HW4: Context-free Languages and Turing Machines

CSE105Sp22

In this assignment,

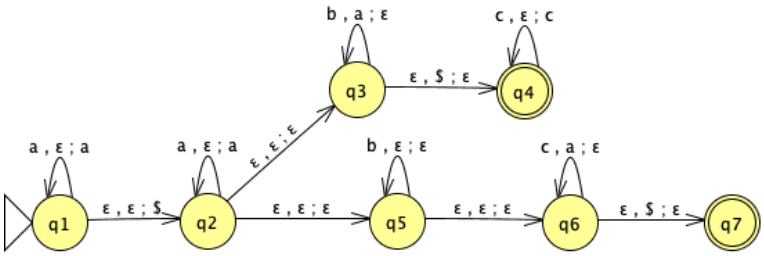
You practiced designing and working with context-free grammars and pushdown automata. You used general constructions to explore the class of context-free languages. You also practiced with the formal definition of Turing machines.

Reading and extra practice problems: Sipser Sections 2.1, 2.2, 2.3 (partially). Chapter 2 exercises 2.1, 2.2, 2.3, 2.4, 2.6, 2.9, 2.10, 2.11, 2.12, 2.13, 2.16, 2.17. Chapter 2 problem 2.30. Chapter 3 exercises 3.1, 3.2.

Assigned questions

1. For this question, we are working over the fixed alphabet $\{a, b, c\}$.

   (a) (*Graded for fair effort completeness*[1])
   Consider the PDA over this alphabet with state diagram



   Give an informal description of this PDA and describe the language it recognizes using set builder notation.

   Hint: Compare the PDA with the machine in Example 2.16 and Figure 2.17 of the textbook (page 116), which recognizes the language $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$ and identify the main differences.

[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 ask that you 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: The main differences between this PDA and the one in the book example are the self loop at q_1 (which changes the set of strings being accepted) and the label of the self-loop at q_4 (which doesn't). Adapting the informal description from the book accordingly, we get

> The PDA first reads a's, pushing some of them onto the stack. It nondeterministically guesses whether to match those a's on the stack either with the b's or the c's, in either case, enforcing that the string has a's followed by b's followed by c's.

The language recognized by this PDA is

$$\{a^i b^j c^k \mid i \geq 0, j \geq 0, k \geq 0 \text{ and } (i \geq k \text{ or } i \geq j)\}$$

(b) (*Graded for correctness*[2])
Consider the CFG $(\{X, S, S_1, S_2, T, Y\}, \{a, b, c\}, R, X)$ where the set of rules R has

$$\begin{aligned} X &\to aX \mid S \mid T \\ S &\to S_1 S_2 \\ S_1 &\to aS_1 b \mid \varepsilon \\ S_2 &\to cS_2 \mid \varepsilon \\ T &\to aTc \mid Y \\ Y &\to bY \mid \varepsilon \end{aligned}$$

For each of the following strings, either give a derivation in this grammar that proves the string is in the language generated by the grammar, or explain why there is no such derivation.

i. *aaaa*
Solution: There are multiple derivations in this grammar that prove *aaaa* is in the language generated by the grammar. We give one example.

$$X \implies aX \implies aaX \implies aaaX \implies aaaaX \implies aaaaT \implies aaaaY \implies aaaa$$

ii. *abbc*
Solution:

$$X \implies T \implies aTc \implies aYc \implies abYc \implies abbYc \implies abbc$$

iii. *aabb*
Solution:

$$X \implies S \implies S_1 S_2 \implies S_1 \implies aS_1 b \implies aaS_1 bb \implies aabb$$

[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.

(c) (*Graded for correctness*) Modify the start variable of this context-free grammar to get a different CFG (with the same set of variables, set of terminals, and set of rules) that generates an **infinite regular language**, if possible. A complete solution will include either (1) the formal definition of this new CFG and an explanation of why the language it recognizes is both infinite and regular, or (2) a sufficiently general and correct argument for why there is no way to choose the start variable to satisfy this requirement.

Solution: Consider the start variable S_2, so we have the grammar

$$(\{X, S, S_1, S_2, T, Y\}, \{a, b, c\}, R, S_2)$$

Each derivation must start with S_2 and the rules with S_2 on their left-hand-side have strings over $\{S_2, c\}$ on their right-hand-side. Thus, all derivations in this grammar will only ever have the variable S_2. The language of this grammar is $\{c^k \mid k \geq 0\} = L(c^*)$ because each string of k many c's can be derived by a $k+1$ length derivation in this grammar (applying $S_2 \to cS_2$ k times, and then applying $S_2 \to \varepsilon$ at the end), and no other strings can be derived because the terminals a and b do not show up in the right-hand-side of any rule with S_2 on the left-hand-side. Thus, the language of this grammar is infinite and regular.

