HW3CSE105W24: Homework Assignment 3

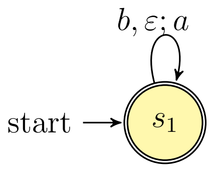
CSE105W24

February 21, 2024

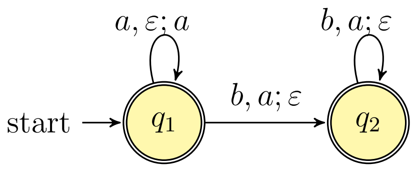
1. **Constructions** (18 points):

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

State diagram for M_1



State diagram for M_2



(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.

Solution:
M_1 recognizes the language $A_1 = \{b^n \mid n \geq 0\}$. A sample string that is **accepted** is ϵ, since the start state is also an accept state. A sample string that is **rejected** is a. Since we do not have a transition that lets us read the symbol a, the string is not accepted by the PDA.
M_2 recognizes the language $A_2 = \{a^n b^m \mid n \geq m \geq 0\}$. A sample string that is **accepted** is aab. For each of the a's read from the input, we push an a onto the stack from q_1. When the b is read, we move to q_2, popping the second a from the stack. We don't push anything during this transition. Since we have finished reading the string and are in an accept state, we accept the string.
A sample string that is **rejected** by M_2 is abb. When we read the a, we push an a onto the stack. When we read the first b, we pop an a from the stack and move to q_2. The stack is now empty. Now, when we read the second b, it will try to pop an a, but the top of the stack is empty. So, we are stuck and the machine will not accept abb.

(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 $i = 1, 2$).

Solution:

$G_1 = (\{S\}, \{a, b\}, \{S \to bS \mid \varepsilon\}, S)$

We see from above that $A_1 = \{b^n \mid n \geq 0\}$. Take any $s = b^n \in A_1$, we can use the rule $S \to bS$ n times to get b^nS and then $S \to \varepsilon$ one time to get b^n, exactly what we want (shows $A_1 \subseteq L(G_1)$. On the other hand, pick a string that is in the language of G_1, we see that the rule is such that we can only add zero or more $b's$ to a string. This means that the string is of the form b^n for some n as required (shows $L(G_1) \subseteq A_1$).

$G_2 = (\{T\}, \{a, b\}, \{T \to aT \mid aTb \mid \varepsilon\}, T)$

A_2 contains all strings of the form a^nb^m with $n \geq m \geq 0$. Any such string can be derived by G_2 by using the rule $T \to aT$ $n - m$ times to get $a^{n-m}T$ and then $T \to aTb$ m times to get a^nb^mT and finally $T \to \varepsilon$ to get a^nb^m. On the other hand, we see that every time we use a rule in G_2, we can only add $a's$ before $b's$. Moreover, we can only add equal or more $a's$ than $b's$. Every such string that can be derived is of the form a^nb^m such that $n \geq m \geq 0$, which is in A_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

$$G = (V_1 \cup V_2 \cup \{S_{new}\}, \Sigma, R_1 \cup R_2 \cup \{S_{new} \to S_1S_2\}, S_{new})$$

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.

Solution:

$G = (\{S, T, S_{new}\}, \{a, b\}, \{S_{new} \to S_1S_2, S \to bS \mid \varepsilon, T \to aT \mid aTb \mid \varepsilon\}, S_{new})$

Consider the string $bab \in A_1 \circ A_2$ where b comes from A_1 and ab comes from A_2.

$$S_{new} \to ST \to bST \to bT \to baTb \to bab$$

(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

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

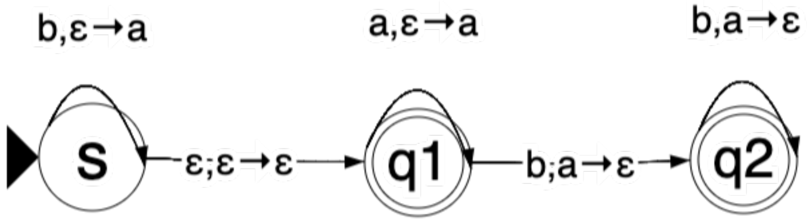
with $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ given by

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

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.

Solution:

Result of the above construction on PDAs M_1 and M_2 to create PDA M:



Consider the string *babb* that is accepted by M but not in $A_1 \circ A_2$.

