$  to be disjoint, the only strings that can be generated from  $S_1$  must be in  $L_1$  (using the rules in  $R_1$ ) and the only strings that can be generated from  $S_2$  must be in  $L_2$  (using the rules in  $R_2$ ). Therefore, any string that can be generated by  $G$  must be the concatenation of a string in  $L_1$  and a string in  $L_2$ , so any string  $G$  generates is in  $L_1L_2$ .

□

(c) Kleene star.

Let  $G_1 = (V_1, \Sigma_1, R_1, S_1)$  be a CFG that generates  $L$ . Define  $G = (V, \Sigma, R, S)$  as follows:

- $V = V_1 \cup \{S\}$ , where  $S \notin V_1$ .
- $\Sigma = \Sigma_1$
- $R = R_1 \cup \{(S, S_1 S), (S, \varepsilon)\}$

**Claim 3.3.**  $G$  generates  $L^*$ .

*Proof.* We show that  $G$  is a CFG that generates  $L^*$ .

(i) **Completeness:** Let  $w$  be a string in  $L^*$ . There are two possible scenarios:

- (i)  $w = \varepsilon$ : Since we have the rule  $S \Rightarrow \varepsilon$ ,  $G$  can generate  $\varepsilon$ .
- (ii)  $w = w_1, \dots, w_n$  with all  $w_i \in L$ . This means that there exists some derivation  $S_1 \Rightarrow^* w_1$  in  $G_1$ ,  $S_1 \Rightarrow^* w_2$  in  $G_1$ , ..., and  $S_1 \Rightarrow^* w_n$  in  $G_1$ . Since  $V_1 \subset V$  and  $R_1 \subset R$ , each one of these derivations is also possible in  $G$  starting from the relevant symbol,  $S_1$ . To derive their concatenations starting from  $S$ , we need a rule that can recursively derive  $S_1$  multiple times from  $S$ . We define this rule in the definition of  $G$  as  $(S, S_1 S)$ , allowing  $G$  to derive  $S_1 S_1 S_1 \dots S_1 S$  from  $S$ , then eventually replace the  $S$  with  $\varepsilon$  and derive each  $w_i$  from the corresponding  $S_1$ .

(ii) **Soundness:** If a string is generated by  $G$ , we claim that it is in  $L^*$ .  $G$  has a single start symbol,  $S$ , which yields either  $\varepsilon$  or  $S_1 S$ . the first case generates  $\varepsilon$ , which is in  $L^*$ . In the second case, repeated expansion of  $S$  in the expression yields  $S_1 S_1 S_1 \dots S_1 S$ . Each  $S_1$  eventually yields a string in  $L$ , and the final  $S$  yields  $\varepsilon$ . Therefore, any string that can be generated by  $G$  must either be the empty string or a concatenation of strings from  $L$  — meaning it is in  $L^*$ .

□



**Problem 4.**

A string  $x \in \Sigma^*$  is called a *square* if  $x = w^2$  for some  $w \in \Sigma^*$ . Let  $L_{sq} = \{w^2 : w \in \{0, 1\}^*\}$ . Consider its complement:

$$\overline{L}_{sq} = \{x \in \{0, 1\}^* : x \text{ is not of the form } w^2 \text{ for any } w \in \{0, 1\}^*\}.$$

- (a) Prove that every even-length string in  $\overline{L}_{sq}$  can be decomposed as  $x = uv$  where the middle symbol of  $u$  differs from the middle symbol of  $v$ .

Let  $x = uv$  be a string in  $\overline{L}_{sq}$  such that  $|u| = |v|$ . Suppose the string  $x$  has length  $2n$ , such that  $x = u_1u_2\cdots u_nv_1v_2\cdots v_n$ . Since  $u \neq v$  (by definition of  $\overline{L}_{sq}$ ), it must be the case that  $u_i \neq v_i$  for some  $1 \leq i \leq n$  (maybe multiple values of  $i$ , but we only care about one).

Suppose  $k$  is the smallest such  $i$  with  $u_k \neq v_k$ .

- (i) If  $k \leq \frac{n}{2}$ , take  $s_1 = u_1, u_2, \dots, u_{2k-1}, \dots, u_n$  and  $s_2 = v_1, v_2, \dots, v_{2k-1}, \dots, v_n$ . Then  $|s_1| = 2k - 1$  meaning the middle symbol of  $s_1$  is  $u_k$ . Likewise,  $|s_2| = n + n - (2k - 1) = 2n - 2k + 1$ , meaning the middle element is at position  $n - k$ . Since  $s_1$  starts at  $2k$ , the middle element is at position  $2k + n - k = n + k$ . This is the element corresponding to  $v_k$  and  $u_k \neq v_k$  so the two strings  $s_1$  and  $s_2$  have differing middle symbols.
- (ii) If  $k > \frac{n}{2}$ , proceed as above but counting up to the corresponding element in  $v$  from the end of the string.

- (b) Using this property, design a context-free grammar that generates  $\overline{L}_{sq}$ .

