1. Use the pumping lemma to show that the language of all words over the alphabet \{a, b\} containing the same number of a’s and b’s is not regular.

Solution. Consider the demon game of Lecture 6. For any \(k\) chosen by the demon, we pick \(x = \epsilon, y = a^k\) and \(z = b^k\). No matter how the demon partitions \(y\) as \(uvw\) with \(v \neq \epsilon\), by picking \(i = 0\) we have that \(xuv^izw = a^mb^k\) for some \(m < k\) which does not contain the same number of a’s and b’s.

2. Given a language \(L\) over an alphabet \(\Sigma\), the complement of \(L\) w.r.t. \(\Sigma\) is defined to be all words over \(\Sigma\) that are not members of \(L\). The reverse of a language is the set of words which, if their letters were reversed, would give words in \(L\).

(a) Prove that if \(L\) is a regular language then the complement of \(L\) is also regular.

Solution. If \(L\) is regular then by definition there is a DFA, say \(M = (Q, \Sigma, \delta, s, F)\), such that \(L(M) = L\). We now show that \(\overline{M} = (Q, \Sigma, \delta, s, Q \setminus F)\) accepts the complement of \(L\) w.r.t. \(\Sigma\), that is, \(L(\overline{M}) = \Sigma^* \setminus L\).

From the definition of DFA’s, it is straightforward to see that for any word \(w \in \Sigma^*\), there is a unique path \(\pi_w\) labelled \(w\) in \(M\) from the initial state \(s\). Since \(M\) and \(\overline{M}\) are the same but the set of final states, we have that for any \(w \in \Sigma^*\), \(M\) and \(\overline{M}\) have the same path \(\pi_w\) starting from \(s\) and labelled \(w\). Suppose that \(\pi_w\) ends with state \(q\). Now, if \(w \in L\) (respectively, \(w \notin L\)) then \(q\) is final (respectively, not final) in \(M\) and consequently \(q\) is not final (respectively, final) in \(\overline{M}\). This shows that \(\overline{M}\) accepts all words not accepted by \(M\). Since \(L(\overline{M}) = \Sigma^* \setminus L\), we can conclude that the complement of \(L\) is also regular.

(b) Prove that if \(L\) is a regular language then so is the reverse of \(L\).

Solution. If \(L\) is regular then (by definition) there is a DFA \(M\) such that \(L(M) = L\). Let \(M = (Q, \Sigma, \delta, q_0, F)\). We construct an \(\epsilon\)NFA \(M^R = (Q \cup \{s\}, \Sigma, \Delta, s, \{q_0\})\) (where \(s \notin Q\)) where \(\Delta\) is defined as follows:

- \(\Delta(s, \epsilon) = F\);
- \(\Delta(s, \sigma) = \emptyset\), for every \(\sigma \in \Sigma\);
- \(\Delta(q, \epsilon) = \emptyset\), for any \(q \in Q\);
- \(\Delta(q, \sigma) = \{q' | \delta(q', \sigma) = q\}\), for any \(q \in Q\) and \(\sigma \in \Sigma\).

The argument of why \(M^R\) accepts the reverse of \(L\) is the following:

- If there is an \(M\) path \(\pi\) from \(q_0\) to a final state \(q_F\) labelled \(w\) there must be an \(M^R\) path \(\pi^R = s\pi^R\) labelled \(w^R\) where \(\pi^R\) is the reverse path of \(\pi\) and \(w^R\)
is the reverse of \( w \). (This claim can be proved by induction on the length of the path.) Thus, if \( w \in L(M) \) then \( w^R \in L(M^R) \).

- Similarly, if there is an \( M^R \) path \( \pi \) starting from \( s \) and ending up at state \( q_0 \), then \( \pi \) must be of the form \( s\pi \) where \( \pi \) is a path from a state in \( F \) to \( q_0 \). Let \( w \) be the label of \( \pi \). From the definition of \( M^R \) we can easily prove (using induction) that, \( \pi^R \) is an accepting path of \( M \) labelled with \( w^R \). Therefore, if \( w \in L(M^R) \) then \( w^R \in L(M) \).

