CSE 105: Homework #3

Due on Feb 15, 2024 at 23:59pm

Professor Minnes

Ray Tsai

A16848188

Problem 1

Consider the push-down automata M_1 and M_2 over $\{a, b\}$ with stack alphabet $\{a, b\}$ whose state diagrams are

(a) (*Graded for completeness*) What is the language A_1 recognized by M_1 and what is the language A_2 recognized by M_2? Include a sample string that is accepted and one that is rejected for each of these PDA. Justify these examples with sample accepting computations or with an explanation why there is no accepting computation.

Proof. Since M_1 only contains a state with a loop which only takes b as an input, $A_1 = b^*$. Hence, $\varepsilon \in A_1$ and $a \notin A_1$, as the starting state is also an accepting state and there are simply no arrows for a's to follow.

For M_2 to take b as an input, there has to be at least one a in the memory stack, and thus the b's in the string cannot precede a's and the number of b is at most that of a. Hence, $A_2 = \{a^m b^n \mid m \geq n, m, n \in \mathbb{Z}_{\geq 0}\}$. For instance, $\varepsilon \in A_2$ and $b \notin A_2$. ε is obviously accepted as the starting state is also an accepting state. b is rejected as there are no a's in the memory stack for the b to be sent to q_2. □

(b) (*Graded for correctness*) Design CFGs G_1 and G_2 over $\{a, b\}$ so that $L(G_1) = A_1$ and $L(G_2) = A_2$. A complete solution will include precise definitions for each of the parameters required to specify a CFG, as well as a brief explanation about why each string in A_i can be derived in G_i and each string not in A_i cannot be derived in G_i (for $i = 1, 2$).

Proof. Let $\Sigma = \{a, b\}$. Consider

$$G_1 = (\{S_1\}, \Sigma, \{S_1 \to bS_1 \mid \varepsilon\}, S_1) \quad \text{and} \quad G_2 = (\{S_2\}, \Sigma, \{S_2 \to aS_2 \mid aS_2b \mid \varepsilon\}, S_2).$$

Since G_1 repeatedly generates b and terminates with ε, $L(G_1)$ contains A_1, the language of all strings that doesn't contain a. Let $s \notin A_1$. s contains A. Since G_1 does not generate a, $a \notin L(G_1)$. Hence, $L(G_1) = A_1$.

Let $k \in A_2$. k is of the form $a^m b^n$, for $m \geq n \geq 0$. Note that whenever G_2 generates b, it generates an a preceding the b, but the generation of a is not binded to b. Hence, k is obviously in $L(G_2)$. Let $p \notin A_2$. p either has a b preceding a or has more b than a, both of which contradicts the rules defined for G_2, and thus $p \notin L(G_2)$. □

(c) (*Graded for completeness*) Remember that the definition of set-wise concatenation is: for languages L_1, L_2 over the alphabet Σ, we have the associated set of strings

$$L_1 \circ L_2 = \{w \in \Sigma^* \mid w = uv \text{ for some strings } u \in L_1 \text{ and } v \in L_2\}$$

In class (and in the review quiz) we learned that the class of context-free languages is closed under set-wise concatenation. The proof of this closure claim using CFGs uses the construction: given $G_1 = (V_1, \Sigma, R_1, S_1)$ and $G_2 = (V_2, \Sigma, R_2, S_2)$ with $V_1 \cap V_2 = \emptyset$ and $S_{new} \notin V_1 \cup V_2$, define a new CFG

$$G = (V_1 \cup V_2 \cup \{S_{new}\}, \Sigma, R_1 \cup R_2 \cup \{S_{new} \to S_1 S_2\}, S_{new})$$

Problem 1 continued on next page...

Apply this construction to your grammars from part (b) and give a sample derivation of a string in $A_1 \circ A_2$ in this resulting grammar.

Proof. Applying this construction to G_1, G_2, we get

$$G = (\{S_1, S_2, S_{new}\}, \Sigma, R, S_{new}),$$

where the set of rules R is

$$\begin{aligned} S_{new} &\to S_1 S_2 \\ S_1 &\to bS_1 \mid \varepsilon \\ S_2 &\to aS_2 \mid aS_2 b \mid \varepsilon. \end{aligned}$$

An example of derivation of $\varepsilon \in A_1 \circ A_2$ is

$$S_{new} \Rightarrow S_1 S_2 \Rightarrow \varepsilon S_2 \Rightarrow \varepsilon.$$

□

(d) (*Graded for correctness*) If we try to extrapolate the construction that we used to prove that the class of regular languages is closed under set-wise concategation, we would get the following construction for PDAs: Given $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, F_2)$ with $Q_1 \cap Q_2 = \emptyset$, define $Q = Q_1 \cup Q_2$, $\Gamma = \Gamma_1 \cup \Gamma_2$, and

