1 Context-free grammars

Give context-free grammars for each of the following languages. For each grammar, describe *in English* the language for each non-terminal, and in the examples above. As usual, we won't get to all of these in section.

1. $\{\mathtt{0}^{2n}\mathtt{1}^n \mid n \geq 0\}$

   Solution: $S \to \varepsilon \mid \mathtt{00}S\mathtt{1}$. ■

2. $\{\mathtt{0}^m\mathtt{1}^n \mid m \neq 2n\}$

   [Hint: If $m \neq 2n$, then either $m < 2n$ or $m > 2n$. Extend the previous grammar, but pay attention to parity. This language contains the string $\mathtt{01}$.]

   Solution: To simplify notation, let $\Delta(w) = \#(\mathtt{0}, w) - 2\#(\mathtt{1}, w)$. Our solution follows the following logic. Let w be an arbitrary string in this language.

   - Because $\Delta(w) \neq 0$, then either $\Delta(w) > 0$ or $\Delta(w) < 0$.
   - If $\Delta(w) > 0$, then $w = \mathtt{0}^i z$ for some integer $i > 0$ and some suffix z with $\Delta(z) = 0$.
   - If $\Delta(w) < 0$, then $w = x\mathtt{1}^j$ for some integer $j > 0$ and some prefix x with either $\Delta(x) = 0$ or $\Delta(x) = 1$.
   - Substrings with $\Delta = 0$ is generated by the previous grammar; we need only a small tweak to generate substrings with $\Delta = 1$.

   Here is one way to encode this case analysis as a CFG. The nonterminals M and L generate all strings where the number of $\mathtt{0}$s is More or Less than twice the number of $\mathtt{1}$s, respectively. The last nonterminal generates strings with $\Delta = 0$ or $\Delta = 1$.

$S \to M \mid L$	$\{\mathtt{0}^m\mathtt{1}^n\}\ m \neq 2n$
$M \to \mathtt{0}M \mid \mathtt{0}E$	$\{\mathtt{0}^m\mathtt{1}^n\}\ m > 2n$
$L \to L\mathtt{1} \mid E\mathtt{1}$	$\{\mathtt{0}^m\mathtt{1}^n\}\ m < 2n$
$E \to \varepsilon \mid \mathtt{0} \mid \mathtt{00}E\mathtt{1}$	$\{\mathtt{0}^m\mathtt{1}^n\}\ m = 2n$ or $2n+1$

   Here is a different correct solution using the same logic. We either identify a non-empty prefix of $\mathtt{0}$s or a non-empty prefix of $\mathtt{1}$s, so that the rest of the string is as "balanced" as possible. We also generate strings with $\Delta = 1$ using a separate non-terminal.

$S \to AE \mid EB \mid FB$	$\{\mathtt{0}^m\mathtt{1}^n\}\ m \neq 2n$
$A \to \mathtt{0} \mid \mathtt{0}A$	$\mathtt{0}^+ = \{\mathtt{0}^i\}\ i \geq 1$
$B \to \mathtt{1} \mid \mathtt{1}B$	$\mathtt{1}^+ = \{\mathtt{1}^j\}\ j \geq 1$
$E \to \varepsilon \mid \mathtt{00}E\mathtt{1}$	$\{\mathtt{0}^m\mathtt{1}^n\}\ m = 2n$
$F \to \mathtt{0}E$	$\{\mathtt{0}^m\mathtt{1}^n\}\ m = 2n+1$

Alternatively, we can separately generate all strings of the form $\mathtt{0}^{\text{odd}}\mathtt{1}^*$, so that we don't have to worry about the case $\Delta = 1$ separately.

$$
\begin{array}{ll}
S \to D \mid M \mid L & \{\mathtt{0}^m\mathtt{1}^n\}\ m \neq 2n \\
D \to \mathtt{0} \mid \mathtt{00}D \mid D\mathtt{1} & \{\mathtt{0}^m\mathtt{1}^n\}\ m \text{ is odd} \\
M \to \mathtt{0}M \mid \mathtt{0}E & \{\mathtt{0}^m\mathtt{1}^n\}\ m > 2n \\
L \to L\mathtt{1} \mid E\mathtt{1} & \{\mathtt{0}^m\mathtt{1}^n\}\ m < 2n \text{ and } m \text{ is even} \\
E \to \varepsilon \mid \mathtt{00}E\mathtt{1} & \{\mathtt{0}^m\mathtt{1}^n\}\ m = 2n
\end{array}
$$

