

CPSC 323

# Compilers and Languages

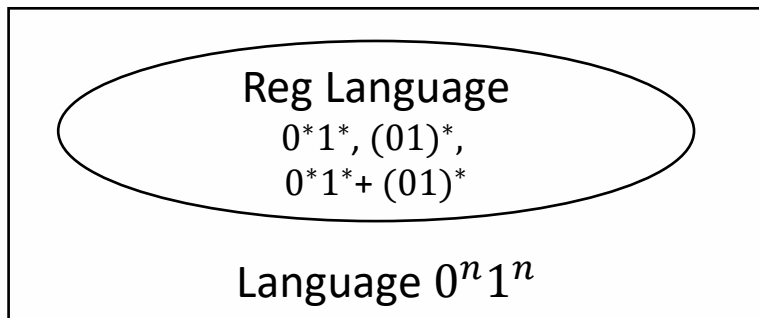
- **Pumping Lemma for RL**
  - **Proof of pumping lemma for nonregular languages**
- **Intro to Syntactic Analysis (*a.k.a.* Parsing)**
- **Context-Free Grammars (CFGs)**



# Recap: Regular Languages (RLs)

- A regular language is a formal language that can be defined by a regular expression and is accepted or recognized by FSA (DFA or NFA).
- How to prove a language is regular?
  - According to the above definition of RL, it is simply to build FSA (DFA or NFA) that accepts or recognizes the language.
- The weakest formal languages widely used
- Many applications

# Nonregular Languages



**Not all languages are regular!**

- Consider the following example languages B and L:

$$B = \{0^n 1^n \mid n \geq 0\}$$

**Question:** Can we build FSA (DFA/NFA) that accepts them? And why?

No. Because that might not be possible using an automaton with finite number of states to recognize language B. This language B of arbitrary amount of information is beyond the capability of FSA. So, language B is not regular.

**More example languages: are they regular or nonregular?**

Language C = {w is a binary string | w has an equal number of 0's and 1's} -- Nonregular

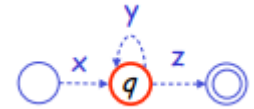
Language D = {w is a binary string | w has an equal number of '01's and '10's} – Regular

# The Pumping Lemma

- It is a method or technique for showing or proving that languages are nonregular.

# The intuition of generalizing the pumping lemma

- Consider an infinite regular language  $L$
- By definition, it is accepted by some FSA  $M$ , which again by definition, has a finite number of states.
- But  $L$  is infinite, there are strings in  $L$  which are very long, and certainly they have more characters than the number of states in  $M$ .
- Since  $M$  accepts every strings in  $L$ , there must be at least one loop in  $M$ . (*exist by the pigeon-hole principle*)
- So, any sufficiently long string  $w \in L$  must traverse some loop, and so can be decomposed into  $xyz$ , where  $y$  is **nonempty** and  $xy^iz \in L$  for any  $i \geq 0$ .
- We say that the substring  $y$  of  $w$  can be pumped.



# Pumping Lemma for Regular Languages

- The Pumping Lemma. If  $L$  is a regular language, there is a positive integer number  $p$  (called the pumping length) such that any string  $w \in L$  with  $|w| \geq p$  can be expressed as  $w = xyz$ , where substrings  $x, y, z$  satisfying the following conditions:

1.  $|y| > 0$
2.  $|xy| \leq p$
3.  $xy^iz \in L$  for  $\forall i \geq 0$

$L$  is regular  $\Rightarrow L$  is pumpable

# How to prove that a language $L$ is not regular using pumping lemma?

Standard technique for proof :  $L$  is not pumpable  $\Rightarrow L$  is not regular