Acceptance by M: Initially the stack is empty, M will read the first b, so M will follow the transition $b;\epsilon;a$ on state s to read a b, pop nothing and push an a onto the stack, so M will loop on state s. Then M will spontaneously move to q_1, pushing and popping nothing. At q_1, M reads in an a, so M will follow the transition $a;\epsilon;a$ to read an a, pop nothing and push an a onto the stack and M stays at q_1. Now there are two a's in the stack. Then, the second b is read, and there is an a at the top of the stack to be popped, so M follows the transition $b;a;\epsilon$ to q_2 pops an a off the stack and push in nothing, and move to state q_2. Now, the stack have one a left inside. At q_2, M reads the third and last b. There is an a at the top of the stack to be popped, so M follows the transition that loops on q_2 to read a b, pop an a and push in nothing. Now, M is done reading the string and M is in q_2, an accept state, so M accepts *babb*.

Not in $A_1 \circ A_2$: From part a), we know that $A_1 = \{b^n \mid n \geq 0\}$ and $A_2 = \{a^n b^m \mid n \geq m \geq 0\}$. The definition of set-wise concatenation for A_1 and A_2 is $A_1 \circ A_2 = \{w \in \Sigma^* \mid w = a_1 a_2$ for some strings $a_1 \in A_1$ and $a_2 \in A_2\}$. Consider all the ways that we can break *babb* into strings a_1 and a_2 such that $a_1 \circ a_2 = babb$. Here, we will show that no matter how we break it up, there is no possible way for $a_1 \in A_1$ **and** $a_2 \in A_2$.

i. $a_1 = \epsilon$, $a_2 = babb$. Clearly, $a_2 \notin A_2$ since there cannot be a b before an a.
ii. $a_1 = b, a_2 = abb$. $a_2 \notin A_2$ since the number of b's are greater than the number of a's ($1 \ngeq 2$).
iii. $a_1 = ba, a_2 = bb$. $a_1 \notin A_1$ since A_1 contains only strings of b's. $a_2 \notin A_2$ since the number of b's are greater than the number of a's ($0 \ngeq 2$).
iv. $a_1 = bab, a_2 = b$. $a_1 \notin A_1$ since A_1 contains only strings of b's. $a_2 \notin A_2$ since the number of b's are greater than the number of a's ($0 \ngeq 1$).
v. $a_1 = babb, a_2 = \epsilon$. $a_1 \notin A_1$ since A_1 contains only strings of b's.

Since we have shown that there exist a string for which M accepts but yet is not in $A_1 \circ A_2$, this construction **cannot** be used to show that the class of context-free languages is closed under set-wise concatenation.

2. **Regular languages are context-free** (10 points):

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 paramaters 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.

Solution:
Given a NFA $N = (Q, \Sigma, \delta_N, q_0, F)$ we define a PDA M with $L(M) = L(N)$ by letting $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where $\Gamma = \Sigma$, $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ given by

$$\delta(q, a, b) = \begin{cases} \delta_N(q, a) \times \{\varepsilon\} & \text{if } q \in Q,\ a \in \Sigma_\varepsilon,\ b = \varepsilon \\ \emptyset & \text{otherwise} \end{cases}$$

