Context Free Grammars

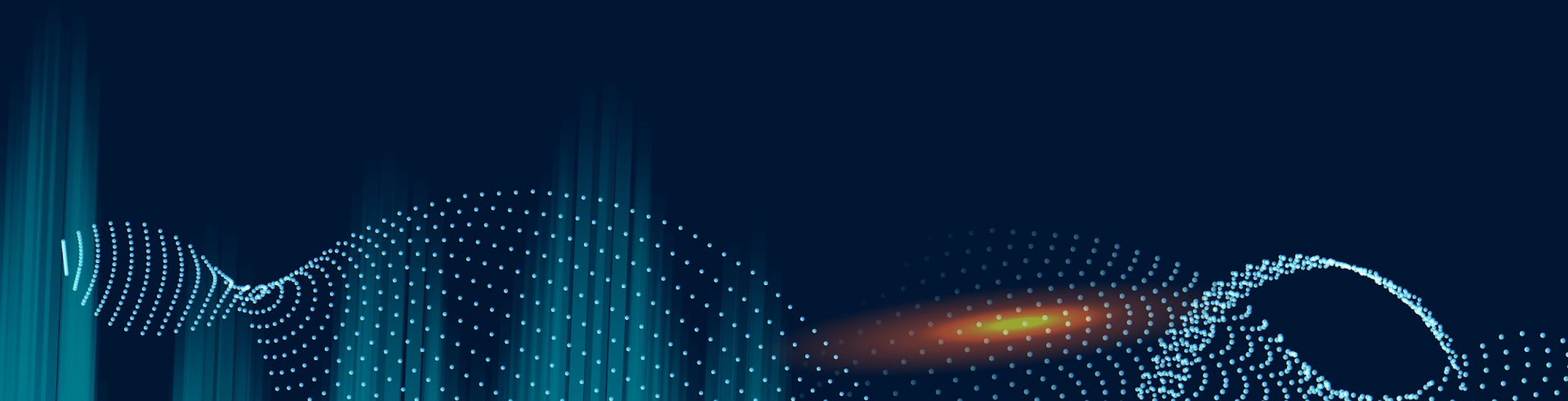
Dr. Gilberto Echeverría
g.echeverria@tec.mx

Based on slides by Dr. Víctor de la Cueva
vcueva@itesm.mx

05 / 2023

How can you tell if a program is written in C / Python / Java / etc.?

What makes them different?



Parts of a language description

LEXICAL

The vocabulary

SYNTAX

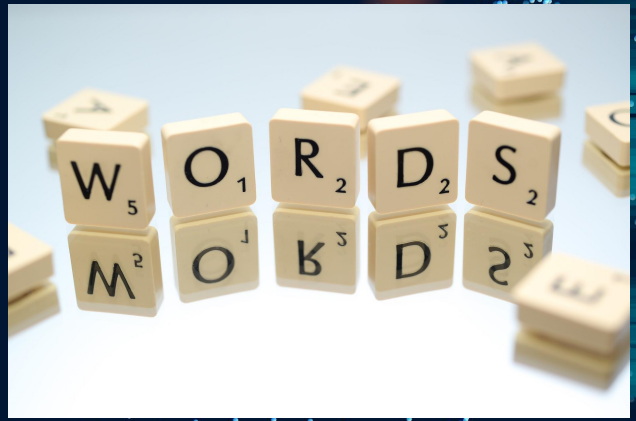
The structure

SEMANTIC

The meaning

Formal language

- A form of communication where primitive symbols (alphabet or vocabulary) and construction rules (grammar) are formally specified
- A valid sequence of symbols is known as a Well-Formed Formula (WFF)
- A formal language is the set of all possible WFFs
- The alphabet must be finite, as well as the length of any WFF. However, the language can have an infinite number of WFFs



