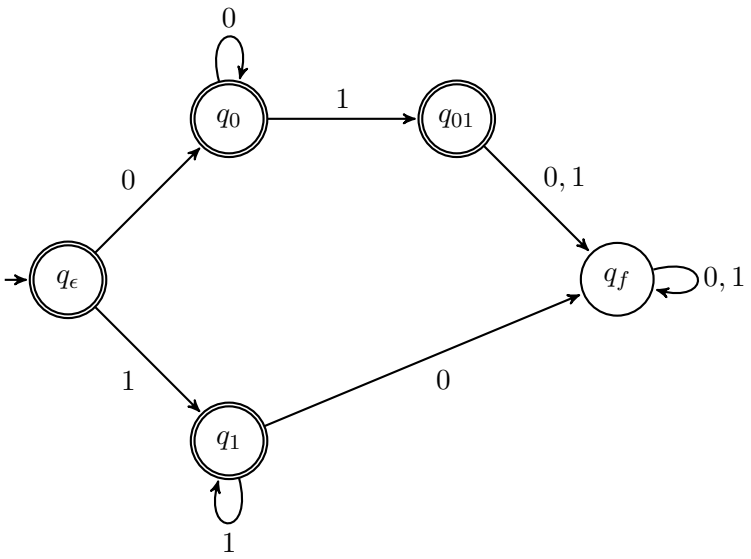
Suggested Solutions to Midterm Problems

1. Let $L = \{w \in \{0,1\}^* \mid w \text{ does not contain 011 or 10 as a substring}\}$.

   (a) Draw the state diagram of a DFA, with as few states as possible, that recognizes L. The fewer states your DFA has, the more points you will be credited for this problem.

   Solution.



   □

   (b) Translate the DFA in (a) systematically to an equivalent context-free grammar (using the procedure discussed in class).

   Solution.

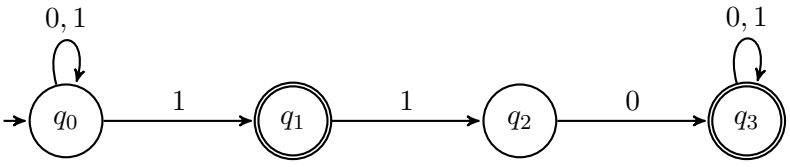
$$\begin{aligned} R_\varepsilon &\to 0R_0 \mid 1R_1 \mid \varepsilon \\ R_0 &\to 0R_0 \mid 1R_{01} \mid \varepsilon \\ R_1 &\to 0R_f \mid 1R_1 \mid \varepsilon \\ R_{01} &\to 0R_f \mid 1R_f \mid \varepsilon \\ R_f &\to 0R_f \mid 1R_f \end{aligned}$$

   □

2. Let $L = \{w \in \{0,1\}^* \mid w \text{ contains 110 as a substring or ends with 1}\}$.

   (a) Draw the state diagram of an NFA, with as few states as possible, that recognizes L. The fewer states your NFA has, the more points you will be credited for this problem.

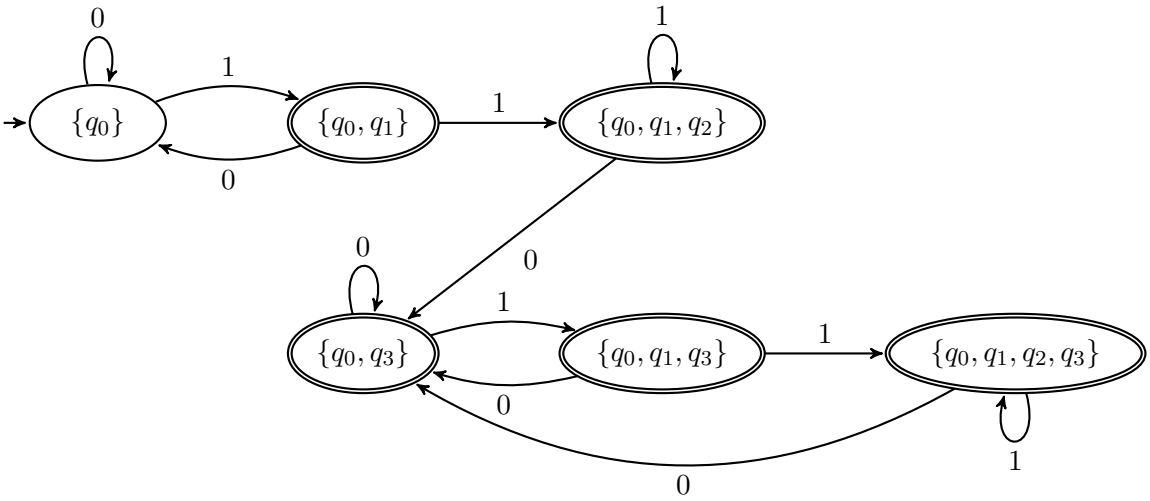
   Solution.



