University of sulaimani
College of science
Department of Computer Science

Computation

Context Free Grammar

Mzhda Hiwa Hama
2024-2025

Regular and Non-Regular Languages

- A language is said to be regular if and only if some finite state machine recognize it .
- Are all language is regular?

  No
- What languages are not regular?

-which are not recognized by any FSM .

-which require memory because finite automata has limited amount of memory it can not store or count strings.

بأتي سؤال في الامتحان تطبيق 3 لغات (مثل السلايد القادم)
وتقول لهو هو RL أو Non-RL وما هو السبب؟

Finite and Infinite Languages

- If L is finite then L is regular.
- Infinite language could be regular or non-regular.
- Example of an infinite regular language:

$$L=\{0^n1^m \mid n,m>0\}$$

- Example of infinite non-regular language:

$$L = \{0^n1^m \mid m>n\}$$

Example of an Infinite Non-Regular Language

- The language $L = \{0^n 1^m \mid m>n\}$ is not regular.

- It is not regular because when we reach substring of 1's we have to remember the length of the 0's substring.
- The length is remembered by states - one state for each letter.
- The length of the 0's prefix can be any number.
- A finite automaton has a finite number of states. So it can not remember the length of 0's prefix.

Example2 Non-Regular Languages

Let's take the language $B = \{0^n 1^n \mid n \geq 0\}$

If we attempt to find a DFA that recognizes B, we discover that the machine seems to need to remember how many 0s have been seen so far as it reads the input. Because the number of 0s isn't limited, the machine will have to keep track of an unlimited number of possibilities. But it cannot do so with any finite number of states.

What is a Grammar?

- In linguistics, grammar is the set of structural rules governing the composition of clauses, phrases, and words in any given natural language.

- In computing, A grammar naturally describes the hierarchical structure of most programming language. For example, an if-else statement in Java can have the form

  if *(expression)* *statement* ***else*** *statement*

Context Free Grammar

- In previous lessons we introduced two different, though equivalent, methods of describing languages: finite automata and regular expressions.

- Now we present CFG is a more powerful method for describing languages. CFGs are more general than regular expressions, i.e. there are more languages definable by CFGs.

- Finite Automata **accept** all regular languages and only regular languages many simple languages are non regular: **$\{a^n b^n \mid n >= 0\}$, {w | w a is palindrome}** and there is no finite automata that accepts them.

- Its more general than Regular Expressions, i.e more languages can be defined by CFG including Non-Regular languages

Example of a Context-Free Grammar

- which we call G1:

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

A grammar consists of a collection of substitution rules, also called productions. Each rule appears as a line in the grammar, comprising a symbol and a string separated by an arrow. The symbol is called a variable. The string consists of variables and other symbols called terminals. The variable symbols often are represented by capital letters. The terminals are analogous to the input alphabet and often are represented by lowercase letters, numbers, or special symbols. One variable is designated as the start variable. It usually occurs on the left-hand side of the topmost rule. For example, grammar G1 contains three rules. G1's variables are A and B, where A is the start variable. Its terminals are 0, 1, and #.

Terminals

These symbols are terminals:

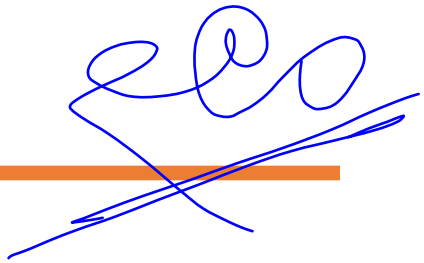
1. Lowercase letters early in the alphabet, such as a, b, c.
2. Operator symbols such as +, *, and so on.
3. Punctuation symbols such as parentheses, comma, and so on.
4. The digits 0,1,.. . ,9.
5. Boldface strings such as **id** or **if**, each of which represents a single terminal symbol.

nonterminals

These symbols are nonterminals:

1. Uppercase letters early in the alphabet, such as A, B, C.
2. The letter S, which, when it appears, is usually the start symbol.
3. Lowercase, italic names such as *expr* or *stmt*.
4. When discussing programming constructs, uppercase letters may be used to represent nonterminals for the constructs. For example, nonterminals for expressions, terms, and factors are often represented by E, T, and F, respectively.

