Programming Languages Foundations

Computer Science III

Author: Eng. Carlos Andrés Sierra, M.Sc.
cavirguezs@udistrital.edu.co

Lecturer
Computer Engineering
School of Engineering
Universidad Distrital Francisco José de Caldas

2025-I

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Babbage Machine

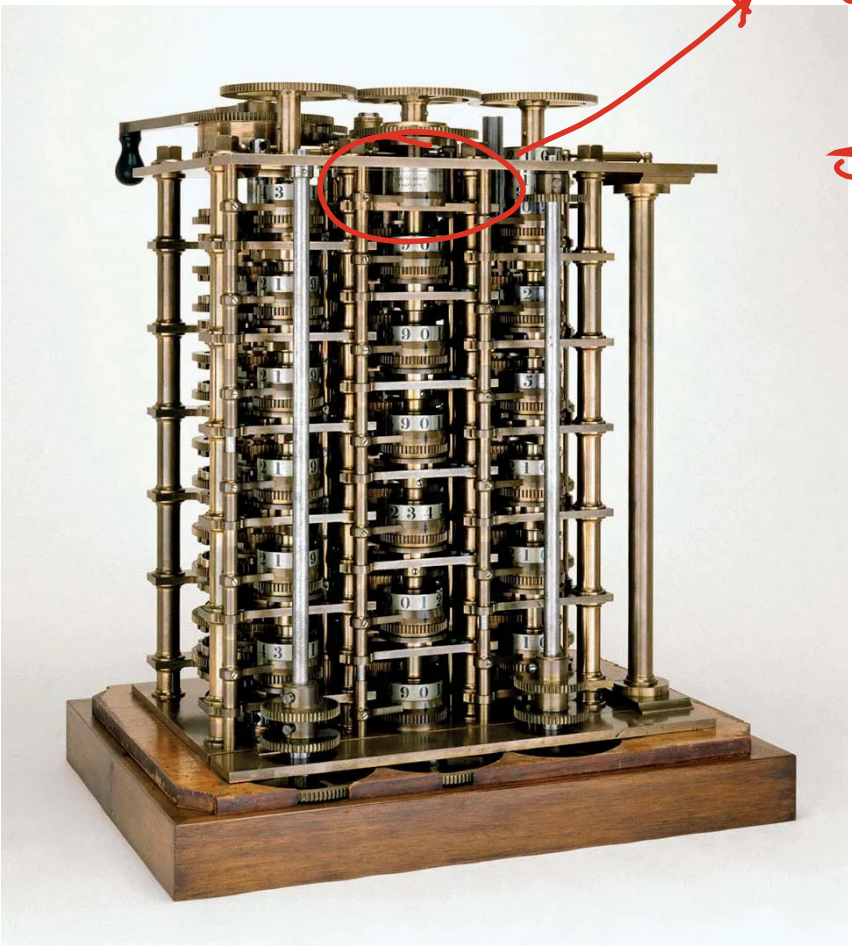


Figure: Analytical Machine

- **Charles Babbage** (1791 — 1871) was an English mathematician, philosopher, inventor, and mechanical engineer.
- He originated the **concept** of a digital programmable computer.
- Considered the "*father of the computer*". He creates the Analytical Engine.
- The **Analytical Engine** was a general-purpose mechanical computer.

Ada Lovelace

- **Ada Lovelace** (1815 — 1852) was an English mathematician and writer.
- She is known for her work on Charles Babbage's early mechanical general-purpose computer, the Analytical Engine.
- She was the **first** to recognize that the machine had applications beyond pure calculation, and to have published the first algorithm intended to be carried out by such a machine.