□

(b) Convert the preceding NFA systematically into an equivalent DFA (using the procedure discussed in class). Do not attempt to optimize the number of states, though you may omit the unreachable states.

Solution.



□

3. Let the *rotational closure* of language A be $RC(A) = \{yx \mid xy \in A\}$.

(a) Show that, for any language A, we have $RC(A) = RC(RC(A))$ (i.e., rotational closure, as an operation/function, is idempotent).

Solution. First of all, it is obvious that, for any language A, we have $A \subseteq RC(A)$ (by taking x or y in the definition of RC to be the empty string). Therefore, for any language A, we have $RC(A) \subseteq RC(RC(A))$ readily. It remains to be proven that $RC(RC(A)) \subseteq RC(A)$. For this, we let Σ be the alphabet and show that, for every $w \in \Sigma^*$, if $w \in RC(RC(A))$, then $w \in RC(A)$.

Suppose $w \in RC(RC(A))$. Let $w = yx$ for some $x, y \in \Sigma^*$ such that $xy \in RC(A)$. For $xy \in RC(A)$ to hold, either $xy = x_1x_2y$ and $x_2yx_1 \in A$ for some $x_1, x_2 \in \Sigma^*$ or $xy = xy_1y_2$ and $y_2xy_1 \in A$ for some $y_1, y_2 \in \Sigma^*$. In the first case where $x_2yx_1 \in A$, we have $yx_1x_2 \in RC(A)$ and hence $w = yx = yx_1x_2 \in RC(A)$; analogously, for the second case. □

(b) Show that the class of regular languages is closed under rotational closure.

Solution. Let A be an arbitrary regular language and $M_A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ be a DFA that recognizes A. To prove that $RC(A)$ is also regular, we construct from M_A (as a building block) an NFA N that recognizes $RC(A)$. We first elaborate on the basic ideas and then give a formal definition for N.

Suppose N is given an input $w = yx$ for some $x, y \in \Sigma^*$ such that $xy \in A$. Let q_x be the state in which M_A ends up after reading x. Starting from q_x, M_A should end at some final state after reading y. For N to accept w, we let N simulate M_A from q_x and, after reading y and reaching a final state, make an epsilon transition (which needs to be added to M_A) to the initial state q_A of M_A and continue simulating M_A with the rest of the input. If N eventually ends up at q_x, then the input w is of the correct form of yx such that $xy \in A$. Any state of M_A may act as q_x. For N to start

and finish the simulation at the same state, we need $|Q_A|$ copies of M_A, one for each state in Q_A, with an epsilon transition added from every final state to the initial state. To start the simulation of M_A from any state, N has an epsilon transition from its initial state to every state of M_A.
So, $N = (Q_A \times Q_A \cup \{q_0\}, \Sigma_\varepsilon, \delta, q_0, \bigcup_{q \in Q_A}\{(q,q)\})$, where

$$\begin{cases} \delta(q_0, \varepsilon) = \bigcup_{q \in Q_A}\{(q,q)\} & \\ \delta((q_1,q_2),a) = \{(q,q_2) \mid \delta_A(q_1,a) = q\} & q_1, q_2 \in Q_A \text{ and } a \in \Sigma \\ \delta((q_1,q_2),\varepsilon) = \{(q_A,q_2)\} & q_1 \in F_A \text{ and } q_2 \in Q_A \\ \delta(q,a) = \emptyset & \text{otherwise} \end{cases}$$

□

4. We define the *avoids* operation for languages A and B to be

   A *avoids* $B = \{w \mid w \in A$ and w doesn't contain any string in B as a substring$\}$.