(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

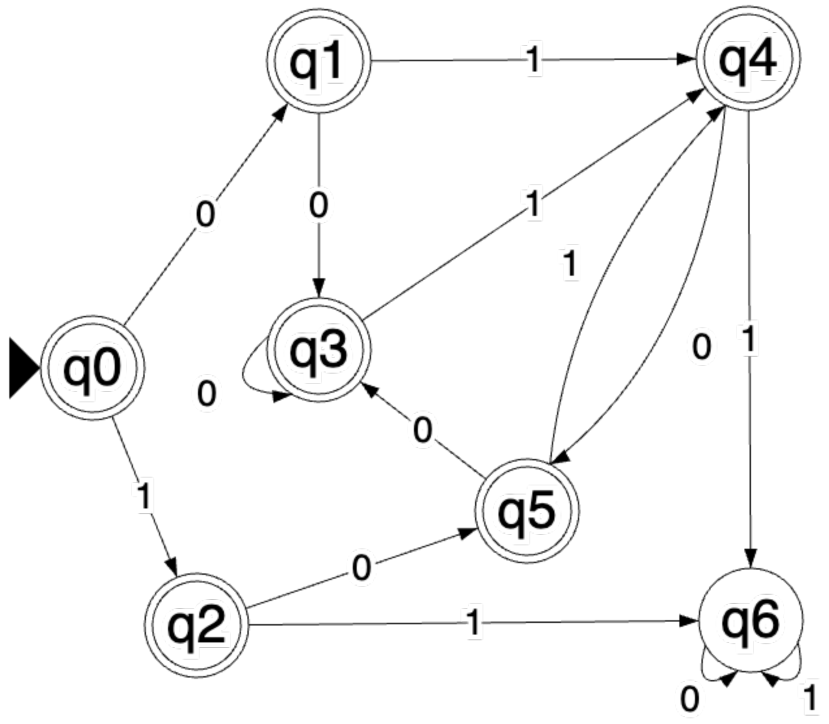
$$\{w \in \{0,1\}^* \mid w \text{ does not have } 11 \text{ as a substring}\}$$

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.*

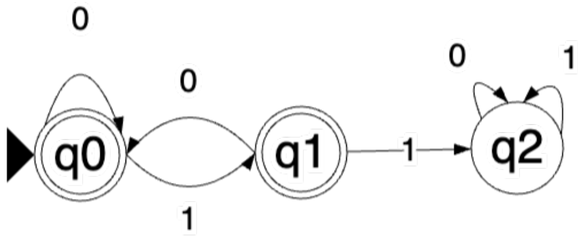
Solution:

The construction below is done by having 7 states for all 7 binary strings shorter than length 3. q_0 is for ε, q_1 for 0, q_2 for 1, q_3 for 00, ...

For the transitions, we make sure that we would transition to the state representing the binary string for the last two bits we read. For example, at q_4, the last two bits we have read are 01. If we read a 0, the last two bits we read would be 10 and so we transition to q_5.



Another way to construct a DFA recognizing the language is to first construct a DFA recognizing all strings with 11 as a substring and then flipping the accept/reject.



Applying the construction, we get $(\{R_0, R_1, R_2\}, \{0,1\}, R, R_0)$ where R is as follows

$$\begin{aligned}
R_0 &\to 0R_0 \mid 1R_1 \mid \varepsilon \\
R_1 &\to 0R_0 \mid 1R_2 \mid \varepsilon \\
R_2 &\to 0R_2 \mid 1R_2
\end{aligned}$$

3. **Classifying languages** (12 points): On page 4 of the week 4 notes, we have the following list of languages over the alphabet $\{a, b\}$

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

Solution:

Regular languages:

- $\{a^n b^n \mid 0 \leq n \leq 5\}$
- $\{b^m a^n \mid m \geq 1, n \geq 3\}$

Nonregular languages:

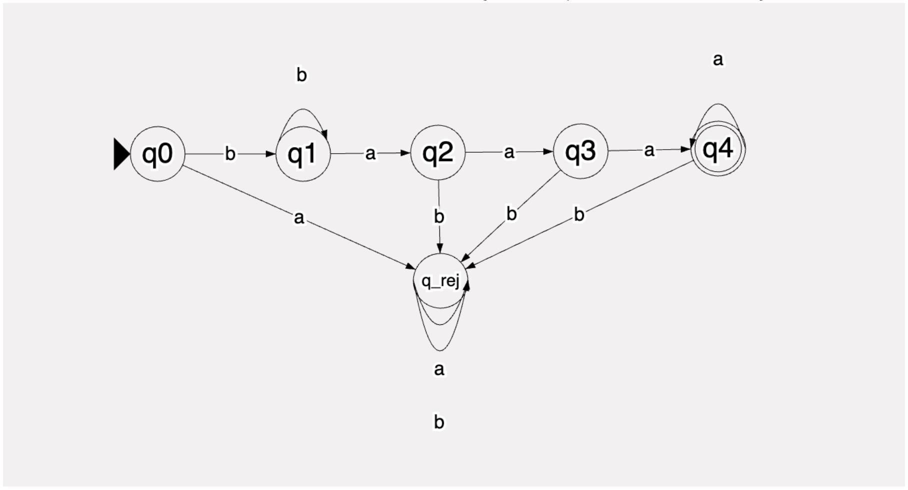
- $\{b^n a^n \mid n \geq 2\}$
- $\{a^m b^n \mid 0 \leq m \leq n\}$
- $\{a^m b^n \mid m \geq n+3, n \geq 0\}$
- $\{w \in \{a,b\}^* \mid w = w^{\mathcal{R}}\}$
- $\{ww^{\mathcal{R}} \mid w \in \{a,b\}^*\}$

(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.