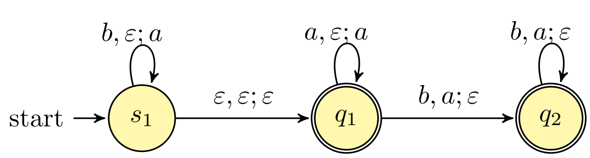
$$M = (Q, \Sigma, \Gamma, \delta, q_1, F_2)$$

with $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ given by

$$\delta(q, a, b) = \begin{cases} \delta_1(q, a, b) & \text{if } q \in Q_1 \setminus F_1,\ a \in \Sigma_\varepsilon,\ b \in \Gamma_{1_\varepsilon} \\ \delta_2(q, a, b) & \text{if } q \in Q_2,\ a \in \Sigma_\varepsilon,\ b \in \Gamma_{2_\varepsilon} \\ \delta_1(q, a, b) & \text{if } q \in F_1,\ a \in \Sigma \text{ or } b \in \Gamma_1 \\ \delta_1(q, a, b) \cup \{(q_2, \varepsilon)\} & \text{if } q \in F_1,\ a = \varepsilon,\ b = \varepsilon \\ \emptyset & \text{otherwise} \end{cases}$$

Apply this construction to the machines M_1 and M_2 from part (a), and then use the resulting PDA to prove that *this* construction cannot be used to prove that the class of context-free languages is closed under set-wise concatenation. A complete solution will include (1) the state diagram of the machine M that results from applying this construction to M_1 and M_2, (2) an example of a string that is accepted by this PDA M but that is **not** in the language $A_1 \circ A_2$ with a description of the computation that witnesses that this string is accepted by M and an explanation of why this string is not in $A_1 \circ A_2$ by referring back to the definitions of A_1, A_2, and set-wise concatenation.

Proof. The following is the state diagram for M:



Consider the string $s = babb$. $babb$ is accepted by M, following the path $s_1 \to q_1 \to q_2 \to q_2$. Note that s is allowed to go through q_2 twice, as the prefix ba pushes two a's onto the memory stack priorly. However, s is not a valid concatenation of A_1 and A_2, as $babb \notin A_1$ and none of $babb, abb, bb, b$ are in A_2. □

Problem 2

Informally, we think of regular languages as potentially simpler than context-free languages. In this question, you'll make this precise by showing that every regular language is context-free, in two ways.

(a) (*Graded for correctness*) When we first introduced PDAs we saw that any NFA can be transformed to a PDA by not using the stack of the PDA at all. Make this precise by completing the following construction: Given a NFA $N = (Q, \Sigma, \delta_N, q_0, F)$ we define a PDA M with $L(M) = L(N)$ by choosing ... A complete solution will have precise, correct definitions for each of the defining parameters of M: the set of states, the input alphabet, the stack alphabet, the transition function, the start state, and the set of accept states. Be careful to use notation that matches the types of the objects involved.

Proof. $M = (Q, \Sigma, \emptyset, \delta_M, q_0, F)$, where $\delta_M : Q \times \Sigma_\varepsilon \times \emptyset_\varepsilon \to \mathcal{P}(Q \times \emptyset_\varepsilon)$ sends (q, σ, γ) to $\delta_N(q, \sigma) \times \{\varepsilon\}$. $\square$

(b) (*Graded for correctness*) In the book on page 107, the top paragraph describes a procedure for converting DFA to CFGs:

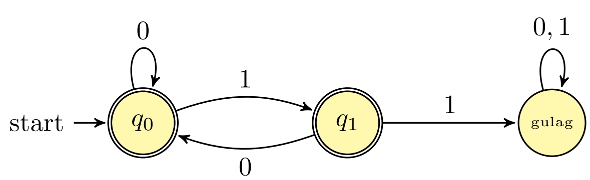
> You can convert any DFA into an equivalent CFG as follows. Make a variable R_i for each state q_i of the DFA. Add the rule $R_i \to aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in the DFA. Add the rule $R_i \to \varepsilon$ if q_i is an accept state of the DFA. Make R_0 the start variable ofthe grammar, where q_0 is the start state of the machine. Verify on your own that the resulting CFG generates the same language that the DFA recognizes.