   Prove that the class of regular languages is closed under the *avoids* operation.

   Solution. The definition of A *avoids* B may be restated equivalently as the set difference between A and $\{w \mid w$ contains a string in B as a substring$\}$. The set difference between two languages C and D, denoted $C \backslash D$, equals $C \cap \overline{D}$. If both C and D are regular, then $C \backslash D$ is also regular, as the class of regular languages is closed under intersection and complementation.

   For a regular language B, the set $\{w \mid w$ contains a string in B as a substring$\}$ can be expressed as $\Sigma^* R_B \Sigma^*$, where Σ is the alphabet and R_B is a regular expression for B, and hence is also regular. So, the class of regular languages is closed under the *avoids* operation. □

5. Consider the following CFG discussed in class, where for convenience the variables have been renamed with single letters.

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

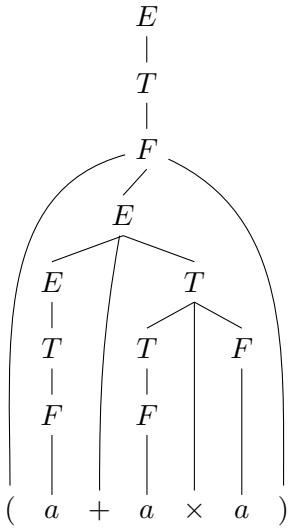
   (a) Give the (leftmost) derivation and parse tree for the string $(a + a \times a)$.

   Solution.

The leftmost derivation

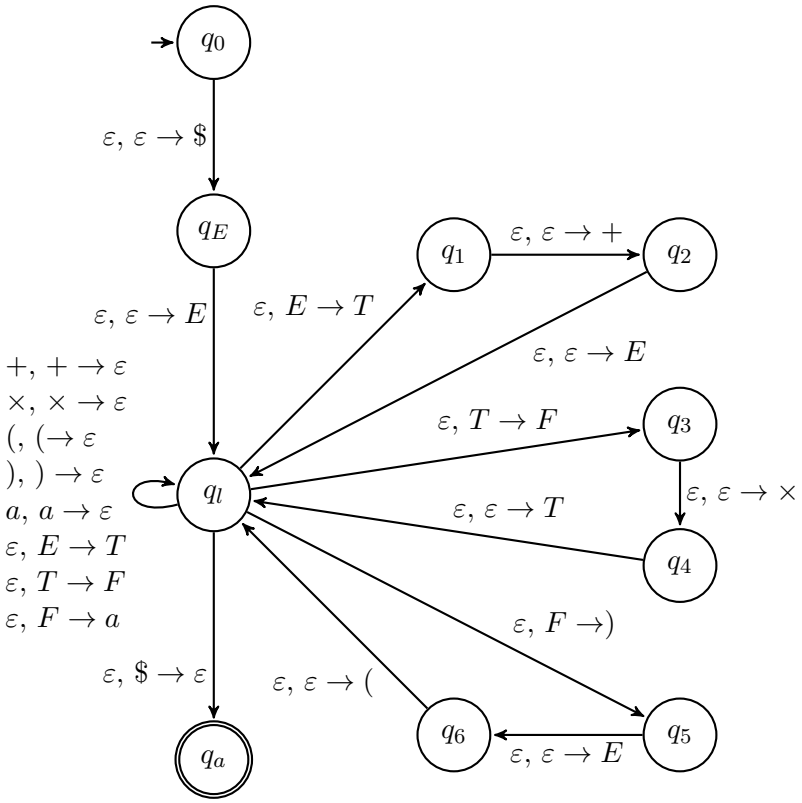
$$
\begin{aligned}
E &\Rightarrow T \\
&\Rightarrow F \\
&\Rightarrow (E) \\
&\Rightarrow (E+T) \\
&\Rightarrow (T+T) \\
&\Rightarrow (F+T) \\
&\Rightarrow (a+T) \\
&\Rightarrow (a+T\times F) \\
&\Rightarrow (a+F\times F) \\
&\Rightarrow (a+a\times F) \\
&\Rightarrow (a+a\times a)
\end{aligned}
$$

