CSCE-433 Formal Languages & Automata
CSCE-627 Theory of Computability

Spring 2022

Instructor: Dr. Jianer Chen
Office: PETR 428
Phone: 845-4259
Email: `chen@cse.tamu.edu`
Office Hours: MWF 10:30–11:30am

Senior Grader: Avdhi Shah
Office: N/A
Phone: tba
Email: `avdhi.shah@tamu.edu`
Office Hours: tba

Solutions to Assignment #4

1. Give a regular (i.e., left-linear) grammar for each of the following languages:
(a) all strings over $\{a, b\}$ that do not contain ab;
(b) (CSCE 433 students only) all strings over $\{a, b\}$ that contain at least one a and every a is immediately followed by at least one b;
(c) (CSCE 627 students only) all strings over $\{a, b\}$ with an even number of a's and an odd number of b's

Solutions.

(a) Once an a occurs in such a string, there cannot be a following b, so another way to think of this set is as b^*a^* (thus, it is actually a regular language). The following grammar first generates a sequence of b's, and then generates a sequence of a's. Note that this grammer is actually a regular grammar.

$$\begin{array}{lcl} S & \rightarrow & \epsilon \mid b \mid bS \mid a \mid aA \\ A & \rightarrow & a \mid aA \end{array}$$

(b) The following grammar first generates a sequence of b's (using S), then generates an a followed by a sequence of b's (using B), and then has the option to go back to S to repeat:

$$\begin{array}{lcl} S & \rightarrow & bS \mid aB \\ B & \rightarrow & b \mid bB \mid bS \end{array}$$

(c) We introduce four variables X_{eaob}, X_{eaeb}, X_{oaob}, and X_{oaeb}, where the variable X_{eaob} is for strings with an even number of a's and an odd number of b's. Similar interpretations are for the variables X_{eaeb}, X_{oaob}, and X_{oaeb}. Thus, the grammer can be (where X_{eaob} is the start variable):

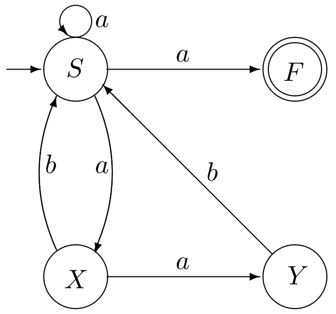
$$\begin{array}{lcl} X_{eaob} & \rightarrow & aX_{oaob} \mid bX_{eaeb} \\ X_{eaeb} & \rightarrow & aX_{oaeb} \mid bX_{eaob} \mid \epsilon \\ X_{oaeb} & \rightarrow & aX_{eaeb} \mid bX_{oaob} \\ X_{oaob} & \rightarrow & aX_{eaob} \mid bX_{oaeb} \end{array}$$

$\square$

2. Convert the following regular grammar into an NFA:

$$\begin{aligned} S &\rightarrow aS \mid aX \mid a \\ X &\rightarrow bS \mid aY \\ Y &\rightarrow bS \end{aligned}$$

Solution. The NFA is given in the following state diagram, where the states are named using the same symbols for the corresponding variables in the grammar.



□

3. Given informal descriptions and state diagrams of pushdown automata for the following languages. (For examples of informal descriptions, see the solutions to Exercise 2.7 on page 160 of the textbook.)

(a) $L_1 = \{wcw^R | w \in \{a,b\}^*\}$. So the set of terminals is $\{a,b,c\}$;

(b) (CSCE 433 students only) L_2 is the set of all binary strings with twice as many 0's as 1's (with no restriction on the order in which the 0's and 1's occur);

(c) (CSCE 627 students only) $L_3 = \{0^n1^n | n \geq 1\} \cup \{0^n1^{2n} | n \geq 1\}$.

Solutions.

(a) We construct a 4-state PDA M to accept the language $L_1 = \{wcw^R | w \in \{a,b\}^*\}$.