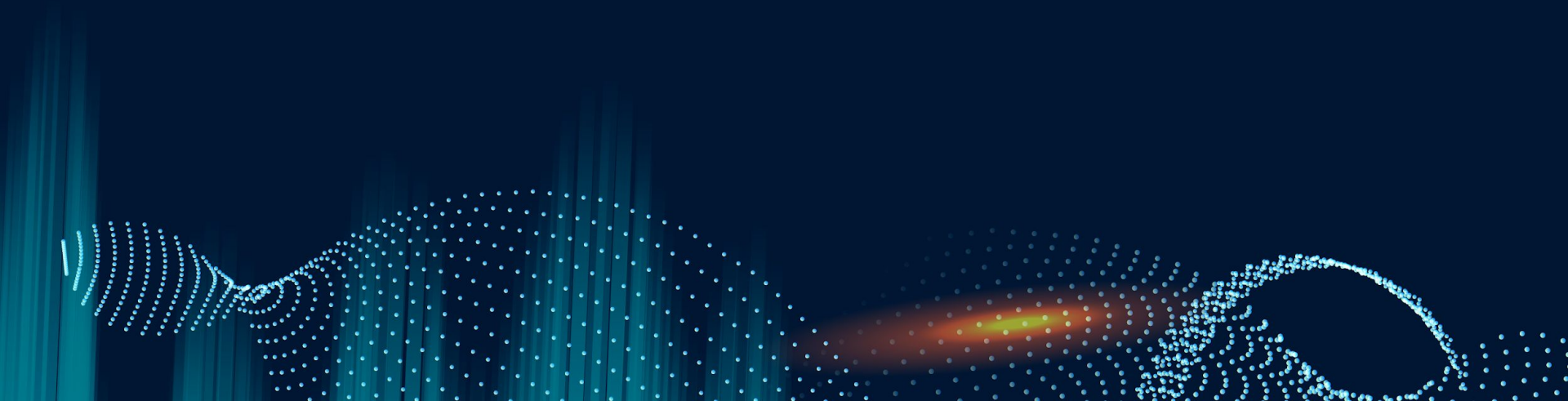
Formal language example

- Alphabet = `{a, b}`
- Grammar: a WFF must have the same number of a's and b's
- WFF examples: `ab, ba, abaabb, …`
- How about describing a programming language?

Formal description tools

- Programming languages use tools to describe each stage: Lexical, Syntax and Semantic
  - Lexical
    - Definition: Regular Expressions
    - Implementation: Finite Auton
  - Syntax
    - Definition: Context-Free Grammars
    - Implementation: Push-Down Auton
  - Semantic
    - No formal definition

Syntax formal representation



Syntax of English sentences

Sentence:

- Subject + Verb + Object

Subject:

- Noun
- Article + Noun
- Article + Adjective + Noun

Object:

- Noun
- Adverb + Noun

https://academicguides.waldenu.edu/writingcenter/grammar/sentencestructure

Specifying a syntax

- Syntax defines how the structures of a certain language are written
- Usually specified using a Context-Free Grammar (CFG), and defined as productions.

Difference from DFA

A CFG is useful to represent nested structures, such as parentheses and brackets in programming languages

Example:

```
for (int i=0; i<10; i++) {
    if (cos(pi) < sqrt(a * (b - c))) {
        printf("Done!\n");
    }
}
```

Context-Free Grammar

A CFG is defined as: **G** = (**V**, **T**, **P**, **S**)

- **V** is a set of non-terminal symbols
- **T** is a set of terminal symbols
- **P** is a set of productions
- **S** is an initial non-terminal symbol, where productions start

Productions

A production consists of:

- a non-terminal on the left side
- the production symbol (→ or ::=)
- a string of terminals and non-teminals on the right

Examples:

```
A → bA
A → c
```

A simple CFG

A CFG to describe simple arithmetic expressions:

G = ({E}, {+, *, (,), id}, P, E)

P =

```
E → E + E
E → E * E
E → (E)
E → id
```

Derivations

Let $A \rightarrow \beta$ be a production in $\mathbf{P}$

Let α, β and γ be strings in $(\mathbf{V} \cup \mathbf{T})^*$

Then $\alpha A \gamma \Rightarrow_G \alpha \beta \gamma$ (The left hand side derives to the right hand side in grammar G)

If $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_m \in (\mathbf{V} \cup \mathbf{T})^*$, m >= 1 and

$\alpha_1 \Rightarrow_G \alpha_2, \alpha_2 \Rightarrow_G \alpha_3, \ldots, \alpha_{m-1} \Rightarrow_G \alpha_m$

Then $\alpha_1 \Rightarrow_G^* \alpha_m$

Language defined by a CFG

The language described by G is L(G):