The parse tree



□

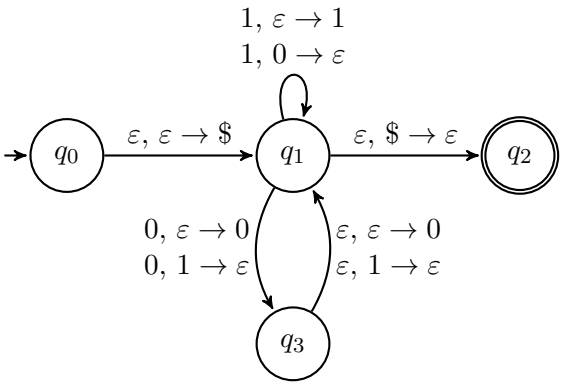
(b) Convert the grammar into an equivalent PDA (that recognize the same language). *Solution.*



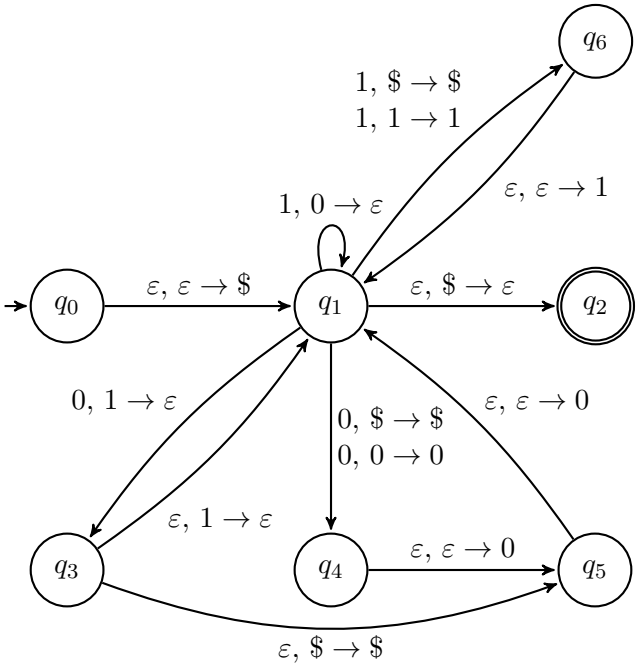
□

6. Draw the state diagram of a PDA that recognizes the following language: $\{w \in \{0,1\}^* \mid w$ has twice as many 1s as 0s$\}$. Please make the PDA as simple and deterministic as possible and explain the intuition behind the PDA.

Solution. A PDA that recognizes the language is shown below. The basic idea is to cancel out every two 1s by a subsequent 0 or the other way around, using the stack to remember outstanding (yet-to-be-cancelled-out) occurrences of 0 or 1. The case when a 0 is read with a 1 outstanding on the stack is effectively the same as a 0 immediately followed by a 1, leaving a 0 on the stack to be cancelled out by a subsequent 1. So, when reading a 0, the PDA pushes two 0s onto the stack, pops two 1s from the stack, or (to allow the case when a 0 is read with a 1 outstanding on the stack) pops a 1 from and pushes a 0 onto the stack. When reading a 1, the PDA pushes a 1 onto the stack or pops a 0 (pushed earlier onto the stack waiting for a 1 to be read) from the stack.



The PDA above is simple enough, but highly nondeterministic. For instance, while there is an outstanding 0 on the stack, the PDA may choose to push a 1 (rather than correctly cancelling out the 0) when reading a 1, even though this choice will turn out to be futile. The following is a more deterministic PDA for the same language.



□

7. For two given languages A and B, define $A \diamond B = \{xy \mid x \in A \text{ and } y \in B \text{ and } |x| = |y|\}$. Prove that, if A and B are regular, then $A \diamond B$ is context-free. (Hint: construct a PDA where the stack is used to ensure that x and y are of equal length.)

   Solution. Given finite-state automata N_A and N_B respectively for A and B, the basic idea is to construct a PDA for recognizing $A \diamond B$ that first simulates N_A and then nondeterministically switches to simulate N_B. The PDA counts the number of symbols while simulating N_A by pushing a marker onto the stack whenever it reads an input symbol and it later cancels out the markers with the input symbols while simulating N_B.