Use this construction to get a context-free grammar generating the language

$$\{w \in \{0,1\}^* \mid w \text{ does not have 11 as a substring}\}$$

by (1) designing a DFA that recognizes this language and then (2) applying the construction from the book to convert the DFA to an equivalent CFG. A complete submission will include the state diagram of the DFA, a brief justification of why it recognizes the language, and then the complete and precise definition of the CFG that results from applying the construction from the book to this DFA. *Ungraded bonus: take a sample string in the language and see how the computation of the DFA on this string translates to a derivation in your grammar.*

Proof. Consider the following state diagram of a DFA M:



Call that language S. Note that the states q_0 and q_1 record the previous input character. Suppose $s \in S$. Since s does not contain consecutive 1's, s would never get sent from q_1 to the gulag, and thus $s \in L(M)$. Suppose $s \notin S$. Then, s contains consecutive 1's, and so it would eventually get sent to the gulag. Hence, $L(M) = S$. Applying the construction to M, we get a CFG $G = (\{R_0, R_1, R_{gulag}\}, \{0,1\}, R, R_0)$, where the set of rules R is

$$R_0 \to 0R_0 \mid 1R_1 \mid \varepsilon$$
$$R_1 \to 0R_0 \mid 1R_{gulag} \mid \varepsilon$$
$$R_{gulag} \to 0R_{gulag} \mid 1R_{gulag}.$$

$\square$

Problem 3

On page 4 of the week 4 notes, we have the following list of languages over the alphabet $\{a, b\}$

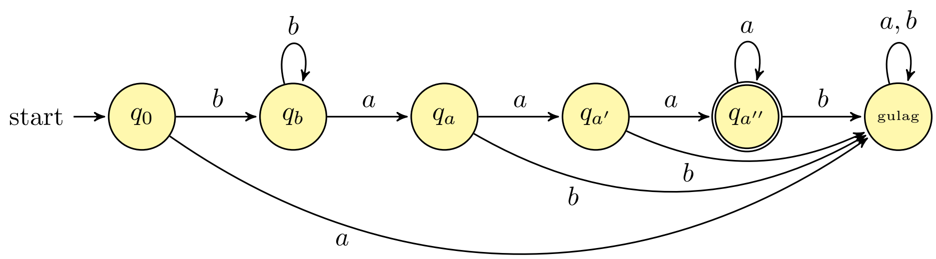
$$\{a^n b^n \mid 0 \leq n \leq 5\} \quad \{b^n a^n \mid n \geq 2\} \quad \{a^m b^n \mid 0 \leq m \leq n\}$$
$$\{a^m b^n \mid m \geq n+3, n \geq 0\} \quad \{b^m a^n \mid m \geq 1, n \geq 3\}$$
$$\{w \in \{a,b\}^* \mid w = w^{\mathcal{R}}\} \quad \{ww^{\mathcal{R}} \mid w \in \{a,b\}^*\}$$

(a) (*Graded for completeness*) Pick one of the regular languages and design a regular expression that describes it. Briefly justify your regular expression by connecting the subexpressions of it to the intended language and referencing relevant definitions.

Proof. Consider $S = \{a^n b^n \mid 0 \leq n \leq 5\}$. $\varepsilon \cup ab \cup aabb \cup aaabbb \cup aaaabbbb \cup aaaaabbbbb$ describes S, as it is the union of all strings in S. □

(b) (*Graded for completeness*) Pick another one of the regular languages and design a DFA that recognizes it. Draw the state diagram of your DFA. Briefly justify your design by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

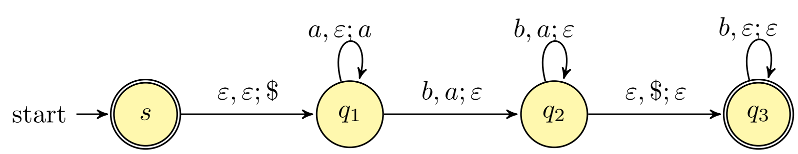
Proof. Consider the following DFA M that recognizes $T = \{b^m a^n \mid m \geq 1, n \geq 3\}$:



The states count the number of times we have met both a and b, to ensure the input strings are of the form $bb \ldots baa \ldots a$ and have at least 1 b and 3 a's. □

(c) (*Graded for completeness*) Pick one of the nonregular languages and design a PDA that recognizes it. Draw the state diagram of your PDA. Briefly justify your design by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Proof. Consider the set $X = \{a^m b^n \mid 0 \leq m \leq n\}$. The following PDA recognizes X:



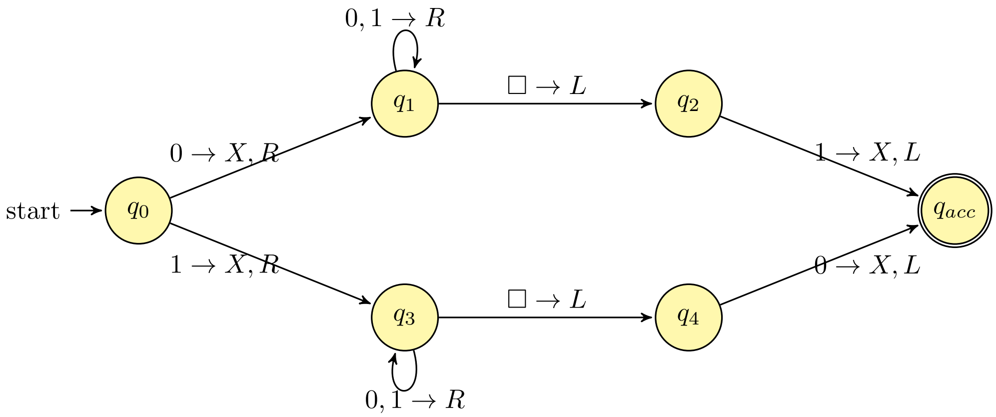
The PDA stores the number of a's in the prefix on the memory stack, and ensures the strings have enough b's to deplete the stack before reaching the accepting state q_3. Hence, X is recognized by the PDA above. □

(d) (*Graded for completeness*) Pick one of the nonregular languages and write a CFG that generates it. Briefly justify your design by by demonstrating how derivations in the grammar relate to the intended language.

Proof. Consider $Y = \{ww^{\mathcal{R}} \mid w \in \{a,b\}^*\}$. The CFG $G = (\{S\}, \{a,b\}, \{S \to aSa \mid bSb \mid \varepsilon\}, S)$ generates Y. Since every rule in R produces the same elements on both sides of S, G inductively generates symmtric strings over $\{a,b\}$, and this is exactly what Y is. $\square$

Problem 4

Consider the Turing machine T over the input alphabet $\Sigma = \{0, 1\}$ with the state diagram below (the tape alphabet is $\Gamma = \{0, 1, X, \square\}$). Convention: we do not include the node for the reject state q_{rej} and any missing transitions in the state diagram have value $(q_{rej}, \square, R)$



(a) (*Graded for correctness*) Specify an example string w_1 of length 4 over Σ that is **accepted** by this Turing machine, or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example string and a precise and clear description of the accepting computation of the Turing machine on this string or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

To describe a computation of a Turing machine, include the contents of the tape, the state of the machine, and the location of the read/write head at each step in the computation.

Hint: In class we've drawn pictures to represent the configuration of the machine at each step in a computation. You may do so or you may choose to describe these configurations in words.

Proof. Consider $w_1 = 0001$. The following is the computation of w_1 in T:

$q_0 \downarrow$				
0	0	0	1	$\square$
	$q_1 \downarrow$			
X	0	0	1	$\square$
		$q_1 \downarrow$		
X	0	0	1	$\square$
			$q_1 \downarrow$	
X	0	0	1	$\square$
				$q_1 \downarrow$
X	0	0	1	$\square$
			$q_2 \downarrow$	
X	0	0	1	$\square$
		$q_{acc} \downarrow$		
X	0	0	X	$\square$

Therefore, T accepts w_1. $\square$

Problem 4 continued on next page...

(b) (*Graded for correctness*) Specify an example string w_2 of length 3 over Σ that is **rejected** by this Turing machine or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example string and a precise and clear description of the rejecting computation of the Turing machine on this string or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

Proof. Consider $w_2 = 000$. The following is the computation of w_1 in T:

$q_0 \downarrow$			
0	0	0	$\square$
	$q_1 \downarrow$		
X	0	0	$\square$
		$q_1 \downarrow$	
X	0	0	$\square$
			$q_1 \downarrow$
X	0	0	$\square$
		$q_2 \downarrow$	
X	0	0	$\square$
			$q_{rej} \downarrow$
X	0	$\square$	$\square$

Therefore, T rejects w_2. $\square$

(c) (*Graded for correctness*) Specify an example string w_3 of length 2 over Σ on which the computation of this Turing machine **loops** or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your example string and a precise and clear description of the looping (non-halting) computation of the Turing machine on this string or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

Proof. Such w_3 does not exist. In the computation of T, T keeps reading the next right element until it reads a blank. After T reads a blank, T proceeds to either q_2 or q_4. Since w_3 is of a finite length, T will eventually read a blank, and thus it must reach either q_2 or q_4. However, q_2 and q_4 only points to either the accepting or rejecting state, and thus T must halt if the input is a string of length 2. $\square$

(d) (*Graded for completeness*) Write an implementation level description of the Turing machine T.

Proof. T starts in q_0 processes the input by marking visited symbols with an X and moving the tape head to the right. If the first input characted T reads is 0, then it transitions to q_1. Otherwise, if the first input character is 0, then it transitions to q_3. Afterwards, T repeatedly reads the next right character until reading a $\square$, which indicates that the string has ended. It then transitions to q_2/q_4, prints a $\square$, and reads the left character, namely the last character of the string. If the ending character of the string is different from the starting character, then the machine transitions to the rejecting state. Otherwise, T transitions to the accepting state. transitions to $\square$