Formal Definitions



- A Context Free Grammar G = (V, ∑, S, P) is given by

1. V a finite set of nonterminal symbols.
2. ∑ a finite set of terminals.
3. S a start symbol, S ε V .
4. P is a set of rules or productions.

- The grammar is considered "context free“, because its production rules can be applied regardless of the context of a nonterminal.
- The single nonterminal on the left hand side can always be replaced by the right hand side.

Context Free Language

- Every grammar defines a language.

  - Languages generated by context-free grammars are known as Context Free Language (CFL).

  - Different Context Free Grammars can generate the same context free language.

  - A language is context-free if there is a context-free grammar with $L = L(G)$

CFG Derivation

- A derivation is a sequence of rewriting operations that starts with the string *S and then repeats driving the string until it contains only terminals:*

Example:

S → SS | aSb | ε

One derivation from the above CFG is “abaabb”

S → SS → aSbS → abS → abaSb → abaaSbb → abaabb

Parse Tree

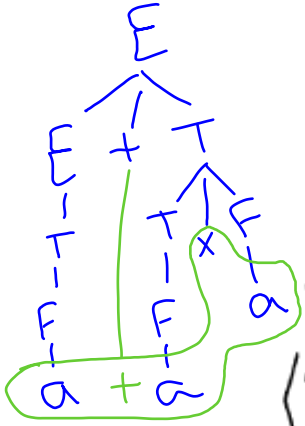
Every derivation can be represented by a ***parse tree:***

- Parse trees are trees labeled by symbols of a particular CFG.
- Leaves: labeled by a terminal or ε.
- Interior nodes: labeled by a variable.

  Children are labeled by the right side of a production for the parent.
- Root: must be labeled by the start symbol.

Example 1

- The two strings **a+a x a**, and **(a+a) x a** can be generated with the following grammar.