■

Solution: Intuitively, we can parse any string $w \in L$ as follows. First, remove the first $2k$ `0`s and the last k `1`s, for the largest possible value of k. The remaining string cannot be empty, and it must consist entirely of `0`s, entirely of `1`s, or a single `0` followed by `1`s.

$$
\begin{array}{ll}
S \to \mathtt{00}S\mathtt{1} \mid A \mid B \mid C & \{\mathtt{0}^m\mathtt{1}^n\}\ m \neq 2n \\
A \to \mathtt{0} \mid \mathtt{0}A & \mathtt{0}^+ \\
B \to \mathtt{1} \mid \mathtt{1}B & \mathtt{1}^+ \\
C \to \mathtt{0} \mid \mathtt{0}B & \mathtt{0}\mathtt{1}^+
\end{array}
$$

Lets elaborate on the above, since k is maximal, $w = \mathtt{0}^{2k}w'\mathtt{1}^k$. If w' starts with `00`, and ends with a `1`, then we can increase k by one. As such, w' is either in 0^+ or 1^+. If w' contains both `0`s and `1`s, then it can contain only a single `0`, followed potentially by $\mathtt{1}^+$. We conclude that $w' \in \mathtt{0}^+ + \mathtt{1}^+ + \mathtt{0}\mathtt{1}^+$.

■

3. $\{\mathtt{0},\mathtt{1}\}^* \setminus \left\{\mathtt{0}^{2n}\mathtt{1}^n \mid n \geq 0\right\}$

   [Hint: Extend the previous grammar. What is missing?]

Solution: This language is the union of the previous language and the complement of $\mathtt{0}^*\mathtt{1}^*$, which is $(\mathtt{0}+\mathtt{1})^*\mathtt{10}(\mathtt{0}+\mathtt{1})^*$.

$$
\begin{array}{ll}
S \to T \mid X & \{\mathtt{0},\mathtt{1}\}^* \setminus \left\{\mathtt{0}^{2n}\mathtt{1}^n\right\}\ n \geq 0 \\
T \to \mathtt{00}T\mathtt{1} \mid A \mid B \mid C & \{\mathtt{0}^m\mathtt{1}^n\}\ m \neq 2n \\
A \to \mathtt{0} \mid \mathtt{0}A & \mathtt{0}^+ \\
B \to \mathtt{1} \mid \mathtt{1}B & \mathtt{1}^+ \\
C \to \mathtt{0} \mid \mathtt{0}B & \mathtt{0}\mathtt{1}^+ \\
X \to Z\mathtt{10}Z & (\mathtt{0}+\mathtt{1})^*\mathtt{10}(\mathtt{0}+\mathtt{1})^* \\
Z \to \varepsilon \mid \mathtt{0}Z \mid \mathtt{1}Z & (\mathtt{0}+\mathtt{1})^*
\end{array}
$$

■

4. All strings in $\{\mathtt{0},\mathtt{1}\}^*$ whose length is divisible by 5.

Solution: We want 5 steps to "count" which index we are on. The only symbol that can end the recurrence is the ε in S. Then we allow any number to transition to the next step and wrap it back to S after 5 inputs.

$$S \to \mathbf{1}A \mid \mathbf{0}A \mid \varepsilon$$
$$A \to \mathbf{1}B \mid \mathbf{0}B$$
$$B \to \mathbf{1}C \mid \mathbf{0}C$$
$$C \to \mathbf{1}D \mid \mathbf{0}D$$
$$D \to \mathbf{1}S \mid \mathbf{0}S$$

■

5. $\{\mathbf{0}^i\mathbf{1}^j\mathbf{2}^{i+j} | i, j \geq 0\}$

Solution: We want to add a **2** for every **0** and **1** added. To do this we divide it into 2 steps. First we add an equal number of **0**s and **2**s on each side. Then we add an equal number of **1**s and **2**s on the inside.

