HW4CSE105F24: Homework assignment 4 solution

CSE105F24

Due: November 12, 2024 at 5pm, via Gradescope

In this assignment,

You will work with context-free languages and their representations. You will also practice analyzing, designing, and working with Turing machines. You will use general constructions and specific machines to explore the classes of recognizable and decidable languages.

Resources: To review the topics for this assignment, see the class material from Weeks 4, 5, and 6. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Chapters 2 and 3. Chapter 2 exercises 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.9, 2.10, 2.11, 2.12, 2.13, 2.16, 2.17. Chapter 3 exercises 3.1, 3.2, 3.5, 3.8.

Assigned questions

1. **Push-down automata (PDA) and context-free grammars (CFG)** (8 points): On page 14 of the week 3 notes, we have the following list of languages over the alphabet $\{a, b\}$

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

(a) (*Graded for completeness*) [1] 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.

[1] This means you will get full credit so long as your submission demonstrates honest effort to answer the question. You will not be penalized for incorrect answers. To demonstrate your honest effort in answering the question, we expect you to include your attempt to answer *each* part of the question. If you get stuck with your attempt, you can still demonstrate your effort by explaining where you got stuck and what you did to try to get unstuck.

Solution: Consider the language

$$L = \{a^n b^n \mid 0 \leq n \leq 5\}$$

Notice that there is a bound on n. Because of this and the form of the strings in the language, we can enumerate this language, and notice that it is finite.

$$L = \{\varepsilon,\ ab,\ aabb,\ aaabbb,\ aaaabbbb,\ aaaaabbbbb\}$$

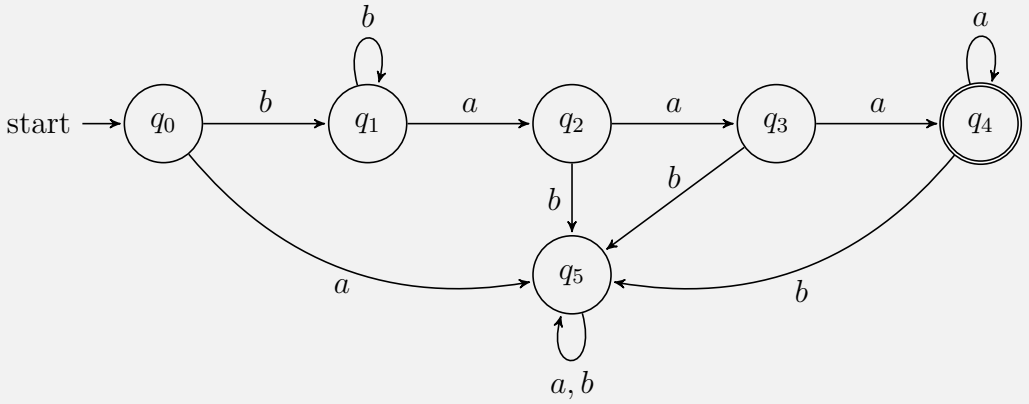
We can use regular expressions to describe this language. The following regular expression R describes L:

$$R = \varepsilon \cup ab \cup aabb \cup aaabbb \cup aaaabbbb \cup aaaaabbbbb$$

Each subexpression is one particular string in L, and since L is finite, it's possible to write out all the strings, consider each one of them as a regular expression, and putting the union symbol between them gives and expression that describes the union of the six singleton sets together.

(b) (*Graded for completeness*) Pick another one of the regular languages and design a deterministic finite automaton (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:



The above DFA recognizes the language

$$L(bb^*aaaa^*) = L = \{b^m a^n \mid m \geq 1, n \geq 3\}$$

Each state corresponds to a subexpression of the regular expression above in the following way:

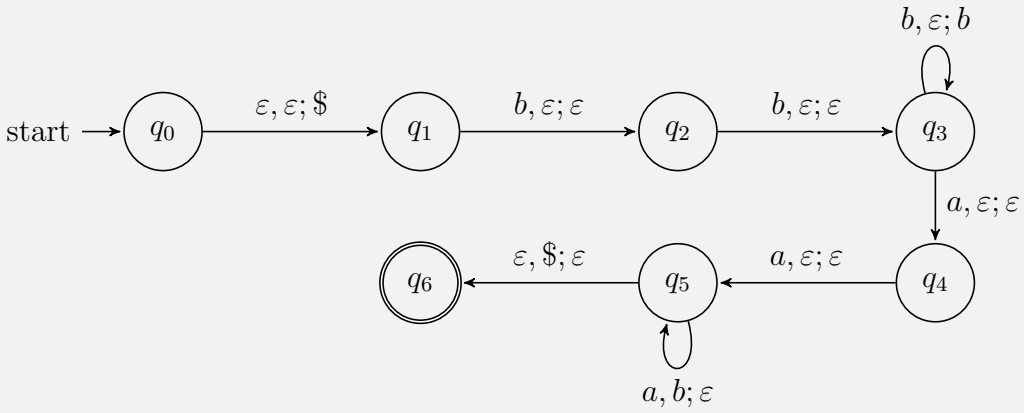
0. q_0 is the start state, "seen nothing".

1. q_1 "seen one or more b."
2. q_2 "seen one or more b followed by one a."
3. q_3 "seen one or more b followed by two a's."
4. q_4 "seen one or more b followed by three or more a's."
5. q_5 is the sink state, "seen some character that breaks the pattern".

This DFA would process the string from start to finish, starting from q_0. An accepting computation would run from q_0 to q_4, possible looping at q_1 and/or q_4. A rejecting computation could end in any of the other non-accept states. If it ends in q_5, then the string breaks the pattern in the language. Otherwise, the computation ends in q_0, q_1, q_2 or q_3 so the string is missing some ending characters to be accepted.

(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: Let's choose the language $\{b^n a^n \mid n \geq 2\}$. The state diagram of the PDA that recognizes the language is the picture as follows.



- q_0: Start state
- q_1: At this state, our stack has \$, which marks the bottom of the stack.
- q_2: At this state, we've seen the first b. Note that there must be at least two bs in a row for a string this language because $n \geq 2$.
- q_3: At this state, we've seen two or more bs. After seeing the first two bs, any additional bs in the string will mean a b is pushed to the stack as a counter.

- $q4$: At this state, we've seen the first a after the bs. Note that at least two as must be in the string for the same reason there are at least two bs.
- $q5$: At this state, we've seen two or more as after the bs. After seeing the first two as, any additional as will result in popping a b off of the stack unless there are no more bs to pop.
- $q6$: At this state, we accept a string if there are no more characters left to read. Note that we intended to reach an empty stack to ensure that the number of bs and as are the same.

If a string is accepted by this PDA, it means that there are at least two bs followed by at least two as, and the number of as is the same as the number of bs. We made sure there are least two of each character, which means the condition $n \geq 2$ is satisfied. And only when the number of as is equal to the number of bs, we know we have reached the bottom of the stack and can accept, which is exactly what our language specifies.

On the other hand, if a string is rejected by the machine, then that means that the string either does not have the same amount of each character in that order or doesn't have at least two of each character. They will be "rejected" because we either can't read the whole string, or because we cannot land in an accept state. Therefore, all strings rejected by the machine are not in the language.

(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: Consider the language

$$\{ww^{\mathcal{R}} \mid w \in \{a, b\}^*\}$$

A CFG that generates it is defined as

$$(\{S\}, \{a, b\}, \{S \to 0S0 \mid 1S1 \mid \varepsilon\}, S)$$

Starting from the start variable S, each step in the derivations in the grammar can substitute one S with $0S0$, $1S1$, or ε, i.e. we can add one 0 on both sides of S, or one 1 on both sides of S, or terminate the derivation. Therefore, all strings in the language derived by the CFG will be palindromes of even length.

2. **General constructions for context-free languages** (21 points):

In class in weeks 4 and 5, we described several general constructions with PDAs and CFGs, leaving their details to homework. In this question, we'll fill in these details. The first constructions help

us prove that the class of regular languages is a subset of the class of context-free languages. The other construction allows us to make simplifying assumptions about PDAs recognizing languages.