2. (*Graded for correctness*) In this question, you'll practice working with formal general constructions for PDAs and translating between state diagrams and formal definitions.

Suppose

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

is a PDA. We can define a new PDA N so that $L(M) = L(N)$ and N is **guaranteed to have an empty stack** at the end of any accepting computation. Informally, the construction is as follows: Add three new states q_1', q_2', q_3' and one new stack symbol #.

- One of the new states q_1' will be the new **start** state and it has a spontaneous transition to the old start state q_0 which pushes the new stack symbol # to the stack.
- The transitions between the old states are all the same.
- From each of the old accept states, **add** a spontaneous transition (that doesn't modify the stack) to the second new state q_2'.
- In this state q_2', pop all old stack symbols from the stack without reading any input.
- When the new stack symbol # is on the top of the stack, transition to the third new state q_3' and accept.

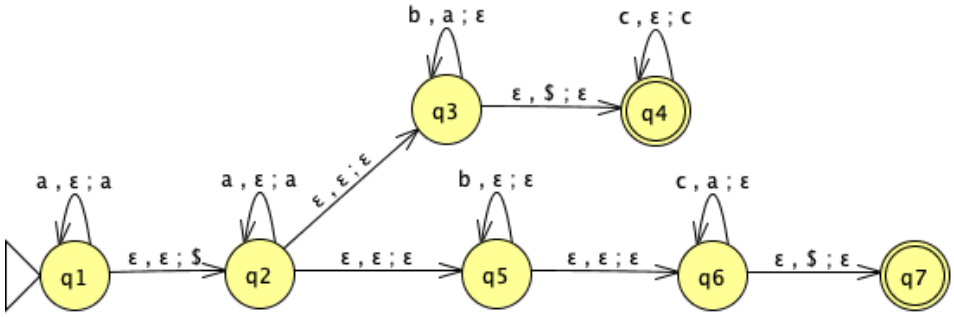
Assume $\{q_1', q_2', q_3'\} \cap Q = \emptyset$ (otherwise, relabel some of the states in Q) and assume that $\# \notin \Gamma$ (otherwise, relabel this stack symbol in Γ). Define N to be

$$N = (Q \cup \{q_1', q_2', q_3'\}, \Sigma, \Gamma \cup \{\#\}, \delta_N, q_1', \{q_3'\})$$

where $\delta_N : Q \cup \{q_1', q_2', q_3'\} \times \Sigma_\varepsilon \times \Gamma_\varepsilon \cup \{\#\} \to \mathcal{P}(Q \cup \{q_1', q_2', q_3'\} \times \Gamma_\varepsilon \cup \{\#\})$ is defined as

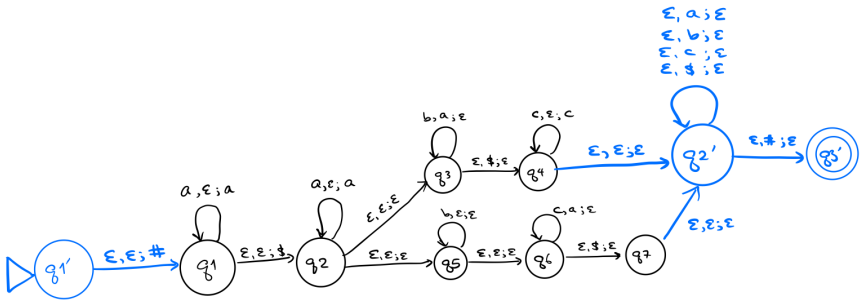
$$\delta_N(\ (q,x,y)\ ) = \begin{cases} \{(q_0, \#)\} & \text{if } q = q_1',\ x = \varepsilon,\ y = \varepsilon \\ \delta(\ (q,x,y)\ ) & \text{if } q \in Q,\ x \in \Sigma,\ y \in \Gamma_\varepsilon \\ \delta(\ (q,x,y)\ ) & \text{if } q \in Q,\ x = \varepsilon,\ y \in \Gamma \\ \delta(\ (q,x,y)\ ) & \text{if } q \in Q \setminus F,\ x = \varepsilon,\ y = \varepsilon \\ \delta(\ (q,x,y)\ ) \cup \{(q_2', \varepsilon)\} & \text{if } q \in F,\ x = \varepsilon,\ y = \varepsilon \\ \{(q_2', \varepsilon)\} & \text{if } q = q_2',\ x = \varepsilon,\ y \in \Gamma \\ \{(q_3', \varepsilon)\} & \text{if } q = q_2',\ x = \varepsilon,\ y = \# \\ \emptyset & \text{otherwise} \end{cases}$$