$$S \to \mathbf{0}S\mathbf{2} \mid A$$
$$A \to \mathbf{1}A\mathbf{2} \mid \varepsilon$$

■

6. $\{\mathbf{0}^i\mathbf{1}^j\mathbf{2}^k | i = j \text{ or } j = k\}$

Solution: There are 2 cases, one where i=j and one where j=k. When i=j we add equal **0**s and **1**s then an arbitrary number of **2**s. When j=k we add equal **1**s and **2**s then an arbitrary number of **0**s.

$$S \to AB \mid XY$$
$$A \to \mathbf{0}A\mathbf{1} \mid \varepsilon$$
$$B \to \mathbf{2}B \mid \varepsilon$$
$$X \to \mathbf{0}X \mid \varepsilon$$
$$Y \to \mathbf{1}Y\mathbf{2} \mid \varepsilon$$

■

7. $\{w \in \{\mathbf{0}, \mathbf{1}\}^* | \#(\mathbf{01}, w) = \#(\mathbf{10}, w)\}$ (function $\#(x, w)$ returns the number of occurrences of a substring x in a string w)

Solution: There are 2 cases, one where the string starts with **0** and one where it starts with **1**. When it starts with **0** we can add as many **0**s as we want but when a **1** is added we eventually need to add another **0** to balance the substrings. The second case is the same with **0** and **1** switched.

$$
\begin{aligned}
S &\to \mathbf{0}A \mid \mathbf{1}X \mid \varepsilon \\
A &\to \mathbf{0}A \mid \mathbf{1}B \mid \varepsilon \\
B &\to \mathbf{1}B \mid \mathbf{0}A \\
X &\to \mathbf{1}X \mid \mathbf{0}Y \mid \varepsilon \\
Y &\to \mathbf{0}Y \mid \mathbf{1}X
\end{aligned}
$$

■

Solution (clever): A clever observation is that the criteria for having the same number of substrings of **01** and **10** is satisfied if the string starts and ends with the same symbol. This means we can break the string into 2 cases where we guarantee that if the string starts with a **0** it ends with a **0** and same with **1**s. (We include the zero and one length cases at the start as they are not included in the recursive step)

$$
\begin{aligned}
S &\to \mathbf{0}A \mid \mathbf{1}X \mid \mathbf{0} \mid \mathbf{1} \mid \varepsilon \\
A &\to \mathbf{0}A \mid \mathbf{1}A \mid \mathbf{0} \\
X &\to \mathbf{0}X \mid \mathbf{1}X \mid \mathbf{1}
\end{aligned}
$$

■

Work on these later:

8. $\{w \in \{\mathtt{0},\mathtt{1}\}^* \} \#(\mathtt{0},w) = 2 \cdot \#(\mathtt{1},w)$ – Binary strings where the number of $\mathtt{0}$s is exactly twice the number of $\mathtt{1}$s.

Solution: $S \to \varepsilon \mid SS \mid \mathtt{00}S\mathtt{1} \mid \mathtt{0}S\mathtt{1}S\mathtt{0} \mid \mathtt{1}S\mathtt{00}$.

For any string w, let $\Delta(w) = \#(\mathtt{0},w) - 2 \cdot \#(\mathtt{1},w)$. To prove the correctness of the grammar, we should prove:

(a) Any string x generated by the grammar satisfies $\Delta(x) = 0$.

(b) Any string x such that $\Delta(x) = 0$ can be generated by the grammar.

Part (a) is trivial, since all the production rules add one $\mathtt{1}$ and two $\mathtt{0}$s.

Part (b) can be proved by induction.

As the base case, any string w such that $|w| \le 3$ and $\Delta(w) = 0$ can trivially be generated by the grammar.

Suppose that for some constant k, any string w such that $|w| = k-3$ and $\Delta(w) = 0$ can be generated by the grammar.

To prove that the grammar can generate any string x with $|x| = k$ and $\Delta(x) = 0$, there are two cases to consider.