- To prove a language nonregular, we just need to find **one counterexample string!**
- **Towards proof by contradiction:**
  1. Assume that  $L$  is regular.
  2. Use the pumping lemma to guarantee the existence of a pumping length  $p$  such that all strings  $w$  of  $|w| \geq p$  in  $L$  can be pumped.
  3. Find a string  $s$  in  $L$  that has  $|s| \geq p$  but that cannot be pumped. That is, demonstrate every decomposition of  $s$  into  $x$ ,  $y$ , and  $z$  with  $|y| > 0$  and  $|xy| \leq p$  leads to a string  $xy^iz \notin L$  for some  $i \geq 0$ . The existence of such string  $s$  contradicts your assumption.  
Hence,  $L$  cannot be regular.
- **Things to note:**
  - You cannot choose  $p$  - you have to assume this is given by the lemma.
  - You can choose the string  $s$
  - You cannot choose one particular decomposition of  $s$ . You have to prove that no possible decomposition of  $s$  can satisfy any of the above three conditions.

## Example 1: pumping up case

- Show the following example language  $B = \{0^n 1^n \mid n \geq 0\}$  is not regular.
  - 1, Assume  $B$  is regular. Let  $p$  be the pumping length given by the lemma.  
Pick a string  $s = 0^p 1^p$   
(you may have other string  $s$ , e.g.,  $0^{p/2} 1^{p/2}$  where  $p$  is assumed as even for simplicity).
  - 2, Since  $s \in B$  and  $|s| \geq p$ , according to the pumping lemma,  $s$  can be decomposed into  $s = xyz$  such that  $xy^i z \in B$ .
  3. Consider all ways as below where  $s$  cannot be pumped or cannot be decomposed into substrings  $x, y, z$ .
    - 1) Suppose  $y$  only contain 0 and at least contain a 0, then  $xy^2 z$  has more 0's than 1's, so it is not in  $B$ .
    - 2) The case where  $y$  only contain 1 follows the similar way as above.
    - 3) Suppose  $y$  contains '01', then  $xy^2 z$  has some 1's appear before 0's, so it is not in  $B$ .Hence, there is no decomposition of  $s$  leads to  $xy^i z \in B$ , which contradicts the assumption.  
We can prove  $B$  is not regular.



## Example 2: pumping down case

- Show the following example language  $L = \{0^i 1^j \mid i \geq j\}$  is not regular.

## Example 2: pumping down case

- Show the following example language  $L = \{0^i 1^j \mid i \geq j\}$  is not regular.
  - 1, Assume  $B$  is regular. Let  $p$  be the pumping length given by the lemma.  
Pick a string  $s = 0^{p+1}1^p$
  - 2, Since  $s \in B$  and  $|s| \geq p$ , according to the pumping lemma,  $s$  can be decomposed into  $s = xyz$  such that  $xy^i z \in L$ .
  3. According to the condition of  $|xy| \leq p$ , then  $y$  contains only 0. Then  $xy^0 z = xz$  has equal or lesser 0's than 1's. There is no decomposition of  $s$  leads to  $xy^i z \in L$ , which contradicts the assumption.
- So, we can prove  $L$  is not regular.

## Limitation of RL: Nonregularity

- Example languages B and L:

$$B = \{0^n 1^n \mid n \geq 0\}$$

$$L = \{(n)^n \mid n \geq 0\}$$

# Intro to Syntactic Analysis (Parsing)

---

- Example1:

```
distance = velocity * time;
```

- Q1: what is the output of lexical analyzer (lexer) for above example code fragment?

# Intro to Syntactic Analysis (Parsing)

---

- Example 2:

```
height = (width + 56) * factor(foo) ;
```

- Q1: what is the output of lexical analyzer (lexer) for above example code fragment?
- Q2: what does a parser do in the next phase of a compilation?

# Intro to Syntactic Analysis (Parsing)

---

- Example 2:

```
height = (width + 56) * factor(foo) ;
```

- Q1: what is the output of lexical analyzer (lexer) for above example code fragment?

```
id: height = ( id:width + int:56 ) * id:factor ( id:foo ) ;
```

- Q2: what does a parser do in the next phase of a compilation?