(a) (*Graded for correctness*) [2] When we first introduced PDAs we observed that any NFA can be transformed to a PDA by not using the stack of the PDA at all. Suppose a friend gives you the following construction to formalize this transformation:

> Given a NFA $N = (Q, \Sigma, \delta_N, q_0, F)$ we define a PDA M with $L(M) = L(N)$ by letting $M = (Q, \Sigma, \Sigma, \delta, q_0, F)$ where $\delta(\ (q, a, b)\ ) = \delta_N(\ (q, a)\ )$ for each $q \in Q$, $a \in \Sigma_\varepsilon$ and $b \in \Sigma_\varepsilon$.

For each of the six defining parameters for the PDA, explain whether it's defined correctly or not. If it is not defined correctly, explain why not and give a new definition for this parameter that corrects the mistake.

Solution: The set of states, input alphabet, stack alphabet, start state, and the set of accept states are defined correctly. However, the transition function is incorrect, because the output type is incorrect. For the output of our PDA transit function, we are expecting an element from $\mathcal{P}(Q \times \Gamma_\varepsilon)$ (where $\Gamma = \Sigma$), but now the output would be an element from $\mathcal{P}(Q)$. A correct definition would be:

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

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

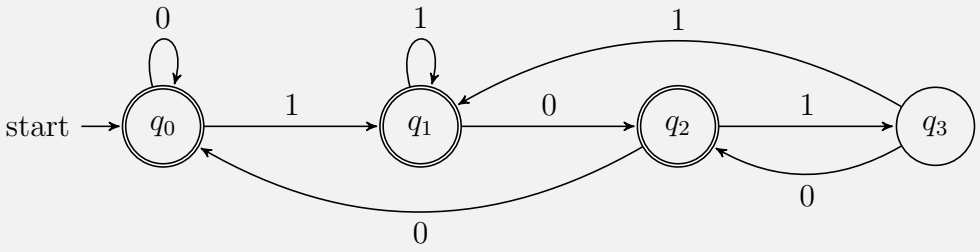
$$\{w \in \{0, 1\}^* \mid w \text{ does not end in } 101\}$$

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 and correct 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

[2] This means your solution will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

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:



This state diagram works by keeping track of the last 3 digits it has read. If the last three digits it's read are in the order 101 then the computation of the string will end up in q_3 which is not an accepting state, which means that the string is rejected.

We can create a CFG defined as $(\{R_0, R_1, R_2, R_3\}, \{0, 1\}, T, R_0)$ where T is a set of rules defined as:

$$R_0 \to 0R_0 \mid 1R_1 \mid \varepsilon$$
$$R_1 \to 0R_2 \mid 1R_1 \mid \varepsilon$$
$$R_2 \to 0R_0 \mid 1R_3 \mid \varepsilon$$
$$R_3 \to 0R_2 \mid 1R_1$$

Ungraded: Let's use the string 010 as an example. The computation of the DFA starts at the start state q_0. Likewise, the derivation also starts with the start variable (which is the corresponding variable R_0). When we read a 0, we're right where we started at q_0, correlating with the rule $R_0 \to 0R_0$ and the first step in derivation is $R_0 \Rightarrow 0R_0$. When we read a 1, we transition to q_1, correlating with the rule $R_0 \to 1R_1$, so we take another step in derivation as $R_0 \Rightarrow 0R_0 \Rightarrow 01R_1$. Finally, when we read a 0, we transition to q_2. Likewise, our CFG tells us we can use the rule $R_1 \to 0R_2$, so our derivation becomes $R_0 \Rightarrow 0R_0 \Rightarrow 01R_1 \Rightarrow 010R_2$. Finally, we have no more characters to read and because we are in an accept state, we know the string 010 is accepted by this DFA, that is to say, in the language. In our CFG, we can use the rule $R_2 \to \varepsilon$ so the derivation $R_0 \Rightarrow 0R_0 \Rightarrow 01R_1 \Rightarrow 010R_2 \Rightarrow 010\varepsilon$ ends with string 010 which has no variable, and thus must be in the language this CFG generates.