First, suppose x can be split into two non-empty substrings $x = y_1 y_2$ such that $\Delta(y_1) = \Delta(y_2) = 0$. Then, by the inductive hypothesis, y_1 and y_2 can be generated by the grammar. Therefore, x can be generated by first applying the rule $S \to SS$ and separately generating y_1, y_2.

Then, suppose x cannot be split into two substrings like the above. In other words, for any proper prefix p of x, we have either $\Delta(p) > 0$ or $\Delta(p) < 0$. This case can be further split into three sub-cases.

- Suppose for every proper prefix p of x, $\Delta(p) > 0$. In this case, x must start with $\mathtt{00}$ and end with $\mathtt{1}$, because otherwise there would be some proper prefix q of x such that $\Delta(q) < 0$: If the string starts with $\mathtt{1}$ or $\mathtt{01}$, then we have $q = \mathtt{1}$ or $q = \mathtt{01}$. If the string starts with $\mathtt{00}$ and ends with $\mathtt{0}$, then we define q in the way that $x = q\mathtt{0}$. Moreover, since x starts with $\mathtt{00}$ and ends with $\mathtt{1}$, if we define r in the way that $x = \mathtt{00}r\mathtt{1}$, then $|r| = k-3$ and $\Delta(r) = 0$. Therefore, by the inductive hypothesis and the production rule $S \to \mathtt{00}S\mathtt{1}$, we conclude that x can be generated.
- Suppose for every proper prefix p of x, $\Delta(p) < 0$. In this case, x must start with $\mathtt{1}$ and end with $\mathtt{00}$, with the similar reasoning. If x starts with $\mathtt{0}$, then we have $q = \mathtt{0}$. If it ends with $\mathtt{10}, \mathtt{01}, \mathtt{11}$, then we can let q be the string obtained by removing two symbols at the end from x. Therefore, defining r in the way that $x = \mathtt{1}r\mathtt{00}$, it is clear that we can generate x with $S \to \mathtt{1}S\mathtt{00}$.
- Suppose there exists a constant $m > 0$ such that for all proper prefix p of x with $|p| \le m$, $\Delta(p) > 0$, and for all proper prefix q of x with $|q| > m$, $\Delta(q) < 0$. This is possible because we are comparing $\#(\mathtt{0},w)$ and $2 \cdot \#(\mathtt{1},w)$(For example, consider a string $\mathtt{010}$, and let $p = \mathtt{0}$ and $q = \mathtt{01}$). In this case, x must start with

0, followed by a substring r_1 such that $\Delta(r_1) = 0$, then followed by **1**, followed by another substring r_2 with $\Delta(r_2) = 0$, then end with a **0**, so that we have $x = \mathbf{0}r_1\mathbf{1}r_2\mathbf{0}$. If x starts with 1, it violates the condition $\Delta(p) > 0$. If it is not followed by $r_1\mathbf{1}$, then there would be no q such that $\Delta(p) < 0$. If it is not followed by $r_2\mathbf{0}$, then there would be y_1, y_2 such that $x = y_1y_2$, $\Delta(y_1) = \Delta(y_2) = 0$. Therefore, with the inductive hypothesis and the production rule $S \to 0S1S0$, we conclude that we can generate x.

Since we were able to generate any string x of length k that satisfies $\Delta(x) = 0$, we conclude that the the grammar is correct. ■

9. $\{\mathbf{0}, \mathbf{1}\}^* \setminus \{ww\}\, w \in \{\mathbf{0}, \mathbf{1}\}^*$.

*[Anti-hint: The language $\{ww\}\, w \in \mathbf{0}, \mathbf{1}^*$ is **not** context-free. Thus, the complement of a context-free language is not necessarily context-free!]*

Solution: All strings of odd length are in L.

Let w be any even-length string in L, and let $m = |w|/2$. For some index $i \le m$, we have $w_i \neq w_{m+i}$. Thus, w can be written as either $x\mathbf{1}y\mathbf{0}z$ or $x\mathbf{0}y\mathbf{1}z$ for some substrings x, y, z such that $|x| = i-1$, $|y| = m-1$, and $|z| = m-i$. We can further decompose y into a prefix of length $i-1$ and a suffix of length $m-i$. So we can write any even-length string $w \in L$ as either $x\mathbf{1}x'z'\mathbf{0}z$ or $x\mathbf{0}x'z'\mathbf{1}z$, for some strings x, x', z, z' with $|x| = |x'| = i-1$ and $|z| = |z'| = m-i$. Said more simply, we can divide w into two odd-length strings, one with a **0** at its center, and the other with a **1** at its center.