3. Show that any finite language is regular.

**Solution.** Let \( L \) be a finite language. Let \( L = \{w_1, w_2, \ldots, w_n\} \), for some \( n \in \mathbb{N} \). Then, it is easy to see that \( L(\alpha) = L \) where \( \alpha \) is the regular expression \( w_1 + w_2 + \ldots + w_n \). Since, we have shown (Lecture 4: Kleene’s theorem) that for any regular expression \( \alpha \) there is finite automaton \( M \) such that \( L(\alpha) = L(M) \), we have that \( L \) is regular.

4. Construct a pushdown automaton that accepts (by final state) the set of strings in \( \{a, b\}^* \) that have an equal number of \( a \)'s and \( b \)'s (\( \{x \mid \#a(x) = \#b(x)\} \)). Specify all the transitions. Simulate your automaton on \( \epsilon, aaba \) and \( abba \).

**Solution.** Graphically, one answer is:

![Graphical representation of PDA](image)

Formally, this means that the PA have two states \( \{0, 1\} \), with 0 initial and 1 final; the stack alphabet \( \Gamma \) is \( \{a, b, \bot\} \) and \( \delta \) is defined as follows:

\[
((0, a, \bot), (0, a\bot)), \quad ((0, b, \bot), (0, b\bot)) \\
((0, a, a), (0, aa)), \quad ((0, b, b), (0, bb)) \\
((0, a, b), (0, \epsilon)), \quad ((0, b, a), (0, \epsilon)) \\
((0, \epsilon, \bot), (1, \epsilon))
\]

You should be aware of how the graphical representation corresponds to an underlying formal mathematical structure.

The required simulations are \( (0, \epsilon, \bot) \rightarrow (1, \epsilon, \epsilon) \), \( (0, aaba, \bot) \rightarrow (0, aba, a\bot) \rightarrow (0, ba, aa\bot) \rightarrow (0, a, a\bot) \rightarrow (0, \epsilon, aa\bot) \) and \( (0, abba, \bot) \rightarrow (0, bba, a\bot) \rightarrow (0, ba, a\bot) \rightarrow (0, a, b\bot) \rightarrow (0, \epsilon, \bot) \rightarrow (1, \epsilon, \epsilon) \).

Notice that the \( \epsilon \)-move could have been applied already in the initial configuration in the latter two simulations (and also at the configuration \( (0, ba, \bot) \) in the last
simulation). In those cases we would have ended up at configuration $(1, x, \epsilon)$ where $x \neq \epsilon$. Even though 1 is a final state, this is not an accepting configuration because we have not read in all the input.

Although the question does not ask this, let’s prove that this automaton does the job. If you are not interested, you can skip the rest of this answer. But it is a good idea to at least think about why the things that you write down are correct. A good way to start is to try some simulations. This is like debugging your code. Unfortunately in real life many bugs don’t get caught at the testing stage. For this reason, it is sometimes good to think!

The trick here is to describe the contents of the stack during an arbitrary point of a computation. We know that the initial configuration is $(0, x, \bot)$. A typical accepting computation on $x = x_1 x_2 \ldots x_k$ proceeds as follows:

$$(0, x_1 x_2 \ldots x_k, \gamma_1) \rightarrow (0, x_2 \ldots x_k, \gamma_2) \ldots (0, x_k, \gamma_k) \rightarrow (0, \epsilon, \gamma_{k+1}) \rightarrow (1, \epsilon, \epsilon)$$

where $\gamma_1 = \gamma_{k+1} = \bot$. Notice that the $\epsilon$-move to the final state must be the last move of any accepting computation (because all the input has to be read in first) and it can only happen if the stack has $\bot$ at the top at the second last step.

Let $\overline{x}_i \overset{\text{def}}{=} x_1 \ldots x_{i-1}$, the string of all the symbols to the left of $x_i$. Let $d(x) \overset{\text{def}}{=} |\#a(x) - \#b(x)|$, the difference between the number of a’s and b’s in $x$. As we noticed during the tutorial, for all $1 \leq i \leq k + 1$ we should have:

$$\#(\gamma_i) = d(\overline{x}_i) + 1 \quad (1)$$

that is, the size of the stack is the difference between the number of a’s and b’s that we have seen so far plus one ($\bot$).