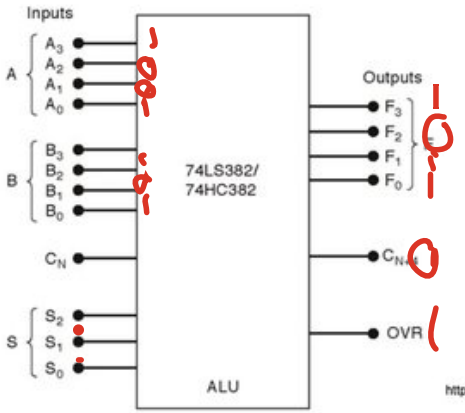
Figure: Ada Lovelace

Physical Binary Language

- **Binary** is a base-2 number system. It uses only two symbols: typically 0 (zero) and 1 (one).
- The **bit** is the basic unit of information in computing and digital communications.

Arithmetic Logic Unit

- Examine the functionality of this 74LS382 ALU chip



Function Table

S_2	S_1	S_0	Operation	Comments
0	0	0	CLEAR	$F_3F_2F_1F_0 = 0000$
0	0	1	B minus A	Needs $C_N = 1$
0	1	0	A minus B	Needs $C_N = 1$
0	1	1	A plus B	Needs $C_N = 0$
1	0	0	A ⊕ B	Exclusive-OR
1	0	1	A + B	OR
1	1	0	AB	AND
1	1	1	PRESET	$F_3F_2F_1F_0 = 1111$

Notes: S inputs select operation.
OVR = 1 for signed-number overflow.

datasheet

http://pdf.datasheetcatalog.com/datasheet/nationalsemiconductor/DS009529.PDF

A = 4-bit input number
B = 4-bit input number
C_N = carry into LSB position
S = 3-bit operation select inputs

F = 4-bit output number
C_{N+4} = carry out of MSB position
OVR = overflow indicator