(c) Let $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F_1)$ be a PDA and let $q_{new}, r_{new}, s_{new}$ be three fresh state labels (i.e. $Q_1 \cap \{q_{new}, r_{new}, s_{new}\} = \emptyset$) and let # be a fresh stack symbol (i.e. $\# \notin \Gamma_1$). We define the PDA M_2 as

$$(Q_2, \Sigma, \Gamma_2, \delta_2, q_{new}, \{s_{new}\})$$

with $Q_2 = Q_1 \cup \{q_{new}, r_{new}, s_{new}\}$ and $\Gamma_2 = \Gamma_1 \cup \{\#\}$ and $\delta_2 : Q_2 \times \Sigma_\varepsilon \times \Gamma_{2\varepsilon} \to \mathcal{P}(Q_2 \times \Gamma_{2\varepsilon})$

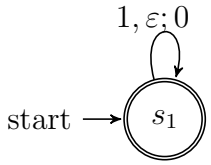
given by

$$\delta_2(\ (q,a,b)\ ) = \begin{cases} \{(q_1,\#)\} & \text{if } q = q_{new}, a = \varepsilon, b = \varepsilon \\ \delta_1(\ (q,a,b)\ ) & \text{if } q \in Q_1 \setminus F_1, a \in \Sigma_\varepsilon, b \in \Gamma_{1\varepsilon} \\ \delta_1(\ (q,a,b)\ ) & \text{if } q \in F_1, a \in \Sigma, b \in \Gamma_{1\varepsilon} \\ \delta_1(\ (q,a,b)\ ) & \text{if } q \in F_1, a = \varepsilon, b \in \Gamma_1 \\ \delta_1(\ (q,a,b)\ ) \cup \{(r_{new},\varepsilon)\} & \text{if } q \in F_1, a = \varepsilon, b = \varepsilon \\ \{(r_{new},\varepsilon)\} & \text{if } q = r_{new}, a = \varepsilon, b \in \Gamma_1 \\ \{(s_{new},\varepsilon)\} & \text{if } q = r_{new}, a = \varepsilon, b = \# \\ \emptyset & \text{otherwise} \end{cases}$$

for each $q \in Q_2$, $a \in \Sigma_\varepsilon$, and $b \in \Gamma_{2\varepsilon}$.

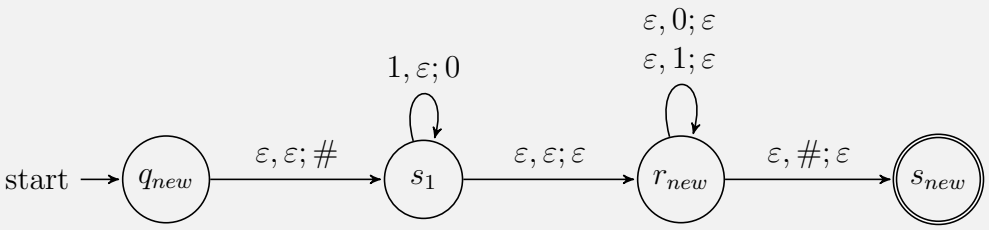
In this question, we'll apply this construction for a specific PDA and use this example to extrapolate the effect of this construction.

i. (*Graded for correctness*) Consider the PDA M_1 with input alphabet $\{0,1\}$ and stack alphabet $\{0,1\}$ whose state diagram is



Draw the state diagram for the PDA M_2 that results from applying the construction to M_1.

Solution: The state diagram of the PDA M_2 is shown below, where input alphabet is $\{0,1\}$ and stack alphabet is $\{0,1,\#\}$:

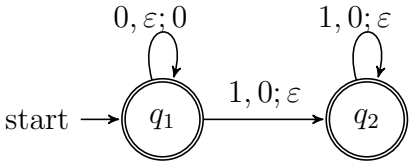


ii. (*Graded for completeness*) Compare $L(M_1)$ and $L(M_2)$. Are these sets equal? Does your answer depend on the specific choice of M_1? Why or why not?