$\{w \mid w \in T^* \text{ and } S \Rightarrow_G^* w\}$

A string is part of the language if:

- It is composed only of terminal symbols
- It can be derived from the starting symbol S

Exercise:

Can the string 00110101 be generated with the grammar defined by:

G = ({S, A, B}, {0, 1}, P, S)

P =

$$S \rightarrow 0B \mid 1A$$

$$A \rightarrow 0 \mid 0S \mid 1AA \mid \lambda$$

$$B \rightarrow 1 \mid 1S \mid 0BB$$

https://www.youtube.com/watch?v=htoFbcwES28&list=PLBInK6fEyqRgp46KUv4ZY69yXmpwKO1ev&index=72

Exercise:

Can the string aabbb be generated with the grammar defined by:

G = ({S, A}, {a, b}, P, S)

P =

$S \rightarrow aAb$

$A \rightarrow aAb \mid \lambda$

https://www.youtube.com/watch?v=htoFbcwES28&list=PLBInK6fEyqRgp46KUv4ZY69yXmpwKO1ev&index=72

Representation of a CFG

Some standards to define a CFG are:

- Backus-Naur Form (BNF)
- Extended Backus-Naur Form (EBNF)
- Syntax diagrams (visual representation)

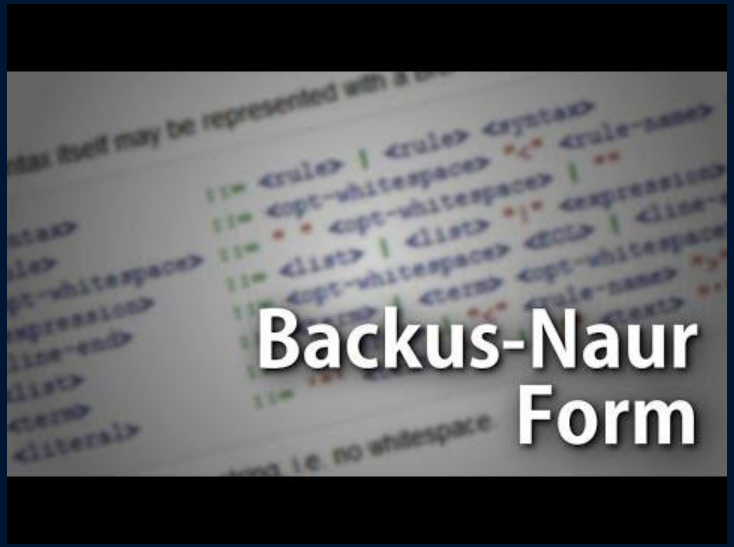
Backus-Naur Form (BNF)

- Defined as [John Backus, 1960]:
  - Non-terminal symbols are enclosed in ‘<‘ and ‘>’
  - Terminal symbols stand alone
  - Productions are indicated with ::=
  - Optional productions are separated by |

- Improved in 1963 by Peter Naur

Source:
https://en.wikipedia.org/wiki/John_Backus
https://en.wikipedia.org/wiki/Peter_Naur

Backus-Naur Form



Example: CFG for English sentences

<sentence> ::= <noun phrase> <verb phrase>

<noun phrase> ::= <adjective> <noun phrase>

<noun phrase> ::= <noun>

<verb phrase> ::= <verb>

<noun> ::= boy

<adjective> ::= Little

<verb> ::= eats

Example: BNF of real numbers

```
<real-number> ::= <digit-sequence> . <digit-sequence>

<digit-sequence> ::= <digit> | <digit> <digit-sequence>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Exercise

- Create the BNF for an arithmetic expression:
  - Allow operators +, -, * and /
  - Allow the use of parentheses
  - An expression can have any number of operators
  - Operands can be numbers or variables

An initial solution

```
<expression> ::=
        <expression> + <expression> |
        <expression> - <expression> |
        <expression> * <expression> |
        <expression> / <expression> |
        ( <expression> ) | <element>

