Recitation 02: NFAs, Pumping Lemma, PDAs

In this recitation, we’ll continue our discussion of regular languages. We’ll introduce NFAs (Non-deterministic Finite Automata) as an equivalent model of computation to DFAs (Deterministic Finite Automata). Secondly, we’ll practice a technique for showing that languages are not regular. To that end, for a given language A, we have to show that no finite automaton M exists such that L(M) = A. This seems harder than proving a language is regular in which case we only had to demonstrate some FA recognizing A. This is where the pumping lemma comes to the rescue! We’ll end with a discussion of PDAs (Push Down Automata) which is a more powerful computation model and practice some problems with it!

Non-Deterministic Finite Automata

The formal definition of NFAs is in the textbook Definition 1.37 but we’ll repeat it here for illustration.

Definition 1 A non-deterministic finite automaton is a 5-tuple (Q, Σ, δ, q₀, F), where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. δ : Qₑ × Σₑ → P(Q) is the transition function,
4. q₀ ∈ Q, is the start state, and
5. F ⊆ Q is the set of accept states.

With that, let’s do a practice example.

Example 1 Show that regular languages are closed under reversal. Where for a given language A, its reversal is defined as follows:

\[ A^R = \{ w^R \mid \text{for } w \in A \} \]
**Solution 1** Let $A$ be a regular language. Then $L(M) = A$ for some DFA $M$. The idea is to exhibit an NFA $N$ recognizing $A^R$. To that end, the first natural step is to reverse the direction of arrows of $M$. Using this approach, note that if we had multiple arrows going into a state $q$ in the original DFA $M$, now we’ll have multiple arrows going out of $q$ in the NFA $N$. But that’s OK, as we’re building an NFA! For example, if $\delta_M(q_1, a) = \delta_M(q_2, a) = q$ we would now have $q_1, q_2 \in \delta_N(q, a)$.

The next step would be to swap accept states with the start state. This won’t work immediately as we can have multiple accept states (or none) but only one start state (even in an NFA). We can fix this problem by adding an extra state $q_0$ to be the start state of $N$. We then add $\epsilon$-transitions from $q_0$ to all the accept states of the original DFA (i.e. $\delta_N(q_0, \epsilon) = F$ where $F$ is the set of accept states for $M$). Another way to think about this is that, since we’re looking at a string $w$ in reverse order, we’re guessing which accept state $M$ uses to accept $w$ and going backward from there.

**Pumping Lemma**

The Pumping lemma is used to prove that a given language $A$ is not regular. Note that we have to show that there is no FA $M$ such that $L(M) = A$. We use proof by contradiction relying on a property for regular languages provided by the pumping lemma.

**Lemma 1** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Example 2** Show that $A_1$ containing strings of the form $\Sigma^k1^k$ for $k \geq 0$ is non-regular.

**Solution 2** Assume by contradiction that $A_1$ is regular. Then $A_1$ must have a pumping length $p$. Consider the string $s = 0^p1^p$. Note that $s \in A_1$ and $|s| = 2 \cdot p \geq p$. For every partition $s = xyz$, by condition 3, we must have $|xy| \leq p$. This implies that $y$ is all 0’s. As $|y| > 0$ by condition 2, if we pump up (i.e. use condition 1 with $i > 1$), then $xy^iz$ will have more 0’s than 1’s and can’t be in $A_1$; a contradiction.

**Example 3** Show that $A_2$ containing strings of the form $1^k\Sigma^k$ for $k \geq 0$ is non-regular.

Example 3 can be solved using closure properties only and without using the pumping lemma. Namely, assume by contradiction that $A_2$ is regular. Then $A_2^R$ is regular from previous section. But $A_2^R$ is nothing but $A_1$. However, we know that $A_1$ is not regular; a contradiction.
Solution 3  Assume by contradiction that $A_2$ is regular. Then $A_2$ must have a pumping length $p$. Consider the string $s = 1^p0^p$. Note that $s \in A_2$ and $|s| = 2 \cdot p \geq p$. For every partition $s = xyz$, by condition 3, we must have $|xy| \leq p$. This implies that $y$ is all 1's. As $|y| > 0$ by condition 2, if we pump down (i.e. use condition 1 with $i = 0$), then $xy^iz$ will have more 0's than 1's and can't be in $A_2$; a contradiction.

PDAs

PDAs (push-down automata) and CFG (context-free grammars) are two models of computation that are equivalent in power (i.e. a language is recognized by a PDA if and only if it’s recognized by a CFG). We call languages recognized by them context-free languages. With that, let’s start by introducing PDAs.

To that end, we note that the formal definition of PDAs is in the textbook definition 2.13, and we won’t repeat it here. On a high level, PDAs act exactly like an NFA (i.e. they have a finite-state control and are non-deterministic so they can branch-off computation and they accept if and only if at least one of the branches accept), except that PDAs have an extra mechanism called the stack. The stack is an infinite Last-In-First-Out (LIFO) data structure. The stack has an alphabet $\Gamma$ that could be different from the input alphabet $\Sigma$. So now the transition function is defined as follows:

$$\delta : Q \times \Sigma \times \epsilon \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma).$$

We can interpret this as: given a state $q$, while reading input symbol $a$, and a symbol $b$ on top of the stack, then $\delta(q, a, b)$ is the set of possibilities where if $(q', b') \in \delta(q, a, b)$, then one branch of the computation will non-deterministically transition to state $q'$, and push $b'$ on top of the stack. We use the notation $(a, b \rightarrow b')$ on top of the arrow going from $q$ to $q'$ to express such a possibility.

Example 4  Let $A$ be context-free language. Let $B$ be a regular language. Prove that $A \cap B$ is context free.

Solution 4  Let $P$ be a PDA recognizing $A$. Let $M$ be a DFA recognizing $B$. We’ll construct a PDA $P'$ recognizing $A \cap B$ proving that $A \cap B$ is context-free.

On a high level, the idea is simple. Like we did for the closure under $\cup$ proof of regular languages, $P'$ should keep track of the states of $P, M$ simultaneously. Thus, set of states for $P'$ is $Q_{P'} = Q_P \times Q_M$. One difference here is that $P$ is non-deterministic (note that we can’t consider a deterministic
push-down automaton as they’re not equivalent), and it has a stack. Both of these issues can be solved by noting that \( P' \) is a PDA as well. So \( P' \) can branch-off non-deterministically every time \( P \) branches, and \( P' \)'s stack can replicate that of \( P \)'s. Note that we can do all this while, at the same time, keeping track of \( M \)'s states as \( M \) is deterministic.

Finally, a non-deterministic branch of \( P' \) accepts if and only if the corresponding non-deterministic branch of \( P \) accepts and \( M \) accepts. This finishes the proof.

Example 2.18 in the textbook was also done during recitation.