Solution:
$L(bb^*aaaa^*) = \{b^m a^n \mid m \geq 1, n \geq 3\}$
The first b is there to indicate that there must be at least 1 b that starts the strings in this language. b^* is next to show that there could be more b's after that. Together, they form the $b^m, m \geq 1$ part of the strings in this language. Similarly, 3 a's are next to show that there must be 3 a's that follow the b's. Connecting that with a^* will give us the $a^n, n \geq 3$ part of the strings of this language.
OR
$L(\epsilon \cup ab \cup aabb \cup aaabbb \cup aaaabbbb \cup aaaaabbbbb) = \{a^n b^n \mid 0 \leq n \leq 5\}$.
This regular expression captures all of the possible elements of $\{a^n b^n \mid 0 \leq n \leq 5\}$ by listing all of the possible elements and unioning them.

(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.

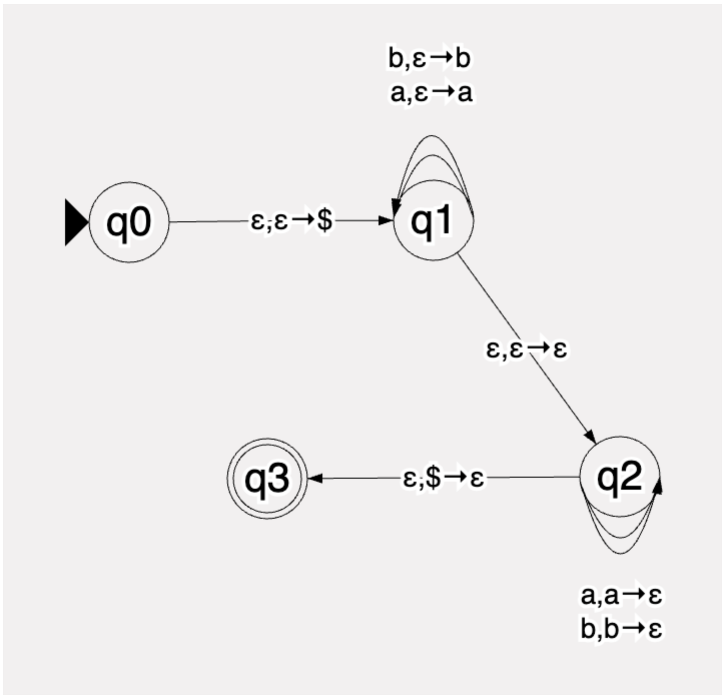
Solution: Consider the language $\{b^m a^n \mid m \geq 1, n \geq 3\}$



$q0$ is the start state. The role of $q1$ is to check that there has been *at least* 1 b. $q2, q3, q4$ are there to check that there has been 1, 2, 3 a's respectively, so $q4$ is the accept state. $q4$ has a self loop on a since it can accept *at least* 3 a's. If any string does not fit into this computation, the computation on it lands in the q_rej state.

(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.

Solution: Consider the language $\{ww^{\mathcal{R}} \mid w \in \{a, b\}^*\}$



$q0, q3$ are there to check for stack emptiness. $q1, q2$ are there to check any possible way the input string can be split into a string w and its reverse $w^{\mathcal{R}}$. Each string starts off at $q0$, and enters $q1$ with a \$ pushed onto the stack at the beginning. As characters are being processed in $q1$, the computation can make a nondeterministic jump to $q2$ to start processing the $w^{\mathcal{R}}$ part. If it does belong in the language, it will reach the bottom of the stack with no characters left to process, and nothing in the stack except \$. At that point, it will enter $q3$ and get accepted. If it's not in this language, it will get stuck in $q2$.