<element> ::= <variable> |
<constant>
```

The problem with this grammar is that it is ambiguous. It can produce more than one parse tree for certain expressions

Derivation trees

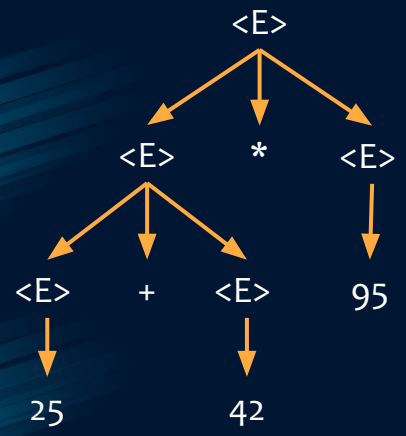
Two ways to derive a string:

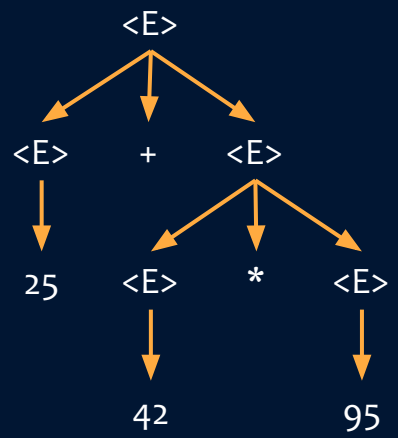
- Leftmost: Deriving the first non terminal that is found to the left
- Rightmost: Deriving first the non terminal found to the right

https://beautifulracket.com/bf/grammars-and-parsers.html

Derivation trees

- The tree structure generated by the CFG when parsing an input
- Example: Parse tree for the expression `25 + 42 * 95`





If a grammar can generate more than one parse tree, it is ambiguous

An improved solution

```
<expression> ::=    <expression> + <term> |
                    <expression> - <term> |
                    <term>

<term> ::=    <term> * <factor> |
              <term> / <factor> |
              <factor>

<factor> ::= (<expression>) |
             <variable> |
             <constant>
```

Extended BNF (EBNF)

- Uses notions of Regular Expressions to simplify BNF statements
- The rules are:
  - Non-terminal symbols are written in UPPERCASE.
  - Terminal symbols of a single character are enclosed in single quotations. Example: ‘+’, ‘-’
  - Longer terminals symbols don’t need quotations
  - Terminal for language keywords are written in **bold**
  - |, represents alternatives (or)
  - ( ), used to group alternatives
  - { }, represent one or more
  - [ ], represent an optional construction

Exercise

- Create the EBNF for an arithmetic expression:
  - Allow operators +, -, * and /
  - Allow the use of parentheses
  - An expression can have any number of operators
  - Operands can be numbers or variables

EBNF solution

```
EXPRESSION ::=   TERM [ { ( ‘+’ | ‘-’ ) TERM } ]

TERM ::= FACTOR [ { (‘*’ | ‘/’ ) FACTOR } ]