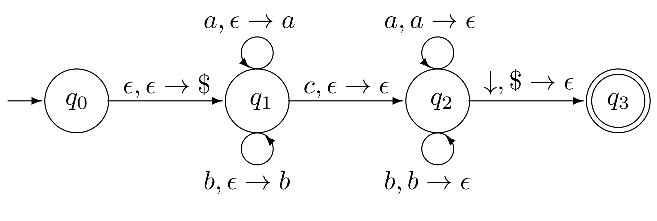
(1) state q_0: the start state, which pushes \$ into the stack to mark the bottom of the stack, then moves to state q_1 to initialize the process;

(2) state q_1: it collects the symbols a and b in the string w before the symbol c, and pushes them into the stack. It moves to state q_3 when it sees the symbol c (without pushing c into the stack). Note that at this point, if you read the stack from the top to the bottom, you get the reverse w^R of the string w stored in the stack;

(3) state q_2: it checks if each symbol in the input after the symbol c matches the corresponding symbol stored in the stack, from stack top to bottom, until it sees the bottom mark \$ in the stack and the end mark $\downarrow$ of the input, which is a sufficient and necessary condition for the input to be of the form wcw^R for a string $w \in \{a,b\}^*$, i.e., to be in the language L_1. In this case, the PDA accepts the input at the final state q_3.

(4) state q_3: the final (i.e., accepting) state.

The state diagram of the PDA is given in the following figure.



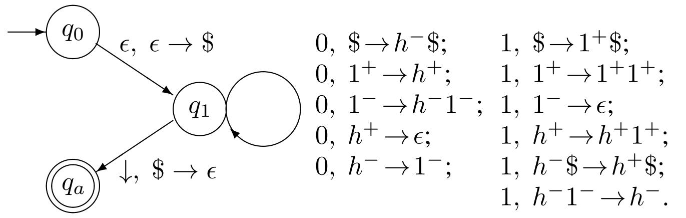
$a, \epsilon \rightarrow a$
$a, a \rightarrow \epsilon$
q_0
$\epsilon, \epsilon \rightarrow \$$
q_1
$c, \epsilon \rightarrow \epsilon$
q_2
$\downarrow, \$ \rightarrow \epsilon$
q_3
$b, \epsilon \rightarrow b$
$b, b \rightarrow \epsilon$

(b) We construct a PDA to accept the language L_2 of all binary strings with twice as many 0's as 1's.

The idea here is to treat each 0 as a $-\frac{1}{2}$, then "add" all numbers in the input to see if it sums 0. Because we can use only a finite number of stack symbols, we introduce four stack symbols h^+ (for $+\frac{1}{2}$), 1^+ (for $+1$), h^- (for $-\frac{1}{2}$), and 1^- (for -1), and perform local computations in the stack while we are reading the input. For example, if the stack top symbol is h^- (i.e., $-\frac{1}{2}$) and the current input symbol is 0 (i.e., another $-\frac{1}{2}$), then we change the stack top symbol to 1^- to include the value for the current input symbol 0. Note that this is implemented by a PDA transition function $\delta(q_1, 0, h^-) = (q_1, 1^-)$, or, in the state diagram notation, "$0,\ h^- \to 1^-$".