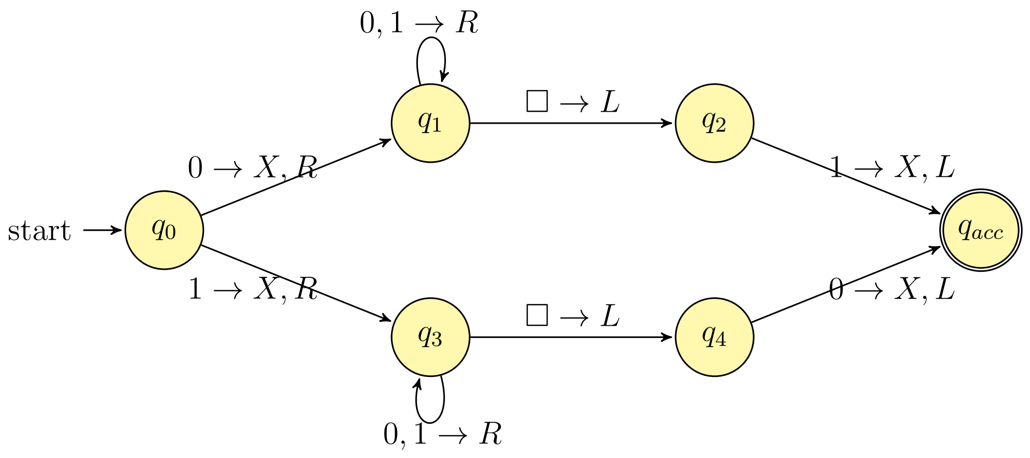
(d) (*Graded for completeness*) Pick one of the nonregular languages and write a CFG that generates it. Briefly justify your design by by demonstrating how derivations in the grammar relate to the intended language.

Solution:

Pick $\{ww^R \mid w \in \{a,b\}^*\}$. Consider the CFG $(\{S\}, \{a,b\}, \{S \to aSa \mid bSb \mid \varepsilon\}, S)$

We see from above that we always add the same character to the back as we add to the front. Intuitively that is how the CFG makes sure that the string will be an even palindrome. Now, consider aabbaa, this string can be derived from $S \to aSa \to aaSaa \to aabSbaa \to aabbaa$.

4. **Turing machines** (10 points): 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 $qrej$ and any missing transitions in the state diagram have value $(qrej, \square, R)$



(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.

Solution:

Consider the string $w_1 = 0101$ that is accepted by T.

Accepting computation of w_1:

Note: We will use the notations from page 168-169 of the textbook to represent the computation of T on w_1. To briefly explain it, the character to the right of the state is where the tapehead is currently pointing to. If there is no character, the tapehead is pointing at a blank cell.

1. $q_0 0101$
2. $X q_1 101$
3. $X1 q_1 01$
4. $X10 q_1 1$
5. $X101 q_1$

6. $X10q_21$
7. $X10Xq_{acc}$

Since the computation reached the accepting state, T halts and accepts $w_1 = 0101$.

(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.

Solution:

Consider the string $w_2 = 000$ that is rejected by T.

Rejecting computation of w_2:

1. q_0000
2. Xq_100
3. $X0q_10$
4. $X00q_1$
5. $X0q_20$
6. $X00q_{rej}$

Since the computation reached the rejecting state (by convention missing transitions go to q_{rej}), T halts and rejects $w_2 = 000$.

(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.

Solution:

No such example exists. There are only 4 strings of length 2 over Σ, namely $\{00, 01, 10, 11\}$. We just need to show none of them causes the machine to loop (run forever). WLOG, consider 00. The TM reads the first 0 and changes it to X before moving to q_1. There, it replaces the 0 with another 0 and moves right to point at an empty cell. At this point, we transition to q_2 and move the tape head left pointing at the last 0. Because there is not a valid transition at q_2 reading 0, we reject. Similarly, all the rest three strings either get accepted or rejected.

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

Solution:

Replace the first non-blank character with X. Scan right until the end of the string, at which point a blank is read, then move left. If the character replaced with an X in the beginning was a 0 or a 1, replace the last character, a 1 or a 0 respectively, with X, then accept the input. If at least one of the above action could not be performed due to no first character, first and last character are not opposites, etc, reject the input.