Computer Science III
Semester 2025-I
Workshop No. 1 — Theory of the Computation

Eng. Carlos Andrés Sierra, M.Sc.
Computer Engineering
Universidad Distrital Francisco José de Caldas

Welcome to the first workshop of the *Computer Science III* course! This workshop focuses on **theory of the computation** for: an *finite-state machines*. By exploring the principles of *regular expressions*, *context-free grammars*, and *Turing machines*, you will gain a deeper understanding of the theoretical foundations of computer science.

Workshop Scope and Objectives:

- **Finite-State Machines:** You will learn how to define finite-state machines for specific languages, and how to derive regular expressions from them.
- **Regular Expressions:** You will explore the relationship between regular expressions and finite-state machines, and how to construct generative grammars from regular expressions.
- **Context-Free Grammars:** You will learn how to define context-free grammars for specific languages, and how to derive derivation trees from them.
- **Derivation Trees:** You will practice constructing derivation trees for specific strings generated by context-free grammars.
- **Real Numbers and Identifiers:** You will explore the grammar for real numbers and identifiers.

Carlos Andrés Sierra, Computer Engineer, M.Sc. in Computer Engineering, Titular Professor at Universidad Distrital Francisco José de Caldas.

Any comment or concern about this document can be sent to Carlos A. Sierra at: *cavirguezs@udistrital.edu.co*.

1. For each of the following `languages`, define the corresponding **finite-state machine**:

   (i) $\Sigma = \{0,1,2\}$. $L = (01^*2 \cup 2102)^*101(01 \cup 12 \cup 20)^*$.

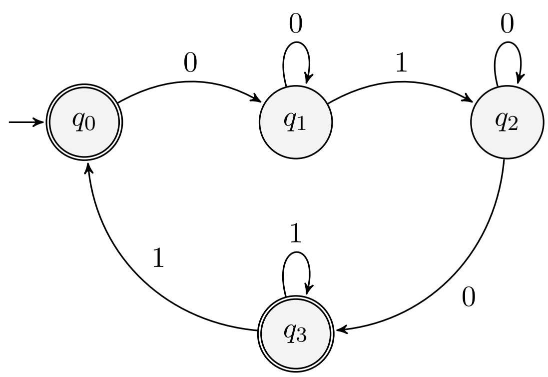
   (ii) $\Sigma = \{a,b,c\}$. $L = (abc \cup bca \cup cab)(abc \cup bca \cup cab)^*$.

   (iii) $\Sigma = \{a,b,c\}$. $L = (abc \cup bca \cup cab)^*(abc \cup bca \cup cab)$.

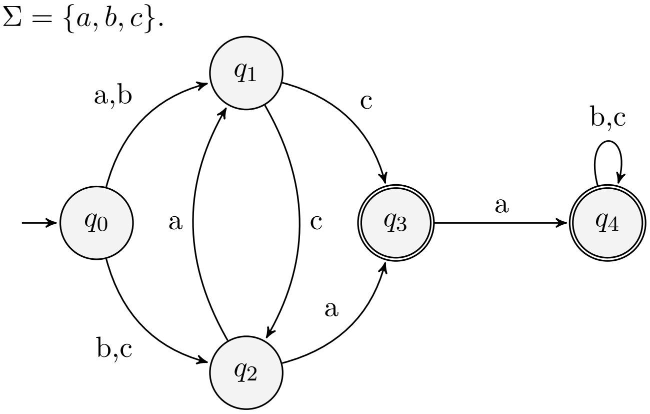
   (iv) $\Sigma = \{0,1,2\}$. $L = (01^*2 \cup 10^*2 \cup 21^*0)^*(01 \cup 12 \cup 20)^*101$.

2. For each one of the following `finite-state machines`, define the corresponding **regular expression** and a `generative grammar`:

   (i) $\Sigma = \{0,1\}$.



   (ii) $\Sigma = \{a,b,c\}$.



3. For each of the following `regular expressions`, define the corresponding **generative grammar** (all over the alphabet $\Sigma = \{a,b,c,d\}$):

   (i) $\{a^i b^j c^j d^i : i,j \geq 1\}$.

   (ii) $\{a^i b^i c^j d^j : i,j \geq 1\}$.

   (iii) $\{a^i b^j c^j d^i : i,j \geq 1\} \cup \{a^i b^i c^j d^j : i,j \geq 1\}$.

   (iv) $\{a^i b^j c^{i+j} : i \geq 0, j \geq 1\}$.

4. Let G a `context-free grammar` with the following productions:

$$G = \begin{cases} S \to ABC \mid BaC \mid aB \\ A \to Aa \mid a \\ B \to BAB \mid bab \\ C \to cC \mid \lambda \end{cases}$$

   Find derivation trees for the following strings:

   (i) $w_1 = abab$.
   (ii) $w_2 = babacc$.
   (iii) $w_3 = abababab c$.