**Derivations**

$$S \Rightarrow AB \mid BA \mid X$$

$$A \Rightarrow 0A0 \mid 0A1 \mid 1A0 \mid 1A1 \mid 0$$

$$B \Rightarrow 0B0 \mid 0B1 \mid 1B0 \mid 1B1 \mid 1$$

$$X \Rightarrow 0X0 \mid 0X1 \mid 1X0 \mid 1X1 \mid 0 \mid 1$$

**CFG**

$G = (V, \Sigma, R, S)$  where:

$$V = \{S, A, B, X\}$$

$$\Sigma = \{0, 1\}$$

$$R = R_S \cup R_A \cup R_B \cup R_X$$

$$R_S = \{(S, AB), (S, BA), (S, X)\}$$

$$R_A = \{(A, 0A0), (A, 0A1), (A, 1A0), (A, 1A1), (A, 0)\}$$

$$R_B = \{(B, 0B0), (B, 0B1), (B, 1B0), (B, 1B1), (B, 1)\}$$

$$R_X = \{(X, 0X0), (X, 0X1), (X, 1X0), (X, 1X1), (X, 0), (X, 1)\}$$

**Idea Behind CFG**

First, note that odd-length strings cannot be squares, so they are all in  $\overline{L}_{sq}$ . To generate these, we add rules to generate strings of odd length, without restriction to the middle symbol (as defined in  $R_X$ ).

To handle strings of even length that are not squares, we use the property that they must be decomposable into two odd-sized strings with differing middle symbols. We add the rules  $S \Rightarrow AB \mid BA$  to generate such strings, where  $A$  generates odd-length strings with a 0 as a middle symbol while  $B$  generates odd-length strings with a 1 as a middle symbol.

**Problem 5.**

Let  $\Sigma$  be an alphabet,  $L \subseteq \Sigma^*$ , and  $\# \notin \Sigma$ . Define the language

$$\text{INTERSPERSE}(\#, L) := \{a_1 \# a_2 \# \dots \# a_n\}, \text{ each } a_i \in \Sigma \text{ and } a_1 a_2 \dots a_n \in L.$$

Let  $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, F)$  be a PDA that recognizes  $L$ . Formally describe a PDA that recognizes  $\text{INTERSPERSE}(\#, L)$ .

Also give a high-level proof that your PDA works correctly.

Let  $M_2 = (\bigcup_{q \in Q} \{q, q_\#, q_\varepsilon\}, \Sigma \cup \{\#\}, \Gamma, \delta_2, q_0, F_2)$  be a PDA such that:

- $F_2 = \{q_\varepsilon : q \in F\} \cup \{q_\# : q \in F\}$
- $\delta_2$  is defined as follows:

$$\delta_2(q, \varepsilon, \gamma) = \delta(q, a, \gamma)_\varepsilon$$

$$\delta_2(q, x, \gamma) = \delta(q, a, \gamma)_\# \text{ if } x \in \Sigma$$

$$\delta_2(q_\#, \#, \varepsilon) = q$$

$$\delta_2(q_\varepsilon, \varepsilon, \varepsilon) = q$$

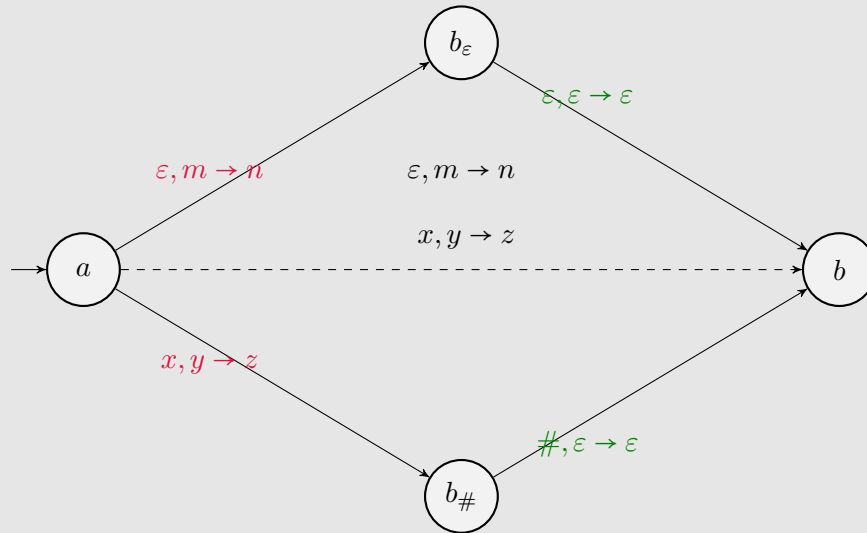


FIGURE 2. Modification of Transitions to Require  $\#$  between any two non-epsilon Symbols.

**Claim 5.1.**  $M_2$  recognizes  $INTERSPERSE(\#, L)$ .

*Proof.* In contrast to  $M_1$ ,  $M_2$  makes the following modifications:

- (i) Any transition that takes the PDA from a state  $a$  to a state  $b$ , now does one of the following:
  - Takes the PDA from  $a$  to  $b_\epsilon$  if the transition reads an epsilon input. From  $b_\epsilon$ , the PDA can choose to take a transition to  $b$  without reading any input or interfering with the stack, so  $b$  is still reachable.
  - Takes the PDA from  $a$  to  $b_\#$  if the transition reads a non-epsilon input. From  $b_\#$ , the only transition forward reads a  $\#$  symbol but does not interfere with the stack, thus ensuring that  $b$  is still reachable only if a  $\#$  symbol exists between the consecutive letters in the input string.
- (ii) For all accepting states  $f \in F_1$ , we define the states  $f_\#$  and  $f_\epsilon$  to be accepting states in  $M_2$ , and make  $f$  non-accepting in  $M_2$ . This ensures that:
  - If  $a_1 a_2 \dots a_n$  is accepted in  $M_1$ , then  $a_1 \# a_2 \# \dots \# a_n$  is accepted in  $M_2$ .
  - We do not accept strings ending with a  $\#$  symbol, e.g.  $a_1 \# a_2 \# \dots \# a_n \#$ , since the state  $f$  itself is non-accepting.

□

**Problem 6.**

Consider the following CFG:

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

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

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

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