# 6.840/18.404 Recitation 2

September 11th 2020

## 1 Pumping Lemma

The pumping lemma is used to prove that a language is not regular. On its own it is not enough to prove that a language is regular. The mere fact that a specific string of a language can be pumped does not imply that the language is regular.

The Pumping Lemma: If $A$ is a Regular language and $s$ is any string in $A$ ($s \in A$) of length at least equal to $p$ (the pumping length), then $s$ can be broken into 3 parts ($s = xyz$), such that the following hold:

- $s = xy^iz$, for all $i \geq 0$
- $|y| > 0$
- $|xy| \leq p$

In the intuitive proof of the pumping lemma, we said that if the pumping length ($p$) is equal to the number of states of the DFA that recognizes language $A$, then any string equal to or longer than $p$ will repeat at least one state. We can designate the substring of $s$ that led us from that state back to itself as the string $y$ in the pumping lemma. The substring of $s$ before $y$, we label $x$ and the one after it we label $z$. Obviously, by repeating this string an arbitrary number of times or just deleting it completely ($i = 0$), we will remain in the language $A$. Hence, we get the first condition above. The second condition results from the fact that the substring $y$ must be nontrivial. Finally, the third condition arises from the fact that when the DFA has read the first $p$ characters of the string $s$, then it will have gone through $p + 1$ states and, therefore, at least one state must have been repeated. We, therefore, choose $y$ to be the substring that took us from the first state that is repeated back to itself.

The way the proofs involving the Pumping Lemma usually work are as follows:

Q: Prove that language A is not regular.

A: Assume to the contrary, that A is regular. Then every string of $A$ can be pumped. Choose a specific string ($s$) that suits our purposes and apply the

pumping lemma on it in such a way so that the string that results from $xy^iz$ for some choice of $i \geq 0$ is not in the language $A$. Then, $A$ is not regular, so we have a Contradiction.

Examples

1) *Pumping Down*: Let $A = \{o^i 1^j | i \geq j\}$. We prove that $A$ is non-regular. Let $s = 0^p 1^p$, where $p$ is the pumping length. Notice that because $|xy| \leq p$, $y = 0^k$ for $1 \leq k \leq p$. If we pump up ($i \geq 2$), then we get a string $s' = 0^{p+ik}1^p$, which is also in $A$, because $p + ik \geq p$. What we need to notice is that pumping down ($i = 0$) will solve the problem, as we will get the string $s'' = o^{p-k}1^p$, where $p - k < p$, so $s'' \notin A$.

2) *Complementarity Proof*: Prove that the language $B = \{w|w$ has an unequal number of 0's and 1's$\}$ is nonregular. Remember that regular languages and, by extension, nonregular languages are closed under complementation. Hence, we only need to prove that $\overline{B}$ is nonregular. Now, $\overline{B} = \{w|w$ has an equal number of 0's and 1's$\}$, which is easy to show it is nonregular by taking the string $s = o^p 1^p$ for example.

# 2 Context Free Grammars

Context Free Grammars (CFGs) are processes for creating a language following a set of rules. The languages that CFGs recognize are called Context Free Languages (CFLs).

Nondeterministic Pushdown Automata, which we'll simply call Pushdown Automata (PDAs) for this course, are an extension to an NFA that also has access to a stack. A stack is a data structure that works using the First-In Last-Out convention. We can write something on the stack by an operation called pushing or read and delete something from the stack by an operation called popping. It turns out that CFGs and PDAs recognize the exact same set of languages, the CFLs. Hence, any CFG can be converted to a PDA and vice versa.

Converting a CFG to a PDA

Let's convert the following CFG to a PDA as an example (page 120 of the book).

$$S \to aTb|b$$

$$T \to Ta|\epsilon$$

As you can see in Fig. 1, we create 3 main states for the PDA: $q_{start}, q_{loop}, q_{accept}$. In the loop state we encode all possible rules by changing the stack, utilizing every possible rule at the same time nondeterministically. We also pop the

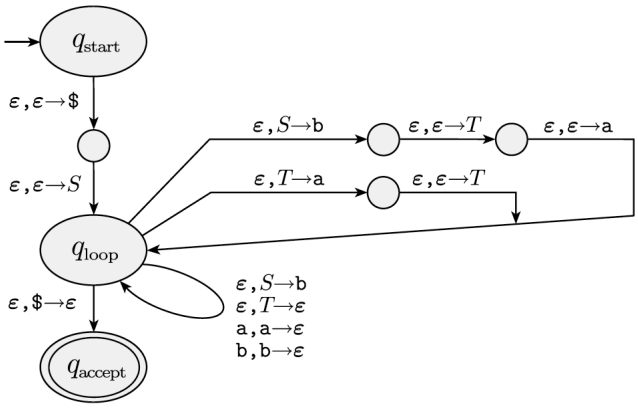


**FIGURE 2.26**
State diagram of $P_1$

Figure 1: The resulting PDA.

terminals if we read the same terminal in the input string and the top of the stack.

CFLs are closed under the regular operations

If we have two CFLs $L_1$ and $L_2$ produced by the CFGs $G_1$ and $G_2$ respectively (and start variables $S_1$ and $S_2$ respectively), then:

*Closure under Union*: The union $L_1 \cup L_2$ is also a CFL. Let $G$ be a new CFG with the addition of a new start state $S$. Simply add the rule $S \to S_1|S_2$. It is easy to check that this is also a CFG that produces the language $L_1 \cup L_2$, so $L_1 \cup L_2$ is a CFL.

*Closure under Concatenation*: The concatenation $L_1L_2$ is also a CFL. Let $G$ be a new CFG with the addition of a new start state $S$. Simply add the rule $S \to S_1S_2$. It is easy to check that this is also a CFG that produces the language $L_1L_2$, so $L_1L_2$ is a CFL.

*Closure under the Star Operation*: The star operation performed on $L_1$ ($L_1^*$) is also a CFL. Let $G$ be a new CFG with the addition of a new start state $S$. Simply add the rule $S \to SS_1|\epsilon$. It is easy to check that this is also a CFG that produces the language $L_1^*$, so $L_1^*$ is a CFL.

Regular Languages are CFLs: The class of Regular languages is a subset of the class of CFLs, because a DFA can be viewed as a simple version of a PDA

that does not use the stack at all. Hence, Regular languages can be recognized by PDAs, so we have: Regular Languages $\subseteq$ CFLs. Also, we have seen some languages, such as $A = \{0^k 1^k | k \geq 0\}$ that are CFLs, but not Regular. Hence, we can say: $\boxed{\text{Regular Languages} \subset \text{CFLs}}$.