FACTOR ::= ‘(‘ EXPRESSION ‘)’ | VARIABLE | CONSTANT
```

Exercise:

Create the leftmost and rightmost parse trees for the string aabaa using the following grammar:

G = ({S, A}, {a, b}, P, S)

P =

$$S \rightarrow aAS \mid aSS \mid \lambda$$

$$A \rightarrow SbA \mid ba$$

https://www.youtube.com/watch?v=u4-rpIlV9NI&list=PLBInK6fEyqRgp46KUv4ZY69yXmpwKOIev&index=73

Exercise:

Define the grammar to write a correct if / elif / else structure in Python:

Real examples of grammars:

https://docs.python.org/3/reference/grammar.html

https://docs.racket-lang.org/dssl2/Formal_grammar.html

Useless symbols

A symbol X is useful if there is a derivation

$S \Rightarrow_G^* \alpha X \gamma \Rightarrow_G^* w$

where $w \in T^*$ and $\alpha, \gamma \in (\mathbf{V} \cup \mathbf{T})^*$

X is useful if:

- it can be derived from S
- a terminal string can be derived from it

Simplification of a CFG

1. Identify the non terminals that can produce some terminal symbol, and the non terminals that produce them
2. Identify the symbols that can be produced from the start symbol

Eliminate the non terminals that are not found in the previous two steps

Useless symbols

Useless symbols can be removed from the grammar

Any productions that use them can be eliminated as well

Exercise:

Determine a simplified version of the following grammar:

G = ({S, A, B, C, E}, {a, c, e}, P, S)

P =

S → AC | B

A → a

C → c | BC

E → aA | e

https://www.youtube.com/watch?v=EF09zxzpVbk&list=PLBlnK6fEyqRgp46KUv4ZY69yXmpwKOIev&index=75

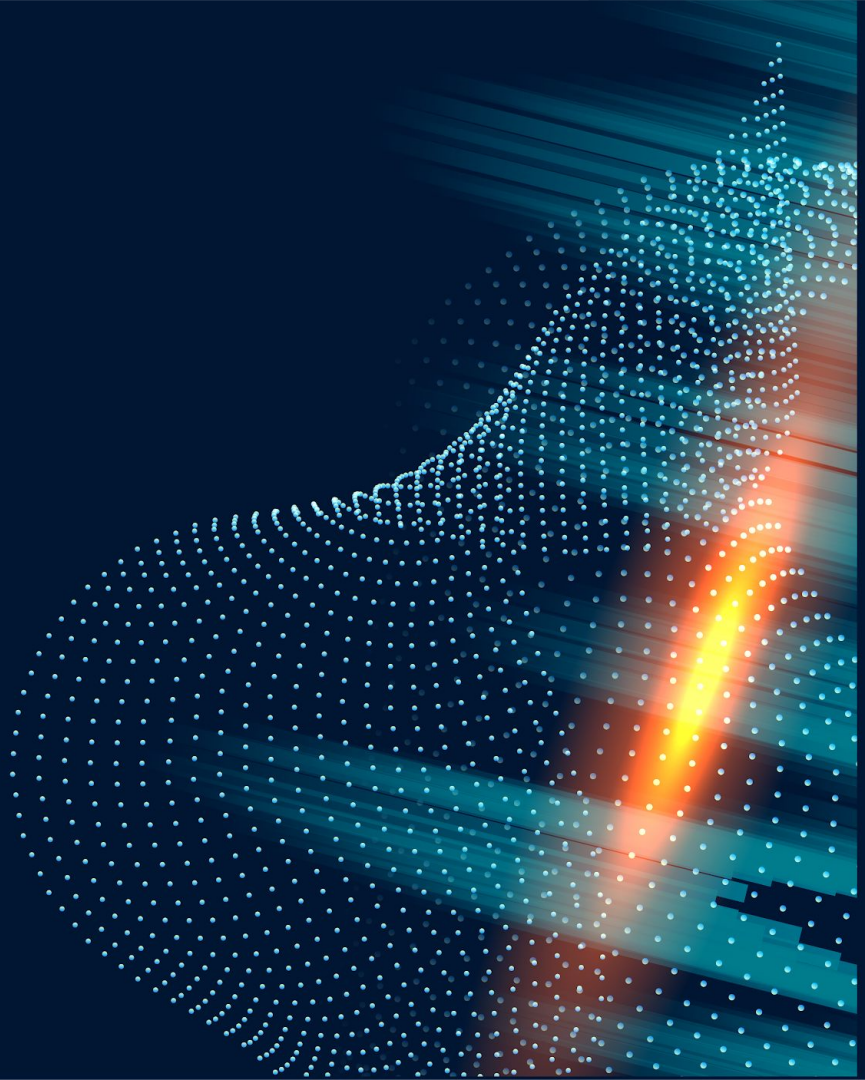
The grammar defines the valid constructions in a language

Try to identify the patterns in new languages you learn

THANKS!

Do you have any questions?
g.echeverria@tec.mx

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.



References

- R. Sethi. Programming Languages: concepts and constructs. Addison-Wesley, 2nd edition (1996).
- K.C. Louden and K. A. Lambert. Programming Languages: Principles and Practice. Cengage 3rd edition (2011).
- R.W. Sebasta. Programming Languages. 3rd ed. Pearson (2012).
- T.A. Sudkamp. Languages and Machines: An Introduction to the Theory of Computer Science. Pearson, 3rd Edition (2005).
- J.E. Hopcroft, R. Motwani, J.D. Ullman. Introduction to Automata Theory, Languages, and Computation. Pearson, 3rd Edition (2006)

References

- http://www.cs.kent.edu/~batcher/CS453111/topic2.html
- https://opendsa-server.cs.vt.edu/ODSA/Books/PL/html/Grammars3.html
- https://docs.python.org/3/reference/grammar.html
- https://beautifulracket.com/bf/grammars-and-parsers.html