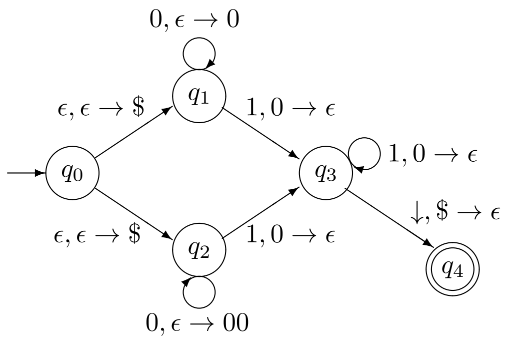
This idea is implemented by the PDA whose state diagram is given in the following figure. The start state q_0 pushes \$ into the stack to initialize the process. The main state is state q_1, which applies the local computations as described above to add the values in the input (thus, 0 is treated as $-\frac{1}{2}$ while 1 is treated as $+1$). You should check the transitions given for the self-loop at the state q_1 to convince yourself that they cover all possible cases correctly. Some of the transitions are briefs of more than one transitions. For example, "$1,\ \$ \to 1^+\$$" is really implemented by two transitions $\delta(q_1, 1, \$) = (q_1', \$)$ and $\delta(q_1', \epsilon, \epsilon) = (q_1, 1^+)$, where q_1' is a new state, i.e., we push the symbol 1^+ into the stack without popping \$ out. Even more complicated, "$1,\ h^-\$ \to h^+\$$" means that if the input symbol is 1, the stack top symbol is h^-, while the symbol below the top symbol h^- is the stack bottom mark \$, then we replace the top symbol h^- by the symbol h^+ (i.e., the total value of the stack was $-\frac{1}{2}$ before we see the new input symbol 1. Thus, adding the new value 1 in the input to the stack should give the total stack value $+\frac{1}{2}$).

Finally, when the PDA reaches the input end $\downarrow$ and sees the stack bottom \$, which means that the total stack value is 0, it accepts.


q_0
$\epsilon,\ \epsilon \to \$$
q_1
$\downarrow,\ \$ \to \epsilon$
q_a
$0,\ \$ \to h^-\$;\quad 1,\ \$ \to 1^+\$;$
$0,\ 1^+ \to h^+;\quad 1,\ 1^+ \to 1^+1^+;$
$0,\ 1^- \to h^-1^-;\quad 1,\ 1^- \to \epsilon;$
$0,\ h^+ \to \epsilon;\quad 1,\ h^+ \to h^+1^+;$
$0,\ h^- \to 1^-;\quad 1,\ h^-\$ \to h^+\$;$
$1,\ h^-1^- \to h^-.$

(c) We construct a PDA to accept the language $L_3 = \{0^n 1^n | n \geq 1\} \cup \{0^n 1^{2n} | n \geq 1\}$

The difficulty here is that we do not know in advance which of the patterns $0^n 1^n$ and $0^n 1^{2n}$ we are looking for. We use nondeterminism to help. The start state q_0 nondeterministically moves to states q_1 and q_2, and in both cases, pushes the symbol \$ to mark the stack bottom. That is, the transition function here is $\delta(q_0, \epsilon, \epsilon) = \{(q_1, \$), (q_2, \$)\}$. We use q_1 to deal with the pattern $0^n 1^n$ while use q_2 to deal with the pattern $0^n 1^{2n}$. Thus, in state q_1, we push a symbol 0 into the stack when we see a 0 in the input. Once we see a symbol 1 in the input, then the state q_1 (for the pattern $0^n 1^n$) should check that the number of 1's in the remaining of the input is equal to the number of 0's stored in the stack. On the other hand, in state q_2, we push *two* 0's into the stack when we see each 0 in the input. Again when we see a symbol 1 in the input, then the state q_2 (for the pattern $0^n 1^{2n}$) should expect that the number of 1's in the remaining of the input is equal to the number of 0's stored in the stack. Thus, at this point, both cases need the same process (i.e., checking the number of 1's in the input is equal to the number of 0's stored in the stack). So both states q_1 and q_2, when seeing a 1 in the input, can go to the same state q_3, which checks the number of 1's in the input with the number of 0's in the stack. Thus, if the input is in the language L_3, then at the end $\downarrow$ of the input, we should see the bottom mark \$ in the stack. The state diagram is given in the following figure.