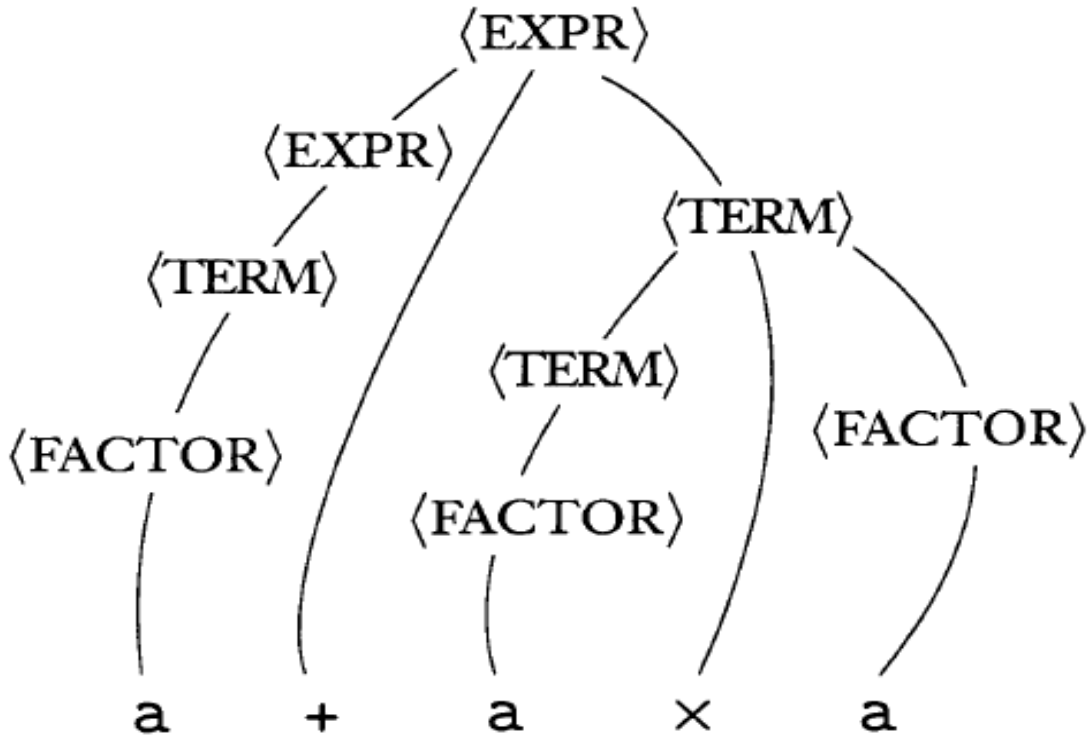


$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle$$

$$\langle \text{TERM} \rangle \rightarrow \langle \text{TERM} \rangle \text{x} \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle$$

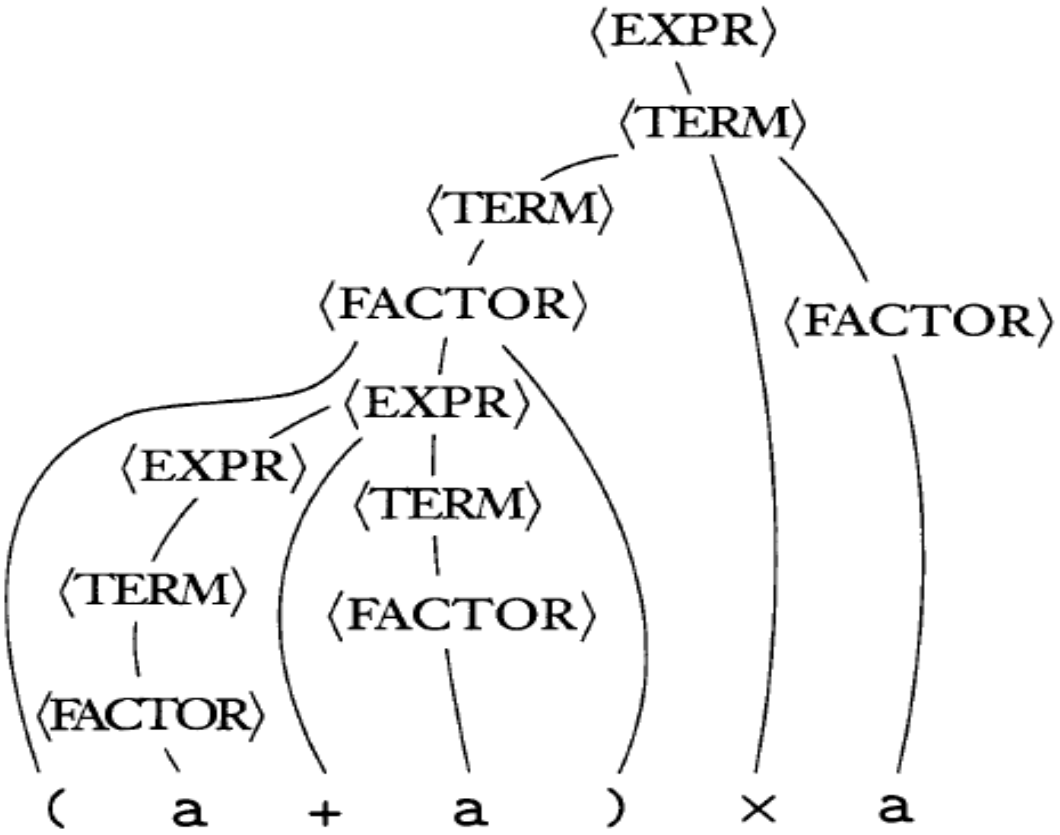
$$\langle \text{FACTOR} \rangle \rightarrow (\langle \text{EXPR} \rangle) \mid \text{a}$$

Example1..cont'd



Parse tree for string
a + a x a

Example1...cont'd



Parse tree for string
(a + a) x a

Example 2

CFG:

E → id

E → E + E

E → E * E

E → (E)

derivation:

E

→ E+E

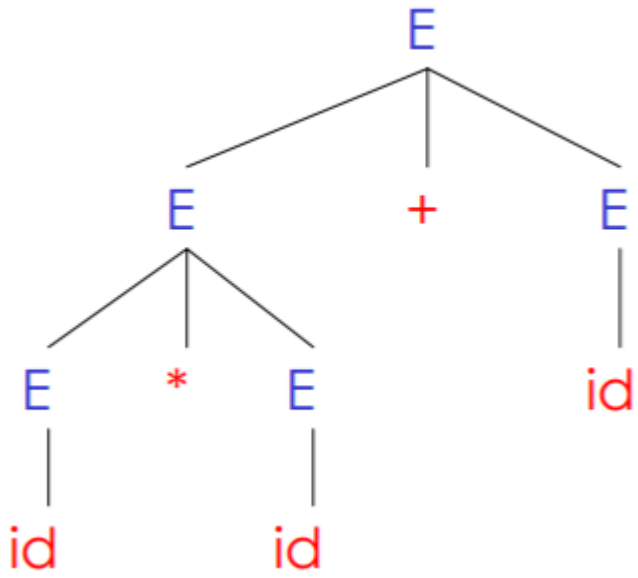
→ E*E+E

→ id*E+E

→ id*id+E

→ id*id+id

Pares Tree for id*id+id



Ambiguous Grammar

- In Computer Science, an **ambiguous grammar** is a grammar for which there exists a string that can have more than one leftmost derivation, while an **unambiguous grammar** is a formal grammar for which every valid string has a **unique** leftmost derivation.

- **Ambiguous grammar** generates more than one **parse tree** as the result of having more than one leftmost derivation for a given String.

Ambiguity problem

- If a grammar can generate a given sentence/string in several different ways, then there will be more than one parse tree for that sentence/string.

- Different parse trees means having several different meanings.

- To allow parsing of programming languages, their grammars have to be unambiguous.

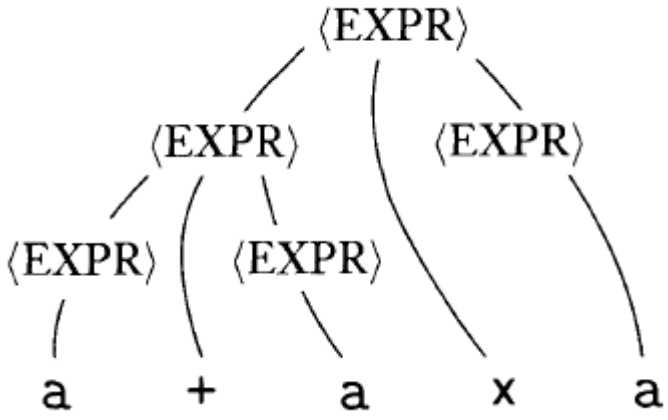
Example 1

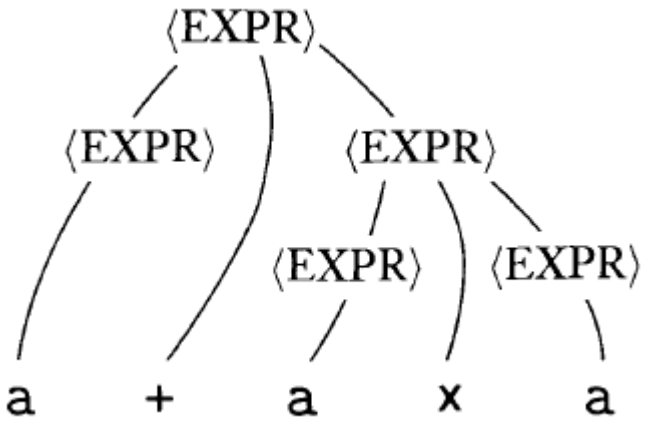
- Consider the following grammar

  Expr ⟶ *number| identifier |Expr + Expr| Expr – Expr*

  Expr ⟶ *Expr * Expr| Expr / Expr| (Expr)*

- This grammar generates the string ***a + a x a*** ambiguously





Example 1…cont'd

- Unambiguous grammar for the same language is

Exp → Exp+Term | Exp-Term | Term

Term → Term*Factor | Term/Factor |Factor

Factor → Number | Identifier | (Exp)

- Now lets try to create parse tree for ***a + a x a***

- This grammar respects precedence rule.