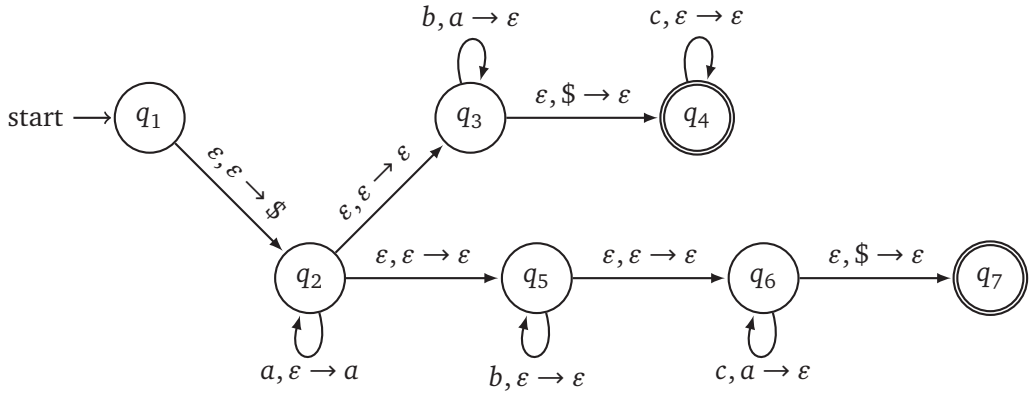
$S \to AB \mid BA \mid A \mid B$	strings not of the form ww
$A \to \mathbf{0} \mid \Sigma A \Sigma$	odd-length strings with **0** at center
$B \to \mathbf{1} \mid \Sigma B \Sigma$	odd-length strings with **1** at center
$\Sigma \to \mathbf{0} \mid \mathbf{1}$	single character

■

2 Push-down automata

The next few problems deal with push-down automata (PDA). The goal of these problems is to simply gain an understanding of PDAs which are the machines needed to recognize a context-free language:

1. What language does the following push-down automata recognize (Hint: This is a non-deterministic automata as most PDAs are)?



Solution: This example illustrates a pushdown automata that recognizes the language:

$$L = \{a^i b^j c^k | i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\} \quad (1)$$

Informally the PDA for this language works by first reading and pushing the a's. When the a's are done, the machine has all of them on the stack so that it can match them with either the b's or c's. Thsi maneuver is a bit trickyh because the machine doesn't know in advance whether to match the a's with the b's or c's. Nondeterminism comes in handy here.

Using its nondeterminism, the PDA can guess whether to match the a's with the b's or with the c's as shown above. Think of the machine as having two branches of its nondeterminism, one for each possible guess. If either of them matches, that branch accepts and the entire machine accepts.

[a]

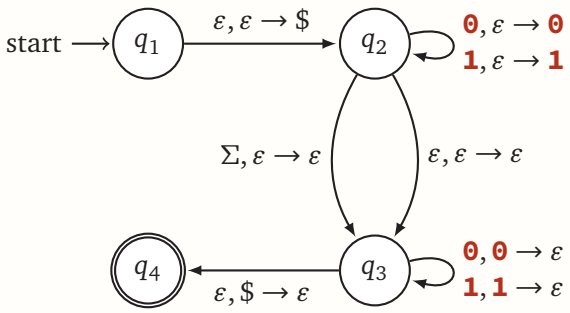
■

[a] Solution borrowed from Sipser et al. "Introduction to the Theory of Computation" textbook (Figure 2.17).