Solution: $L(M_1)$ and $L(M_2)$ are equal. Here, $L(M_1) = L(M_2) = \{1^k \mid k \geq 0\}$. More generally, no matter how we choose M_1, we get $L(M_1) = L(M_2)$. Loosely speaking, what M_2 does is to follow M_1's computation and "empty the stack" before a string is accepted. More specifically, for any $s \in L(M_1)$, there exists a computation of M_1 on s that processes the whole string and ends in one of

the accept states in M_1, and thus there exists a computation of M_2 on s that processes the whole string and reaches r_{new}. In r_{new}, the stack content can be popped until there is only one # symbol at the bottom of the stack that was pushed when transitioning from q_{new} to the original start state of M_1. Then, from r_{new}, we can pop out the # from stack and end the computation in s_{new} which is the accept state of M_2. Therefore, s will be accepted by M_2. This shows that every string accepted by M_1 is also accepted by M_2. On the other hand, for any string $t \notin L(M_1)$, there does not exist a computation of M_1 on t that processes the whole string and ends in one of the accept states in M_1, and thus there does not exist a computation of M_2 on t that can reach r_{new} and thus ends in s_{new} which is the only accept state of M_2. Therefore, t will be rejected by M_2. This shows that every string rejected by M_1 will also be rejected by M_2. Therefore, $L(M_1) = L(M_2)$ no matter what M_1 we choose.

iii. (*Graded for completeness*) Consider the PDA N with input alphabet $\{0, 1\}$ and stack alphabet $\{0, 1\}$ whose state diagram is



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

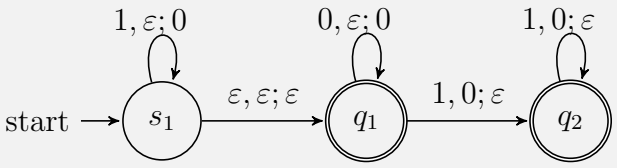
$$L_1 \circ L_2 = \{w \in \Sigma^* \mid w = uv \text{ for some strings } u \in L_1 \text{ and } v \in L_2\}$$

In class, we discussed how extrapolating the construction that we used to prove that the class of regular languages is closed under set-wise concategation by drawing spontaneous transitions from the accepting states in the first machine to the start state of the second machine doesn't work. Use the example of M_1 and N_1 to prove this by showing that

$$L(M_1) \circ L(N)$$

is **not** the language recognized by the machine results from taking the two machines M_1 and N, setting the start state of M_1 to be the start state of the new machine, setting the set of accepting states of N to be the set of accepting states of the new machine, and drawing spontaneous arrows from the accepting states of M_1 to the start state of N.

Solution: The state diagram of the new machine P we get following the construction is the following, where the input alphabet is $\{0, 1\}$ and the stack alphabet is $\{0, 1\}$:



We can get that $L(M_1) = \{1^k \mid k \geq 0\}$ and $L(N) = \{0^m 1^n \mid m \geq n \geq 0\}$. Therefore, $L(M_1) \circ L(N) = \{1^k 0^m 1^n \mid k \geq 0,\ m \geq n \geq 0\}$. However, $L(P) = \{1^k 0^m 1^n \mid k \geq 0,\ m \geq 0,\ 0 \leq n \leq k + m\}$. Consider the string 1011 that is accepted by P. However, $1011 \notin L(M_1) \circ L(N)$. Therefore, $L(P) \neq L(M_1) \circ L(N)$.