Leftmost and Rightmost Derivation

- For every sentence in a language defined by an unambiguous grammar, there is only one parse tree that generates this sentence.
- There are ***many different derivations corresponding to this parse tree.***
- **Leftmost derivation:** Replaces leftmost non-terminal in each step.
- **Rightmost derivation:** Replaces the rightmost non-terminal in each step.

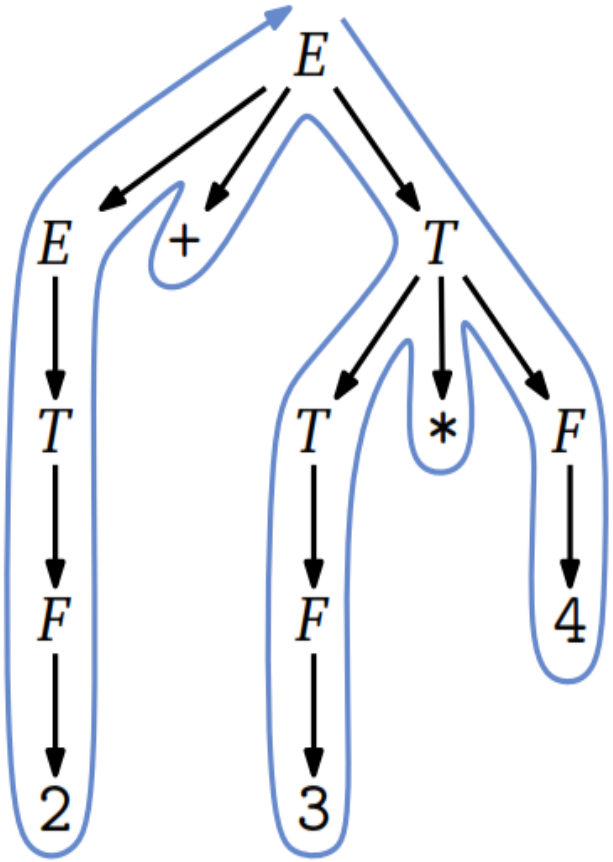
هذا السؤال خاص بسلايد 22

Example 2 Unambiguous Grammar

- Using the unambiguous grammar provided in example1, we will derive 2+3*4

Leftmost

$$
\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow T + T \\
&\Rightarrow F + T \\
&\Rightarrow 2 + T \\
&\Rightarrow 2 + T * F \\
&\Rightarrow 2 + F * F \\
&\Rightarrow 2 + 3 * F \\
&\Rightarrow 2 + 3 * 4
\end{aligned}
$$



rightmost

$$
\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow E + T * F \\
&\Rightarrow E + T * 4 \\
&\Rightarrow E + F * 4 \\
&\Rightarrow E + 3 * 4 \\
&\Rightarrow T + 3 * 4 \\
&\Rightarrow F + 3 * 4 \\
&\Rightarrow 2 + 3 * 4
\end{aligned}
$$

CFG for Programming Languages

- Context-free grammars are used to formally describe the syntax of programming languages.
- Parsing a program involves finding the parse tree of the program.
- Context-free grammars for programming languages must be unambiguous and must capture the program structure.

Designing CFGs

The following are some techniques that might help you in creating your CFGs

- Matching – enforcing the fact that your grammar generates matching pairs of symbols.