It determines whether the input sequence of tokens forms a valid program.

# Lexer and Parser

---

Phase	Input	Output
Lexer	String of characters	Sequence of tokens
Parser	Sequence of tokens	Syntactic structure of the input program in the form of parse tree

Q3: How does a parser parse?

# Context-Free Grammars (CFGs)

---

- They are the fundamental tool used to describe the syntax of a programming language in parser design.
- They are much more powerful than regular expressions as they allow for ***recursive structure*** that is a natural structure in the programming languages.



# Definition of a CFG

---

- A context-free grammar (CFG) is a set of **recursive rules** (or *productions*) used to describe *patterns of strings in a language*.

- A CFG  $G$  is a 4-tuple of  $G = (T, N, S, R)$ :

where:

- $T$ : A finite set of terminals over symbols in alphabet  $\Sigma^*$
- $N$ : A finite set of non-terminals (*also named variables*)  
(note that  $N$  and  $T$  are disjoint sets)
- $R$ : A finite set of rules (*also named productions*)
  - Each rule has the form  $A \rightarrow \alpha$ , where  $A \in N$ , and  $\alpha \in (N \cup T)^*$
  - **Note:** rules are written with single arrows
- $S$  ( $S \in N$ ): A **unique** start symbol of strings
  - $S$  is nonterminal symbol of the *first rule* in set  $R$

# Definitions of a CFG (cont'd)

---

- A finite set of terminals  **$T$** 
  - A terminal is a discrete symbol in alphabet  $\Sigma^*$  that can appear in the language, otherwise known as **a token**. *Examples of terminals: identifiers, keywords, and operators.*
  - We use *lower-case letters* to represent terminals.
- A finite set of non-terminals (also named variables)  **$N$** 
  - A nonterminal (also named a variable) represents a structure that can occur in a language, but is **not** a literal symbol. *Example of nonterminals: declarations, statements, and expressions.*
  - We will use *upper-case letters* to represent non-terminals: *P for program, S for statement, E for expression, etc.*
- **The first rule (or production) of  $R$  is special**: it is the top-level definition of a program, and its nonterminal is known as the *start symbol*  **$S$**  ( **$S \in N$** ).

# Examples of CFG

- Consider the following CFG **G** that describes **expressions** involving *addition*, *integers*, and *identifiers*:

Grammar G:

1. $P \rightarrow E$
2. $E \rightarrow E + E$
3. $E \rightarrow \text{ident}$
4. $E \rightarrow \text{int}$

where  $\mathbf{N} = \{P, E\}$ ,  $\Sigma = \{\text{ident}, \text{int}, +\}$ , and  $\mathbf{R} = \{P \rightarrow E, E \rightarrow E+E, E \rightarrow \text{ident}, E \rightarrow \text{int}\}$ .

- Abbreviating grammars.** By convention, a group of productions that have the same left-hand side are shown with the left-hand side written only once. In all but the first production in the group, write  $|$  instead of  $\rightarrow$ . For example, the above grammar for expressions is usually written as follows.

$$\mathbf{R} = \{P \rightarrow E, E \rightarrow E + E \mid \text{ident} \mid \text{int}\}$$

- Grammar G can also be denoted as:

$$P \rightarrow E$$
$$E \rightarrow E + E$$
$$\mid \text{ident}$$
$$\mid \text{int}$$

**Note:** rules (or productions) of G are written with single arrows

# Examples of CFG

---

- What is the CFG for the following example code statement:  
**distance = velocity \* time;**

# Examples of CFG

---

- What is the CFG for the following example code statement:

```
height = (width + 56) * factor(foo);
```

# Regular languages are context-free

---

- **Theorem.** For every regular language  $L$ , there exists a CFG  $G$  such that  $L=L(G)$ .
- **Corollary.** Every regular language is a CFL.
- **The class of regular languages (RLs) is a proper subclass of CFLs.**