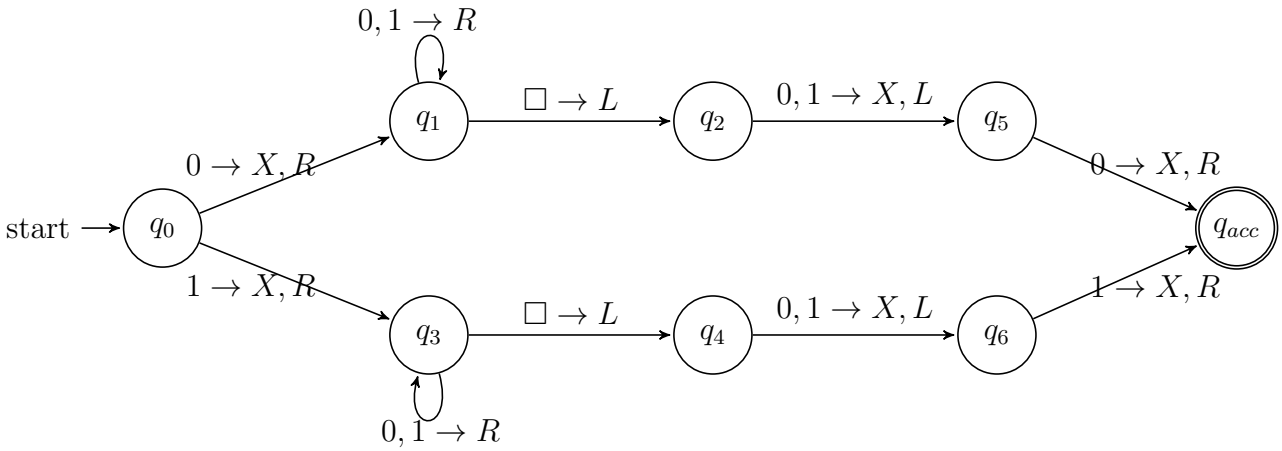
iv. (*Graded for completeness*) Describe the language recognized by the machine that results from taking the two machines M_2 and N, setting the start state of M_2 to be the start state of the new machine, setting the set of accepting states of N to be the set of accepting states of the new machine, and drawing spontaneous arrows from the accepting states of M_2 to the start state of N. Use this description to explain why we used the construction of M_2 from M_1 and how this construction could be used in a proof of the closure of the class of context-free languages under set-wise concatenation.

Solution: The language recognized by the new machine will be $\{1^k 0^m 1^n \mid k \geq 0,\ m \geq n \geq 0\}$ which is the same as $L(M_2) \circ L(N)$. We used the construction M_2 from M_1 because after the accepting computations of M_2, the stack is guaranteed to be empty, so that the computation of the N part of the new machine will not be affected by the previous stack content.

We can use this construction to prove that the set of context-free languages is closed under concatenation. Given a context-free language L_1 recognized by PDA P_1, and given context-free language L_2 recognized by PDA P_2. Then we apply the special construction to transform P_1 into P_1' where accepting computations will end with empty stack. Then build a new machine M that results from taking the two machines P_1' and P_2, setting the start state of P_1' to be the start state of M, setting the set of accepting states of P_2 to be the set of accepting states of M, and drawing spontaneous arrows from the accepting states of P_1' to the start state of P_2. In this case, we showed that there exists a PDA that recognizes $L_1 \circ L_2$, therefore $L_1 \circ L_2$ is also a context-free language. Hence we proved that the set of context-free languages is closed under concatenation.

3. **Turing machines** (12 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 $w_1 = 1010$ and we will trace the computation of T on w_1.

Accepting computation:
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. $q_0 1010$
2. X$q_3$010
3. X0$q_3$10
4. X01$q_3$0
5. X010q_3
6. X01$q_4$0

7. X0$q_6$1X

8. X0Xq_{acc}X

Since the computation reached the accepting state, T halts and accepts $w_1 = 1010$.

(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 $w_2 = 010$ and we will trace the computation of T on w_2.

Rejecting computation:

1. q_0010
2. Xq_110
3. $X1q_10$
4. $X10q_1$
5. $X1q_20$
6. Xq_51X
7. $X\square q_{rej}X$

Since the computation reached the rejecting state, T halts and rejects w_2.

(c) (*Graded for correctness*) Specify an example string w_3 of length 2 over Σ on which the computation of this Turing machine is **never halts** 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.

Note: when a Turing machine does not halt on a given input string, we say that it **loops** on that string.

Solution: There is no such example. There are 4 strings of length 2 over Σ, namely $\{00, 01, 10, 11\}$. The sequence of configurations of the computation of T on 00 is:

$q_0 00$, $X q_1 0$, $X 0 q_1$, $X q_2 0$, $q_5 XX$, $\square q_{rej} X$. The computation enters q_{rej}, so the computation on 00 halts and 00 is rejected. Similarly, tracing the computation of the other three strings, we can see that all of them are rejected. Therefore, there is no example string of length 2 over Σ on which the computation of T never halts.