If we can prove (1) then in particular we would have that $\#(\gamma_{k+1}) = d(x) + 1$. Since it is easy to see that we can accept precisely when $\#(\gamma_{k+1}) = 1$, this means that we accept if and only if $d(x) = 0$, i.e. that the number of a’s and b’s in $x$ is equal. This would prove that the automaton does its job.

So we need to prove (1). We can try to do this by induction on $i$. The base case is $i = 1$. Here (1) holds because $\overline{x}_1 = \epsilon$ and the stack contains $\bot$ in the initial configuration. So $\#(\gamma_k) = 1$ and $d(\epsilon) + 1 = 1$ and we are done.

Now let’s think about the inductive step. We can suppose that $\#(\gamma_l) = d(\overline{x}_l) + 1$ and we need to prove that $\#(\gamma_{l+1}) = d(\overline{x}_{l+1}) + 1$. By definition $\overline{x}_{l+1} = \overline{x}_l x_l$. So we need to prove that

$$\#(\gamma_{l+1}) = d(\overline{x}_l x_l) + 1$$

Now $x_l$ will be either $a$ or $b$. Here we run into a problem – we know about the length of the stack at the $l$th step in the computation but we don’t know anything about what is on the stack! This is a common feature of induction proofs – you get into the proof and discover that your inductive hypothesis was not strong enough.
So instead of proving (1) let’s try to prove something stronger, that is, something that implies (1) but is more informative:

\[
\gamma_i = \begin{cases} 
  a^{d(x_i)} \bot & \text{if } \#a(x_i) - \#b(x_i) > 0 \\
  b^{d(x_i)} \bot & \text{if } \#b(x_i) - \#a(x_i) > 0 \\
  \bot & \text{if } \#a(x_i) - \#b(x_i) = 0
\end{cases}
\]  

(2)

for all \(1 \leq i \leq k + 1\).

The base case is proved exactly as before. Now let us assume that (2) holds for \(i = l\). Now we need to consider all the cases:

Case \(x_l = a\) and \(\#a(x_l) - \#b(x_l) > 0\): Then \(d(x_{l+1}) = d(x_l) + 1\) because there were more \(a\)’s then \(b\)’s previously and we are adding another one. Here the inductive hypothesis tells us that \(\gamma_l = a^{d(x_l)} \bot\). At this point the automaton must use the third transition to push an \(a\) on the stack. So \(\gamma_{l+1} = a^{d(x_l)+1} \bot = a^{d(x_{l+1})} \bot\). So (2) holds for \(i = l + 1\).

Case \(x_l = a\) and \(\#b(x_l) - \#a(x_l) > 0\): By the inductive hypothesis, \(\gamma_l = b^{d(x_l)} \bot\). So the automaton must use the fifth rule to pop a \(b\) off the stack. It is easy to see that (2) holds for \(i = l + 1\) (we end up with \(\gamma_{l+1}\) fitting either in the second or the third case of (2)).

Case \(x_l = a\) and \(\#b(x_l) - \#a(x_l) = 0\): Clearly \(d(x_{l+1}) = 1\): By the inductive hypothesis, \(\gamma_l = \bot\). Here the automaton uses the first transition and pushes an \(a\) resulting in \(a \bot\). So (2) holds for \(i = l + 1\).

The three cases when \(x_l = b\) are similar, since the rules for \(a\) and \(b\) are symmetric. This completes the proof of (2), which implies (1), which in turn we used to prove the correctness of the pushdown automaton.

5. Given two PDAs, say \(P_1\) and \(P_2\), construct a PDA \(P\) that accepts \(L(P_1) \cup L(P_2)\).

**Solution.** For \(i \in \{1, 2\}\), let \(P_i = (Q_i, \Sigma_i, \Gamma_i, \delta_i, s_i, \bot, F_i)\). Without loss of generality, we assume that \(Q_1 \cap Q_2 = \emptyset\), \(\bot_1 = \bot_2 = \bot\), and \(s \notin (Q_1 \cup Q_2)\).