(a) (*Graded for correctness*) Illustrate this construction by considering the PDA M over the input alphabet $\{a, b, c\}$



and applying the construction above to create the related PDA N and include its state diagram in your submission. *Note: you may include the formal definition of your PDA, but this is not required.*

Solution: The state diagram of the result of applying the construction to M is



(b) (*Graded for correctness*) Pick a string of length 5 over the alphabet of the PDA M and use it to demonstrate the difference in M and in N by

- describing an accepting computation of M on this string for which the stack is not empty at the end of the computation, and
- describing an accepting computation of N on this string for which the stack is empty at the end of the computation.

In your descriptions of these computations, include both the sequence of states visited by the machine as well as snapshots of the full contents of the stack at each step in the computation. You may hand-draw and scan these traces of the computations.

Hint: You will need to pick your example string wisely. It must be accepted by M and there must be a computation of M on your string which ends with a nonempty stack. Not all choices of length 5 strings work.

Solution: Consider the string $aaabb$. We'll trace the accepting computations of M and N on this string and show that the stack of M is not empty at the end of this accepting computation, but that the stack of N is empty at the end of this accepting computation.

	State	Action for input	Stack
	$q1$	read a	push a: TOP a
	$q2$	spontaneous	push \$: TOP \$$a$
	$q2$	read a	push a: TOP a\$$a$
For M:	$q2$	read a	push a: TOP aa\$$a$
	$q3$	spontaneous	no pop/push: TOP aa\$$a$
	$q3$	read b	pop a: TOP a\$$a$
	$q3$	read b	pop a: TOP \$$a$
	$q4$	spontaneous	pop \$: TOP a

The path ends in $q4$, an accepting state, after processing all five characters of the input. Note that the stack is not empty at the end of this accepting computation.

	State	Action for input	Stack
	$q1'$	spontaneous	push #
	$q1$	read a	push a: TOP a#
	$q2$	spontaneous	push \$: TOP \$$a$#
	$q2$	read a	push a: TOP a\$$a$#
	$q2$	read a	push a: TOP aa\$$a$#
For N:	$q3$	spontaneous	no pop/push: TOP aa\$$a$#
	$q3$	read b	pop a: TOP a\$$a$#
	$q3$	read b	pop a: TOP \$$a$#
	$q4$	spontaneous	pop \$: TOP a#
	$q2'$	spontaneous	no pop/push: TOP a#
	$q2'$	spontaneous	pop a: TOP #
	$q3'$	spontaneous	pop #: stack is empty

The path ends in $q3'$, an accepting state, after processing all five characters of the input.

3. (*Graded for fair effort completeness*)

Fix an arbitrary alphabet Σ. Prove that the class of context-free languages over Σ is closed under concatenation in two ways:

(a) Prove that, for any languages L_1, L_2 over Σ, if there are PDAs M_1 and M_2 such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$, then there is a PDA that recognizes $L_1 \circ L_2$.

Solution: Given M_1 and $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, F_2)$, apply the construction from the previous question to obtain from M_1 a PDA $N_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, \{q_3'\})$ so that $L(M_1) = L(N_1)$ and each accepting computation of N_1 is such that the stack is empty when the machine enters the accept state. Assume that $Q_1 \cap Q_2 = \emptyset$. We now create the PDA $M = (Q_1 \cup Q_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, q_1, F_2)$ where

$$\delta(\ (q,r,s)\ ) = \begin{cases} \delta_1(\ (q,r,s)\ ) & \text{if } q \in Q_1 \setminus \{q_3'\},\ r \in \Sigma,\ s \in \Gamma_1 \\ \delta_2(\ (q,r,s)\ ) & \text{if } q \in Q_2,\ r \in \Sigma,\ s \in \Gamma_2 \\ \{(q_2, \varepsilon)\} & \text{if } q = q_3',\ r = \varepsilon,\ s = \varepsilon \\ \emptyset & \text{otherwise} \end{cases}$$

Informally, M simulates the computations of N_1 until and unless the computation enters the accept state of the machine, in which case it spontaneously transitions to simulating the computation of M_2 on the rest of the input string. A formal proof would explicitly show that $L(M) = L(M_1) \circ L(M_2)$.