Bonus: is there a string of some other length on which T does not halt?

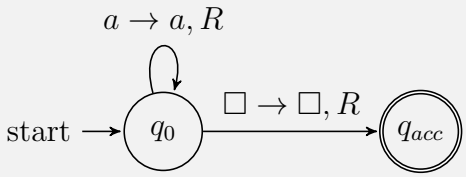
4. **Implementation-level descriptions of deciders and recognizers** (9 points):

For this question, consider the alphabet $\Sigma = \{a, b, c\}$.

(a) (*Graded for correctness*) Give an example of an infinite language over Σ (that is not Σ^*) and give two different Turing machines that recognize it: one that is a decider and one that is not. A complete solution will include a precise definition for your example language, along with **both** a state diagram and an implementation-level description of each Turing machines, along with a brief explanation of why each of them recognizes the language and why one is a decider and there other is not.

Solution: Consider the language $L = \{a^m \mid m \geq 0\}$. For the following Turing machines, fix $\Sigma = \{a, b, c\}$ and $\Gamma = \{a, b, c, \square\}$. Convention: we do not include the node for the reject state q_{rej} and any missing transitions in the state diagram have value $(q_{rej}, \square, R)$.

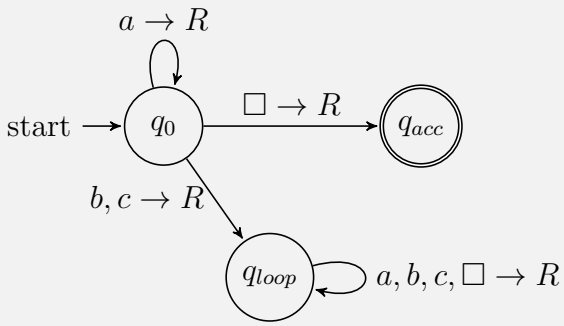
Decider:



Implementation-level description: Scan the tape from left to right. If at some point we see a b or c, reject. If at some point the tape symbol is blank, accept.

Explanation: This Turing machine will accept all strings with only a's and reject all strings that contain b or c. Therefore it recognizes the language $L = \{a^m \mid m \geq 0\}$ which is the set of all strings containing only a's. It will never loop on any input. Therefore, it is a decider.

Turing machine that is not a decider:



Implementation-level description: Scan the tape from left to right. If at some point we see a b or c, move right and continue scanning right no matter what we see afterwards. If we have not seen b or c so far, accept the string when the tape symbol is blank.

Explanation: This Turing machine will accept all strings with only a's and loop on all strings that contains b or c. Therefore it recognizes the language $L = \{a^m \mid m \geq 0\}$ which is the set of all strings containing only a's. It loops on some input. Therefore, it is not a decider.

(b) (*Graded for completeness*) True or false: There is a Turing machine that is not a decider that recognizes the empty set. A complete solution will include a witness Turing machine (given by state diagram or implementation-level description or high-level description) and a justification for why it's not a decider and why it does not accept any strings, or a complete and correct justification for why there is no such Turing machine.

Solution: True. Consider the following high-level description of a Turing machine M:

" On input w

1. Go to step 1."

This Turing machine M will loop on any input string, i.e. M does not halt on any input. Therefore, M is not a decider and it does not accept any strings.

(c) (*Graded for completeness*) True or false: There is a Turing machine that is not a decider that recognizes the set of all string Σ^*. A complete solution will include a witness Turing machine (given by state diagram or implementation-level description or high-level description) and a justification for why it's not a decider and why it accept each string over $\{a, b, c\}$, or a complete and correct justification for why there is no such Turing machine.

Solution: False. For any Turing machine M that recognizes the set of all strings Σ^*, this means that the computation of M on any string $w \in \Sigma^*$ enters q_{accept} in finitely

many steps, i.e. the computation of M on w halts. Therefore, M is a decider. Thus there does not exist a Turing machine that is not a decider that recognizes the set of all string Σ^*.