We define \(P = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \Gamma_1 \cup \Gamma_2, \delta, s, \bot, F_1 \cup F_2)\) where \(\delta\) is the smallest set such that: for \(i \in \{1, 2\}\)

- \(\delta_i \subseteq \delta\);
- \((s, \epsilon, \bot), (s', \bot)) \in \delta\).

To prove that \(L(P) = L(P_1) \cup L(P_2)\), you can proceed as follow. First show by induction that if \(C \Rightarrow_{P_1} C'\) then \(C \Rightarrow_{P} C'\). This allows to prove that \(L(P_1) \cup L(P_2) \subseteq L(P)\). (I advice you to work out the proof.) To prove that \(L(P) \subseteq L(P_1) \cup L(P_2)\), you can show that for \(i \in \{1, 2\}\), if \(C \Rightarrow_{P} C'\) where \(C\) is a configuration of \(P_i\) then \(C \Rightarrow_{P_i} C'\). (Again I advice that you work out the proof.)

\(^1\)notice that this assumption is important for the correctness of the construction below.
6. Consider the following context-free grammar $G$:

$$
S \rightarrow ABS \mid AB \\
A \rightarrow aA \mid a \\
B \rightarrow bA
$$

Which of the followings strings are in $L(G)$ and which are not? Provide derivations for those that are in $L(G)$. Explain the reasons for the strings that are not in $L(G)$.

a) $aabaab$

**Solution.** This string cannot be generated by the grammar. The only production that generates a $b$ is $B \rightarrow bA$ and $A$ must generate at least one $a$. So no string that is generated can end with a $b$.

b) $aaaaba$

**Solution.**

$$
S \rightarrow AB \rightarrow aAB \rightarrow aaAB \rightarrow aaaaB \rightarrow aaaaBA \rightarrow aaaaaba
$$


c) $aabbba$

**Solution.** Again, since the only production that generates a $b$ is $B \rightarrow bA$ and $A$ has to generate at least one $a$, the grammar cannot generate a string with two successive $b$.

d) $abaaba$

**Solution.**

$$
S \rightarrow ABS \rightarrow aBS \rightarrow abAS \rightarrow abaS \rightarrow abaAB \rightarrow abaaB \rightarrow abaabA \rightarrow abaaba
$$

7. Give a grammar with no $\epsilon$- or unit productions that generates the set $L(G) - \{\epsilon\}$ where $G$ is the grammar:

$$
S \rightarrow \alpha Sbb \mid T \\
T \rightarrow bTaa \mid S \mid \epsilon
$$

**Solution.** This is an application of the construction from lecture 6. Recall that for every $\epsilon$-production $B \rightarrow \epsilon$ and for every production of the form $A \rightarrow \alpha B\beta$ we have to add a production $A \rightarrow \alpha\beta$. Also for every unit production of the form $A \rightarrow B$ and production of the form $B \rightarrow \gamma$ we have to add $A \rightarrow \gamma$. This process can be done incrementally. One application gives:

$$
S \rightarrow aSbb \mid T \mid \epsilon \mid bTaa \\
T \rightarrow bTaa \mid S \mid \epsilon \mid baa \mid aSbb
$$
We have added $T \rightarrow baa$ and an extra $\epsilon$-production $S \rightarrow \epsilon$ by following the rule for $\epsilon$-productions. We have added $S \rightarrow bTaa$ and $T \rightarrow aSbb$ by following the rule for unit productions.

Another application gives:

\[
S \rightarrow aSbb \mid T \mid \epsilon \mid bTaa \mid abb \mid baa \\
T \rightarrow bTaa \mid S \mid \epsilon \mid baa \mid aSbb \mid abb
\]

We have added the productions $S \rightarrow abb$, $S \rightarrow baa$ and $T \rightarrow abb$ by following the rule for $\epsilon$-productions. The process stops here because another application does not give any more productions. At this point we can remove the $\epsilon$ and the unit productions to obtain:

\[
S \rightarrow aSbb \mid bTaa \mid abb \mid baa \\
T \rightarrow bTaa \mid baa \mid aSbb \mid abb
\]

Notice that $S$ and $T$ have the same available productions so in fact we can simplify the grammar to:

\[
S \rightarrow aSbb \mid bSaa \mid abb \mid baa
\]

although this last step is not necessary.