   Suppose $N_A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ and $N_B = (Q_B, \Sigma, \delta_B, q_B, F_B)$, assuming A and B have the same alphabet. We construct the PDA $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, \{q_{\text{accept}}\})$ for $A \diamond B$ as follows:

   - $Q = \{q_{\text{start}}, q_{\text{accept}}\} \cup Q_A \cup Q_B$, where $q_{\text{start}}, q_{\text{accept}} \notin Q_A \cup Q_B$.
   - $\Gamma = \{x, \$\}$.
   - δ is defined as follows.

$$\begin{cases} \delta(q_{\text{start}}, \varepsilon, \varepsilon) = \{(q_A, \$)\} & \\ \delta(q, a, \varepsilon) = \{(q', x) \mid q' \in \delta_A(q, a)\} & q \in Q_A \text{ and } a \neq \varepsilon \\ \delta(q, \varepsilon, \varepsilon) = \{(q', \varepsilon) \mid q' \in \delta_A(q, \varepsilon)\} & q \in Q_A \\ \delta(q, \varepsilon, \varepsilon) = \{(q_B, \varepsilon)\} & q \in F_A \\ \delta(q, a, x) = \{(q', \varepsilon) \mid q' \in \delta_B(q, a)\} & q \in Q_B \text{ and } a \neq \varepsilon \\ \delta(q, \varepsilon, \varepsilon) = \{(q', \varepsilon) \mid q' \in \delta_B(q, \varepsilon)\} & q \in Q_B \\ \delta(q, \varepsilon, \$) = \{(q_{\text{accept}}, \varepsilon)\} & q \in F_B \\ \delta(q, a, t) = \emptyset & \text{otherwise} \end{cases}$$

   It should be clear that $L(M) = A \diamond B$; we omit the detailed proof. □

8. Prove *by induction* that, if G is a CFG in Chomsky normal form, then for any string $w \in L(G)$ of length $n \geq 1$, exactly $2n - 1$ steps are required for any derivation of w.

   Solution. The proposition still holds even if we include all other strings not in $L(G)$ that can be derived from non-start symbols. We will prove this stronger variant by induction on n, the length of an arbitrary nonempty string w. The strengthening in fact will make the inductive proof easier, as we will have a stronger induction hypothesis for the inductive step.

   Base case ($|w| = 1$): The only way to produce a string of length 1 is by applying at the beginning a rule of the form $A \to a$, which constitutes a one-step derivation.

   Inductive step ($|w| = n > 1$): To produce a string of length larger than one, one must first apply a rule of the form $A \to BC$, where B and C are non-start symbols. Suppose the B part eventually produces a string x of length l and the C part a string y of length m such that $xy = w$ and $l + m = n$. From the induction hypothesis, these two parts of derivation take $2l - 1$ and $2m - 1$ steps, respectively. So, the derivation of a string of length n requires $1 + (2l - 1) + (2m - 1) = 2(l + m) - 1 = 2n - 1$ steps. □

9. Let A be the language of all palindromes over $\{0, 1\}$ with equal numbers of 0s and 1s. Prove, using the pumping lemma, that A is not context free. (Note: a *palindrome* is a string that reads the same forward and backward.)

   Solution. We take s to be $1^p 0^p 0^p 1^p$, where p is the pumping length, and show that s cannot be pumped. There are basically three ways to divide s into $uvxyz$ such that $|vy| > 0$ and $|vxy| \leq p$:

Case 1: vxy falls (entirely) within the substring 1^p0^p. No matter what strings v and y get from the division, when we pump down (i.e., $i = 0$), we will lose some 1s or 0s (or both) in the resulting string s'. If we lose some 1s, then there will not be a sufficient number of 1s to match the 1^p in the suffix 0^p1^p and s' is on longer a palindrome. If all 1s remain, then we must lose some 0s and there will be fewer 0s than 1s in s'.

Case 2: vxy falls within 0^p0^p. No matter what strings v and y get from the division, when we pump down (i.e., $i = 0$), there will be fewer 0s than 1s in the resulting string.