5. As follows there is a `context-free grammar` to generate **real numbers** without sign, the alphabet is $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., +, -, E\}$:

   `<real>` $\to$ `<digits> <decimal> <exp>`
   `<digits>` $\to$ `<digit> <digits>` | `<digit>`
   `<digit>` $\to$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
   `<decimal>` $\to$ `<digits>` | λ
   `<exp>` $\to$ E`<digits>` | E+`<digits>` | E-`<digits>` | λ

   Define the derivation tree for the following strings:

   (i) $w_1 = 47.236$
   (ii) $w_2 = 321.25E + 35$
   (iii) $w_3 = 0.8E9$
   (iv) $w_4 = 0.8E + 9$

6. The following is a `context-free grammar` to generate **identifiers**, identifiers are strings of letters and digits, starting with a letter:

   `<identifier>` $\to$ `<letter> <lsds>`
   `<lsds>` $\to$ `<letter> <lsds>` | `<digit> <lsds>` | λ
   `<letter>` $\to$ a | b | c | ... | x | y | z | A | B | C | ... | X | Y | Z
   `<digit>` $\to$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

   Draw the derivation tree for the following names:

   (i) $w_1 = MyVariable$
   (ii) $w_2 = temp2$
   (iii) $w_3 = string2int$
   (iv) $w_4 = 2NotAVariable$

7. For each of the following cases, define a regular expression as used in a compiler based on the Python `re` library:

   (i) **Identifier:** A regular expression to match valid identifiers (variable names, function names, etc.).

   (ii) **Integer Literal:** A regular expression to match integer literals.

   (iii) **Floating Point Literal:** A regular expression to match floating-point literals.

   (iv) **String Literal:** A regular expression to match string literals enclosed in double quotes.

   (v) **Single-line Comment:** A regular expression to match single-line comments starting with '//'.

   (vi) **Multi-line Comment:** A regular expression to match multi-line comments enclosed in '/* */'.

   (vii) **Whitespace:** A regular expression to match whitespace characters (spaces, tabs, newlines).

   (viii) **Operators:** A regular expression to match common operators (e.g., '+', '-', '*', '/', '==', '!=').

   (ix) **Keywords:** A regular expression to match reserved keywords (e.g., 'if', 'else', 'while', 'return').

   (x) **Hexadecimal Literal:** A regular expression to match hexadecimal literals.

8. Let G a `context-free grammar` with the following productions:

```
S -> Program
Program -> StatementList
StatementList -> Statement StatementList | <lambda>
Statement -> Assignment | IfStatement | WhileStatement | ReturnStatement
Assignment -> Identifier "=" Expression ";"
IfStatement -> "if" "(" Expression ")" "{" StatementList "}" ElsePart
ElsePart -> "else" "{" StatementList "}" | <lambda>
WhileStatement -> "while" "(" Expression ")" "{" StatementList "}"
ReturnStatement -> "return" Expression ";"
Expression -> Term Expression'
Expression' -> "+" Term Expression' | "-" Term Expression' |
<lambda>
Term -> Factor Term'
Term' -> "*" Factor Term' | "/" Factor Term' | <lambda>
Factor -> "(" Expression ")" | Identifier | Number
Identifier -> [a-zA-Z_][a-zA-Z0-9_]*
Number -> [0-9]+
```

Explanation:

- **S** is the start symbol.

- **Program** consists of a list of statements.
- **StatementList** is a sequence of statements or an empty sequence ($< lambda >$).
- **Statement** can be an assignment, an if statement, a while statement, or a return statement.
- **Assignment** assigns an expression to an identifier.
- **IfStatement** includes an optional else part.
- **WhileStatement** represents a while loop.
- **ReturnStatement** returns an expression.
- **Expression** consists of terms combined with addition or subtraction.
- **Term** consists of factors combined with multiplication or division.
- **Factor** can be an expression in parentheses, an identifier, or a number.
- **Identifier** matches typical variable names.
- **Number** matches sequences of digits.

Based on the provided context-free grammar, create derivation trees for the following statements:

(a) **Exercise 1:**

```
x = 5 + 3 * 2;
```

(b) **Exercise 2:**

```
if (x > 0) {
    y = x - 1;
} else {
    y = 0;
}
```

(c) **Exercise 3:**

```
while (x < 10) {
    x = x + 1;
}
```

(d) **Exercise 4:**

```
return (a + b) * c;
```

Deadline: Wednesday, May 14th, 2025, 6:00. Submissions after this deadline may incur penalties in accordance with course policies.

Good luck, and remember: this workshop is your starting point for conceptualizing and designing a compiler.