Example1: L = $\{0^n 1^n \mid n \geq 0\}$. The CFG for L is

S ⟶ ε

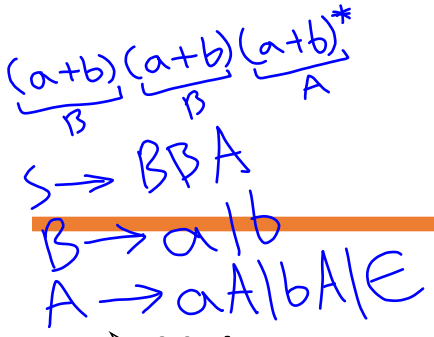
S ⟶ 0S1

The grammar forces that every 0 to match a 1

Example2: L = $\{0^n 1^{2n} \mid n \geq 0\}$. The context-free grammar for L is

S → 0S11 | ε

The grammar forces every 0 to match to 11.



Designing CFGs…cont'd

- Using recursive relationships – when you can represent strings in the language in terms of shorter strings in the language.

Example: L – the language of all strings of balanced brackets. For example,

(()())() ∈ L, but (() ∉ L.

Every string w in L can be viewed as either

- (u), for u ∈ L.

- uv, for u ∈ L, v ∈ L, or

This gives the following grammar:

S ⟶ (S)

S ⟶ SS | ε

Designing CFGs…cont'd

- Combining languages into one, using the rules.

Example1: L1 = $\{0^n1^m \mid n > m\}$, L2= $\{0^n1^m \mid n < m\}$

Then, if X generates L1, and Y generates L2, our grammar is

$$S \longrightarrow X \mid Y$$

L1 is just the language of strings 0^n1^m with one or more extra 0's in front. So,

$$X \longrightarrow 0\ X \mid 0\ E$$
$$E \longrightarrow 0\ E\ 1 \mid \varepsilon$$

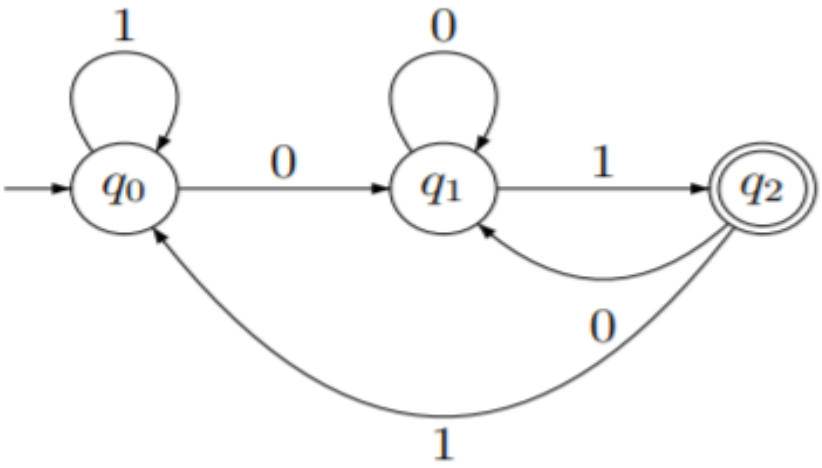
L2 is just the language of strings 0^n1^m with one or more extra 1's in the end. So,

$$Y \longrightarrow Y\ 1 \mid E\ 1$$
$$E \longrightarrow 0\ E\ 1 \mid \varepsilon$$

Designing CFGs…cont'd

- If a language is regular and you can construct a DFA for it, then you can convert the DFA into an equivalent CFG.

EX:



The corresponding grammar would be:

$$q_0 \rightarrow 0q_1 \mid 1q_0$$
$$q_1 \rightarrow 0q_1 \mid 1q_2$$
$$q_2 \rightarrow 0q_1 \mid 1q_0 \mid \varepsilon$$

Note: For each accepting state qf, add a rule qf → ε

Homework

A/ Give context free grammar for the following languages:

1. L= { w ∈ {a,b}* | w contains an odd number of a's }
2. L= length of string is at most 2.
3. strings over { 0,1 } that contain equal number of 0's and 1's
4. If ∑={0,1}Generate CFG for the following
5. B = {w|w starts and ends with the same symbol}
6. C = {w|w has odd length and its middle is 0}
7. D={w|w has even length}