2. Develop the PDA for the language:

$$L = \{w \text{ is a palidrome and } w \in \{\mathtt{0}, \mathtt{1}\}^*\} \tag{2}$$

Solution:



The informal description of the PDA is as follows:

Begin by pushing the symbols that are read onto the stack. At each point, non-deterministically guess that the middle of the string has been reached and and then change into popping off the stack for each symbol red, checking to see that they are the same. IN a palindrome, you can either have even or odd number of characters. To make the PDA accept odd-lengthed palindromes, we add a edge between q_2 and q_3 to "burn" one character. If they were always the same symbol, and the stack empties at the same time as the input is finished, accept; otherwise, reject.

[a] ■

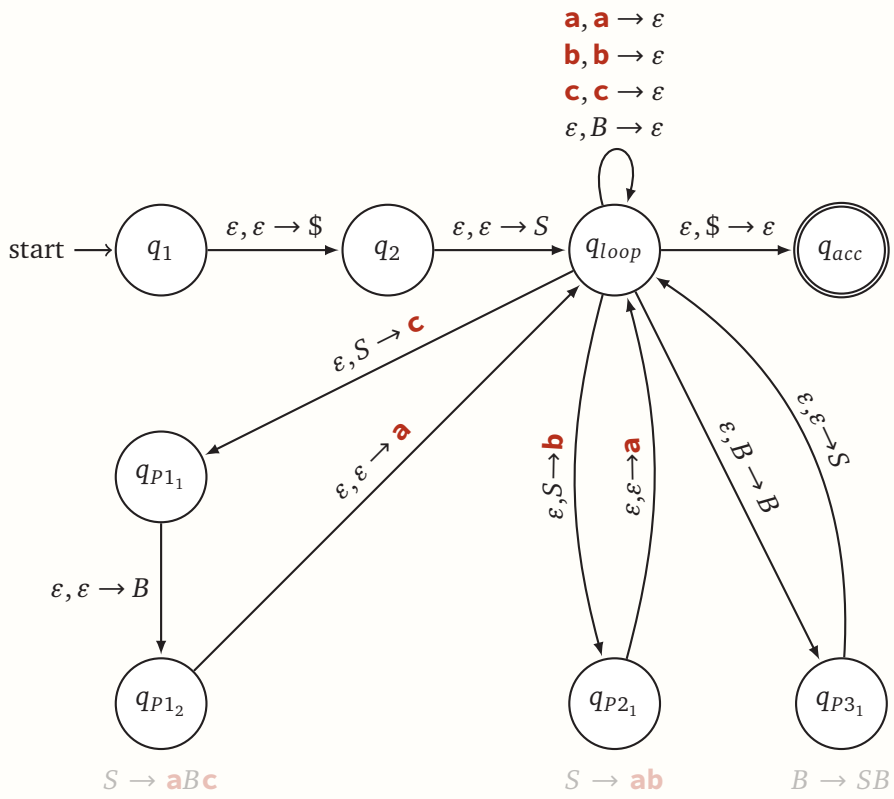
[a] based off an example in from Sipser et al. "Introduction to the Theory of Computation" textbook (Example 2.18).

Work on these later:

3. Convert the following CFG into a PDA:

$$S \to \mathbf{a}B\mathbf{c} \mid \mathbf{ab}$$
$$B \to SB \mid \varepsilon$$

Solution:



As discussed in lecture, there is a simple (though slightly tedious) procedure for converting a CFG into a non-deterministic PDA. The main idea is that you want to convert the accepted string on the stack only popping the leftmost terminals in the correct order. Hence, we construct the pda above as follows:

(a) We need to mark the beginning of the stack with a dollar symbol to avoid triggering any of our epsilon transitions should we run out of stack memory.

(b) Next we need to put the start variable on the stack to begin constructing the string.

(c) Next we insert a loop state which does two things:

- Pops terminals off the stack. This is akin to reading the characters of the string left to right.
- provide an anchor for the other state loops that represent the production rules.

(d) For each production rule, we want to replace a variable on the stack with the mix of terminals/variables that the production rule specifies. Starting at the loop state, we create paths which describe all the production rules in the grammar.

(e) Finally we add an accept state that is only reachable when the input is empty and the stack is clear of all terminals/variables.

a

■

[a] Inspired by the youtube video: https://www.youtube.com/watch?v=ZImtQBMSW_Y