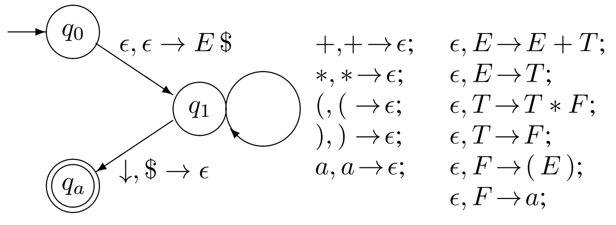
□

4. Convert the following CFG into an equivalent PDA:

$$\begin{aligned} E &\to E+T \mid T \\ T &\to T*F \mid F \\ F &\to (E) \mid a \end{aligned}$$

Solution. As we discussed in class, the PDA works for the grammar as follows. First, it pushes a bottom mark \$ then the start variable (here the state variable is E) into the stack, and moves to state q_1 for the main process. According to the algorithm given in the class, if the stack top symbol is a terminal (note that there are five terminals in this grammar: $+, *, (,), a$), then the state q_1 should check if it matches the current input symbol. On the other hand, if the stack top symbol is a variable X (regardless what is the input symbol), then the state q_1 (nondeterministically) substitutes the variable X by the right-hand side of an X-production. For example, if the stack top symbol is E, then we nondeterministically substitute this E using the two productions of E (i.e.,$E \to E+T \mid T$), which give $E+T$ and T, respectively (that is, the transition function is $\delta(q_1, \epsilon, E) = \{(q_1, E+T), (q_1, T)\}$). This process continues until we see the end mark $\downarrow$ of the input and the bottom mark \$ of the stack, which means we have successfully derive the input from the start variable E, i.e., the input is a string derivable from the grammar.

The state diagram of the PDA is given in the following figure.



$\square$

5. Use the pumping lemma for context-free languages to prove that the following languages are not context-free:

(a) $\{0^n1^n0^n1^n \mid n \geq 0\}$;

(b) $\{w_1cw_2c\ldots cw_k \mid k \geq 2$, each $w_i \in \{a,b\}^*$ and $w_i = w_j$ for some $i \neq j\}$. The alphabet is $\{a,b,c\}$. Each string in the language consists of at least two substrings of a's and b's, the substrings are separated by c's, and at least two of the substrings are equal;

(c) the set of all strings over $\{a,b,c,d\}$ such that the number of a's equals the number of b's, and the number of c's equals the number of d's. Note that there is no restriction on the order in which the symbols occur.

Proof.

(a) The language $L_1 = \{0^n1^n0^n1^n | n \geq 0\}$.

Assume the contrary that the language L_1 is context-free, and let p be the pumping number for L_1. Consider the string $s = 0^p1^p0^p1^p$, which is in L_1. By the pumping lemma, the string s can be written as $s = uvxyz$ with $|vxy| \leq p$ and $|vy| > 0$, and $uv^ixy^iz \in L_1$ for all $i \geq 0$. We call each of the substrings of s that are of the forms 0^p or 1^p a "block" of s.

Case 1. v and y are both contained in the same block. For example, suppose that v and y are both contained in the first block 0^p, then, since $|vy| > 0$, the string uv^2xy^2z would have more 0's in the first block then the 1's in the second block, so it is not a string in the language L_1. This gives a contradiction. The cases where v and y are both contained in the other blocks can lead to a contradiction using a similar analysis.

Case 2. v is contained in one block and y is contained in a different block. Since $|vxy| \leq p$, these two blocks must be adjacent. For example, assume that v is in the first block 0^p and y is in the second block 1^p. Then, since $|vy| > 0$, in the string $s_2 = uv^2xy^2z$, the number of 0's and the number of 1's in the first two blocks cannot be equal to that in the last two blocks, which gives a string not in the language L_1, deriving a contradiction. The cases where v and y are contained in other two adjacent blocks can lead to a contradiction using a similar analysis.