(b) Prove that, for any languages L_1, L_2 over Σ, if there are CFGs G_1 and G_2 such that $L_1 = L(G_1)$ and $L_2 = L(G_2)$, then there is a CFG that generates $L_1 \circ L_2$.

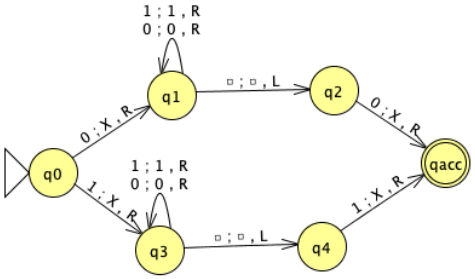
Solution: One approach is to take the PDA built in part (a) of this question and then use the transformation from the book (Theorem 2.20) to transform this PDA to CFG.

Alternatively, we can directly build the CFG G where $L(G) = L_1 \circ L_2$, given CFGs $G_1 = (V_1, \Sigma, R_1, S_1)$ and $G_2 = (V_2, \Sigma, R_2, S_2)$ where $S_1 \in V_1, S_2 \in V_2, V_1 \cap V_2 = \emptyset, L(G_1) = L_2, L(G_2) = L_2$. For $S_{new} \notin V_1 \cup V_2$, define

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

We claim that $L(G) = L_1 \circ L_2$ because every derivation in G must begin with an application of the rule $S_{new} \to S_1 S_2$, after which S_1 is replaced by a string from L_1 and S_2 is replaced by a string from L_2, as required for the definition of set-wise concatenation.

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: 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, for example 1101. At the start of the computation, the machine has 1101 on the four left-most cells of the tape and the rest of the cells blank, with the read write head scanning the leftmost 1. The computation starts in $q0$ and the first transition is to $q3$, replacing the leftmost 1 with X and moving the read write head one step R. That is, the non-blank tape contents are $X101$ and the read-write head is at the leftmost 1. The next transition follows the self-loop so stays at state $q3$, overwrites the 1 with a 1, and moves the read-write head one step R. That is, the non-blank tape contents are $X101$ and the read-write head is at the 0. The self-loop at $q3$ is used again, leading to the same non-blank tape content $X101$ with the read-write head moving to the rightmost 1. Following the self-loop once more, the non-blank tape content is still $X101$ and the read-write tape is scanning a blank cell immediately to the right of the rightmost 1. At this point, the machine transitions to $q4$, overwriting a blank on the blank cell, so the non-blank contents are still $X101$, now with the read-write head scanning the rightmost 1. From $q4$, the machine transitions to $qacc$, overwriting the rightmost 1 with an X so the tape contentx at the end of the computation are $X10X$ and the read-write head is scanning the blank cell immediately to the right of the rightmost X.

(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, for example 011. The computation starts in $q0$ and the first transition is to $q1$, replacing the leftmost 9 with X and moving the read write head one step R. That is, the non-blank tape contents are $X11$ and the read-write head is at the leftmost 1. The next transition follows the self-loop so stays at state $q1$, overwrites the 1 with a 1, and moves the read-write head one step R. That is, the non-blank tape contents are $X11$ and the read-write head is at the rightmost 1. The self-loop at $q1$ is used again, leading to the same non-blank tape content $X11$ with the read-write head scanning a blank cell immediately to the right of the rightmost 1. At this point, the machine transitions to $q2$, overwriting a blank on the blank cell, so the non-blank contents are still $X11$, now with the read-write head scanning the rightmost 1. From $q2$, the machine transitions to $qrej$, the default for missing arrows in the state diagram because the read-write head is scanning a 1, it overwrites that 1 with a blank and moves the read-write head one step to the R. Thus, the machine halts the computation (and rejects) with the nonblank contents of the tape being $X1$ and the read-write head scanning the blank cell two away from the rightmost 1.

(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: There is no such example (in other words, this machine is a decider). We see that by tracing all possible computations of the Turing machine in the state diagram. At each step of the computation, while there are characters from the input string on the tape, the read/write head of the tape moves to the R, and moves L then R once it reaches the end of the input string. In other words, the computation of the Turing machine on the input string ε has exactly one step (immediately transitions to $qrej$), and on all other input strings W, has exactly $|w| + 2$ many steps. Thus, for each input string the Turing machine halts (enters $qacc$ or $qrej$) after finitely many steps and never loops.

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

Solution: "If the leftmost cell of the tape is blank, reject. Check first character of the input string, cross it off, and scan to the right without changing any other cells of the tape. If the last character of the input string matches its first character, cross it off and accept. Otherwise, reject."