Case 3: vxy falls within 0^p1^p. This is analogous to Case 1.

□

10. Let $A = \{wtw^R \mid w, t \in \{0,1\}^* \text{ and } |w| = |t|\}$, where w^R is the reverse of w. Prove that A is not context free.

Solution. We take s to be $1^p0^p(01)^p0^p1^p$, where p is the pumping length, and show that s cannot be pumped. Note that s indeed is of the form wtw^R with $w = 1^p0^p$, $t = (01)^p$, and $|w| = 2p = |t|$. Note also that the $(2p + 1)$-th symbol of s is a 0, while the last $(2p+1)$-th symbol is a 1; similarly, the $(2p+2)$-th symbol is a 1, while the last $(2p+2)$-th symbol is a 0. Each of the two pairs of symmetrical positions contain different symbols and are sufficiently far apart, so if we pump up s in between the symmetrical positions particularly, the resulting string becomes longer and will not be of the form wtw^R with $|w| = |t|$. There are basically five ways to divide s into $uvxyz$ such that $|vy| > 0$ and $|vxy| \leq p$ and we exam each of them below.

Case 1: vxy falls (entirely) within the substring 1^p0^p. If either v or y saddles on the middle point and contains both 1 and 0, then when we pump down, the first p symbols will contain some trailing 0s and cannot be the reverse of 1^p at the end of the resulting string (which is of length at least $3p$). Otherwise, either v contains some 1s but no 0s or both v and y contain only 0s. In the first case, when we pump up, the first $2p$ symbols will have more 1s than 0s and hence cannot be the reverse of 0^p1^p at the end of the resulting string (which is of length greater than $6p$). In the second case, when we pump up, the $(2p+1)$-th symbol will remain a 0 and the last $(2p+1)$-th symbol will also remain a 1 and hence the resulting string (of length greater than $6p$) cannot be of the form wtw^R (with w of length at least $2p + 1$).

Case 2: vxy falls within $0^p(01)^{\frac{p}{2}}$. In this case, no matter what v and y contain, when we pump up, the $(2p + 1)$-th symbol will remain a 0, while the last $(2p + 1)$-th symbol will remain a 1, and hence the resulting string (of length greater than $6p$) cannot be of the form wtw^R (with w of length at least $2p + 1$).

Case 3: vxy falls within $(01)^p$. This is analogous to Case 2.

Case 4: vxy falls within $(01)^{\frac{p}{2}}0^p$. This case is a bit more subtle and is further divided into five subcases:

(a) both v and y are within $(01)^{\frac{p}{2}}$. When we pump up, the $(2p + 1)$-th symbol will remain a 0 and the last $(2p + 1)$-th symbol will remain a 1 and hence the resulting string (of length greater than $6p$) cannot be of the form wtw^R (with w of length at least $2p + 1$).

(b) v is within $(01)^{\frac{p}{2}}$ and y saddles on the middle point. The same argument for Subcase (a) applies, when we pump up.

(c) v is within $(01)^{\frac{p}{2}}$ and y is within 0^p. If y is nonempty, then when we pump up ($i = 2$), the last $(2p+2)$-th symbol will become a 0, while the $(2p+2)$-th symbol will remain

a 1; otherwise (y is empty), when we pump up, the same argument for Subcase (a) applies.

(d) v saddles on the middle point and y is within 0^p. This is analogous to Subcase (c).

(e) both v and y are within 0^p. This is also analogous to Subcase (c).

Case 5: vxy falls within 0^p1^p. This is analogous to Case 4(c).

□

Appendix

- A context-free grammar is in **Chomsky normal form** if every rule is of the form

$$\begin{aligned} A &\rightarrow BC \text{ or} \\ A &\rightarrow a \end{aligned}$$

where a is any terminal and A, B, and C are any variables—except that B and C may not be the start variable. In addition,

$$S \rightarrow \varepsilon$$

is permitted if S is the start variable.

- (Pumping Lemma for Context-Free Languages)

If A is a context-free language, then there is a number p such that, if s is a string in A and $|s| \geq p$, then s may be divided into five pieces, $s = uvxyz$, satisfying the conditions:

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.