**CS 4510: Automata and Complexity** **Fall 2019**

# Lecture 12: Context Free Grammars

October 7, 2019

*Lecturer: Santosh Vempala* *Scribe: Nicolas Soong*

## 12.1 Review

Earlier, we learned that a DFA recognizes regular languages, and a TM recognizes recursively enumerable langauges.

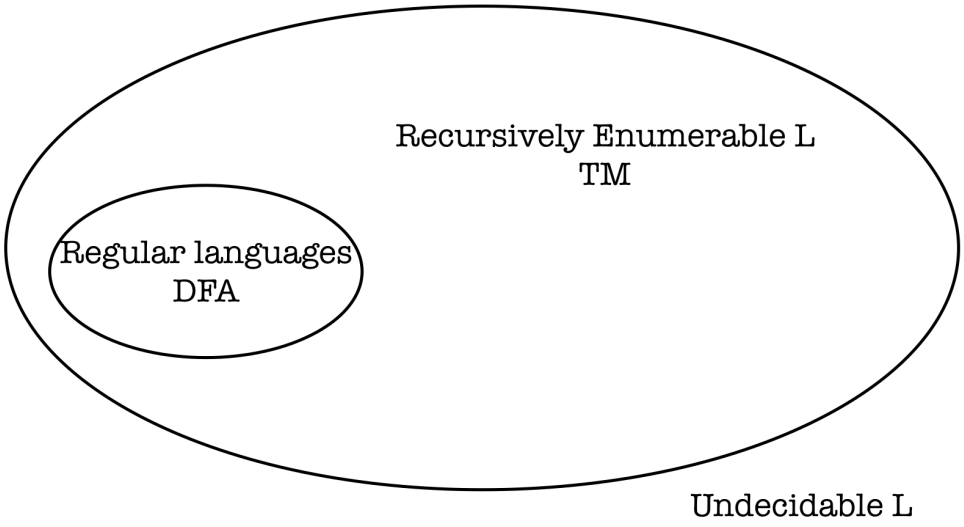


Figure 12.1: DFA $\subseteq$ TM

## 12.2 Introduction

Let's take our example $L = \{0^n 1^n\}$. We can show that $L$ is not regular using the pumping lemma. Suppose there exists a DFA with $p$ states. By pumping lemma, let $n = p$.

$$0^n 1^n = xyz$$

Given the two conditions $|y| > 0$ and $|xy| \leq p$, this implies that $y = 0^i$. Hence, $xz = 0^{n-i}1^n \notin L$. But this contradicts our assumption that $L$ is regular. $\Rightarrow\!\!\!\!\!\!\not\Leftarrow$

So a DFA cannot recognize $L$, but what if there's another type of automata that recognizes $L$? Remember the stack concept from CS 1332? It follows the FIFO principle, where the each item added goes on top of the stack, and the topmost item gets removed first.

This automata that involves the stack principle is called the **pushdown automata**.

The pushdown automata has the similar tuple as that of the DFA, but with one extra symbol, $\Gamma$ - the stack symbols. Recall that

$Q$ - states
$\Sigma$ - the input alphabet
$q_0 \in Q$ - the starting state
$F \subseteq Q$ - the accepting state

Now the transitions are

$$\delta(q, a, b) \to q, c, \text{ i.e. } Q \times \Sigma \times \Gamma \to \mathbb{P}(Q \times \Gamma)$$

Going back to our example, $L = \{0^n 1^n\}$, the algorithm works like this:

```
1 while 0 do
2 |  Push
3 while 1 do
4 |  if stack not empty then
5 |  |  Pop
6 |  else
7 |  |  Reject
8 if end of input and stack empty then
9 |  Accept
```

We use the stack symbols in $\Gamma$ so we push to stack first. When encountered again, stack is empty.

### 12.2.1 Another Example: Balanced Parentheses

Compilers must be aware of balanced parentheses on two conditions: equal number of "(" and ")" and the number of "(" always greater than or equal to the number of ")". The question is: will a DFA work? It won't work because "(" can lead by a lot. What about a PDA? Based on this algorithm:

```
1 while input do
2 |  if ( then
3 |  |  Push
4 |  if ) then
5 |  |  Pop, if empty, Reject
6 if not empty then
7 |  Reject
8 else
9 |  Accept
```

Are all languages accepted by PDAs such as $\{0^n 1^n 2^n\}$? Can we even use the pumping lemma? We'll find the answer to the latter question later, but it turns out there's a grammar that's more general than regular languages - the **context free grammar**.

This example generates the language, $\{0^n 1^n\}$

$$S \to 0S1$$
$$S \to \epsilon$$
$$\text{Example: } S \to 0S1 \to 00S11 \to 000S111 \to 000111$$

This grammar applies to balanced parenthesis:

$$S \to (S)$$
$$S \to SS$$
$$S \to \epsilon$$

The context free grammar has this tuple: $(V, \Sigma, R, S)$, where

$V$: variables (uppercase letters)
$\Sigma$: terminals (lowercase letters)
$R$: production rules
$S \in V$: starting variable
The transitions are $V \to (V \cup \Sigma)^*$

Another form of CFG is **Chomsky Normal Form**. It's the same tuple, but each rule is either:

$$A \to BC,$$
$$A \to a, \text{ or}$$
$$S \to \epsilon$$

Every CFG has an equivalent CNF, and vice-versa.