Case 3. Excluding Cases 1-2, then at least one of v and y crosses the boundary of two adjacent blocks. Since $|vxy| \leq p$, the other of v and y must be entirely contained in a single block. For example, assume v crosses the boundary of the first block 0^p and the second block 1^p (which also implies that v is not empty), and y is entirely contained in the second block 1^p. Then the string $s_2 = uv^2xy^2z$ would have more than four blocks that consist of either consecutive 0's or consecutive 1's, which gives a string not in the language L_1. This is a contradiction. The cases where v crosses the boundary of other two adjacent blocks and where y crosses the boundary of two adjacent blocks can lead to a contradiction using a similar analysis.

In summary, in all possible cases, we would derive a contradiction. As a result, our assumption that the language L_1 is context-free cannot be true. This proves that the language L_1 is not context-free.

(b) The language $L_2 = \{w_1cw_2c\dots cw_k | k \geq 2, \text{each } w_i \in \{a,b\}^* \text{ and } w_i = w_j \text{ for some } i \neq j\}$.

Assume the contrary that the language L_2 is context-free. and let p be the pumping number for L_2. Consider the string $s = a^pb^pca^pb^p$. Thus, $w_1 = a^pb^p = w_2$ so the string s in L_2. By the pumping lemma, the string s can be written as $s = uvxyz$ with $|vxy| \leq p$ and $|vy| > 0$, and $uv^ixy^iz \in L_1$ for all $i \geq 0$. We call each of the substrings of the form a^p or b^p a "block" of s.

Case 1. v and y are on the same side of c in the string s, and neither contains c. Then, since $|vy| > 0$, in the string $s_2 = uv^2xy^2z$, the length of the substring on the left side of c would be different from that of the substring on the right side of c, which is not a string in L_2. Contradiction.

Case 2. either v or y contains c. By the definition of the language L_2, $k \geq 2$. So a string in L_2 contains at least one c. However, the string $s_0 = uv^0xy^0z$ contains no c so it cannot be in the language L_2. Contradiction.

Case 3. Excluding Cases 1-2, we must have v on the left side of c and y on the right side of c, and neither contains c. Because $|vxy| \leq p$, v is a substring of b's and y is a substring of a's. Thus, in the string $s_2 = uv^2xy^2z$ (note that c is contained in x), since $|vy| > 0$, either (in case $|v| > 0$) the number of b's before c is larger than the number of b's after c, or (in case $|y| > 0$) the number of a's before c is smaller than the number of a's after c. Thus, s_2 cannot be in L_3. Contradiction.

In summary, in all possible cases, we would derive a contradiction. As a result, our assumption that the language L_2 is context-free cannot be true. This proves that the language L_2 is not context-free.

(c) the language L_3 of all strings over $\{a,b,c,d\}$ such that the number of a's equals the number of b's and the number of c's equals the number of d's. Note that there is no restriction on the order in which the symbols occur.

Assume the contrary that the language L_3 is context-free, and let p be the pumping number for L_3. Consider the string $s = a^pc^pb^pd^p$ in the language L_3. By the pumping lemma, the string s can be written as $s = uvxyz$ with $|vxy| \leq p$ and $|vy| > 0$, and $uv^ixy^iz \in L_1$ for all $i \geq 0$. We call each of the substrings of the form a^p, b^p, c^p, d^p a "block" of s.

Since the blocks a^p and b^p are separated by the block c^p of length p, and since $|vxy| \leq p$, if vy contains a's, then vxy, thus vy, cannot contain b's, and if vy contains b's, then vxy, thus vy, cannot contain a's. In both cases, since $|vy| > 0$, the number of a's and the number of b's in the string $s_0 = uv^0xy^0z$ cannot be the equal. This gives a contradiction that the string s_0 is not in the language L_3.

Using exactly the same way, the cases where either vy contains c's or vy contains d's would lead to the contradiction that the string s_0 is not in the language L_3.

Since $|vy| > 0$, vy must contain at least one of the symbols a, b, c, and d, which will always lead to a contradiction as shown above. As a result, our assumption that the language L_3 is context-free cannot be true. This proves that the language L_3 is not context-free. $\square$