ECE331: MIPS Instructions-I 13

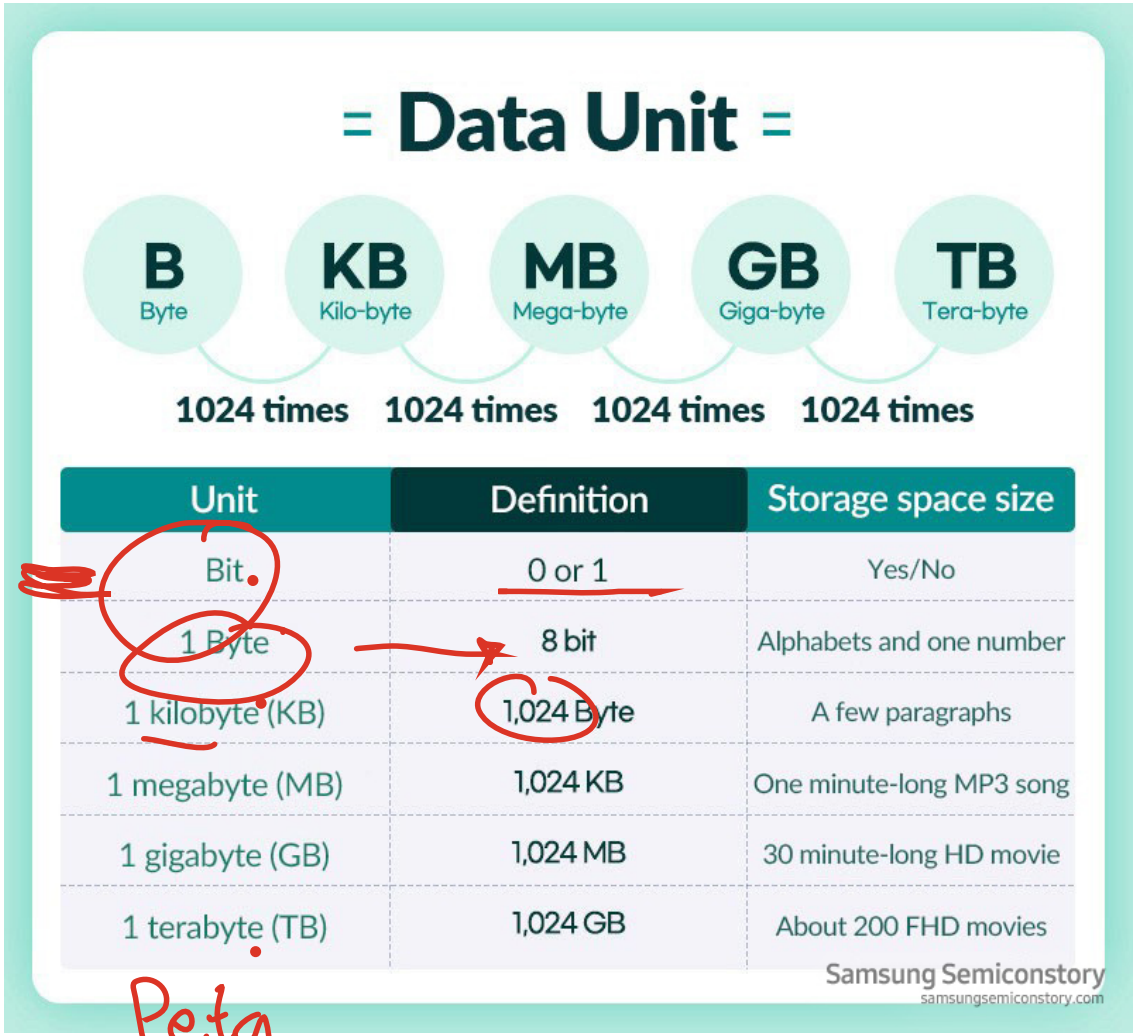
Bits and Bytes

$2^{10} = 1024$

$2^8 = 256$

ASCII

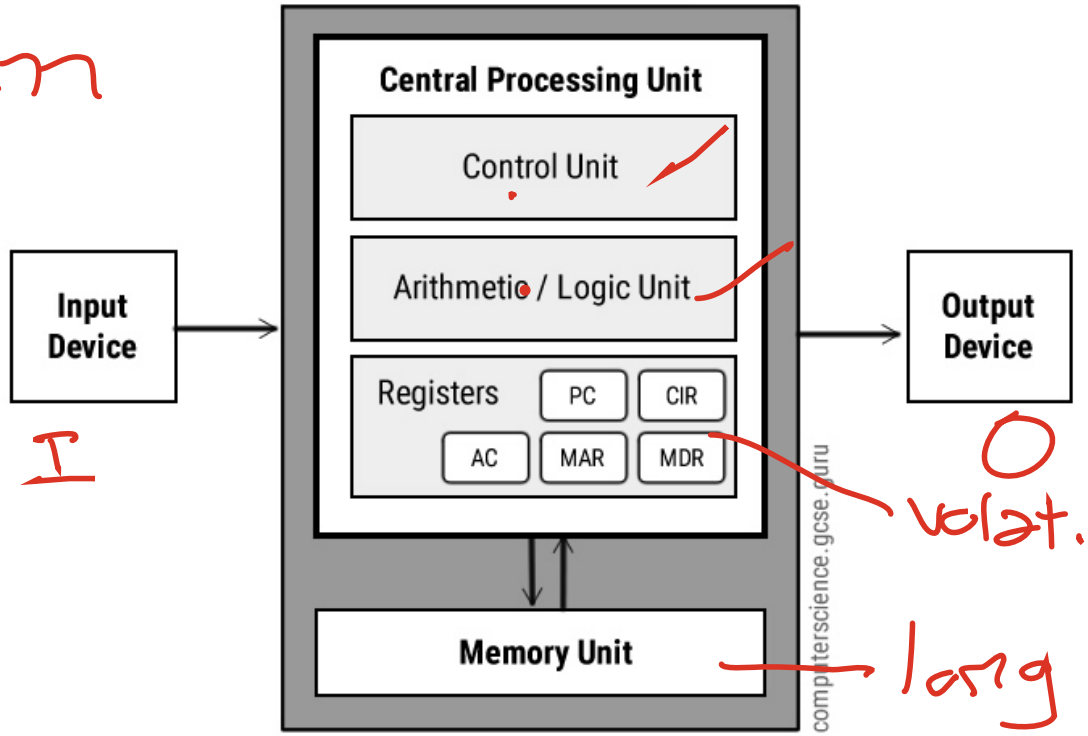
0 - 255



Peta

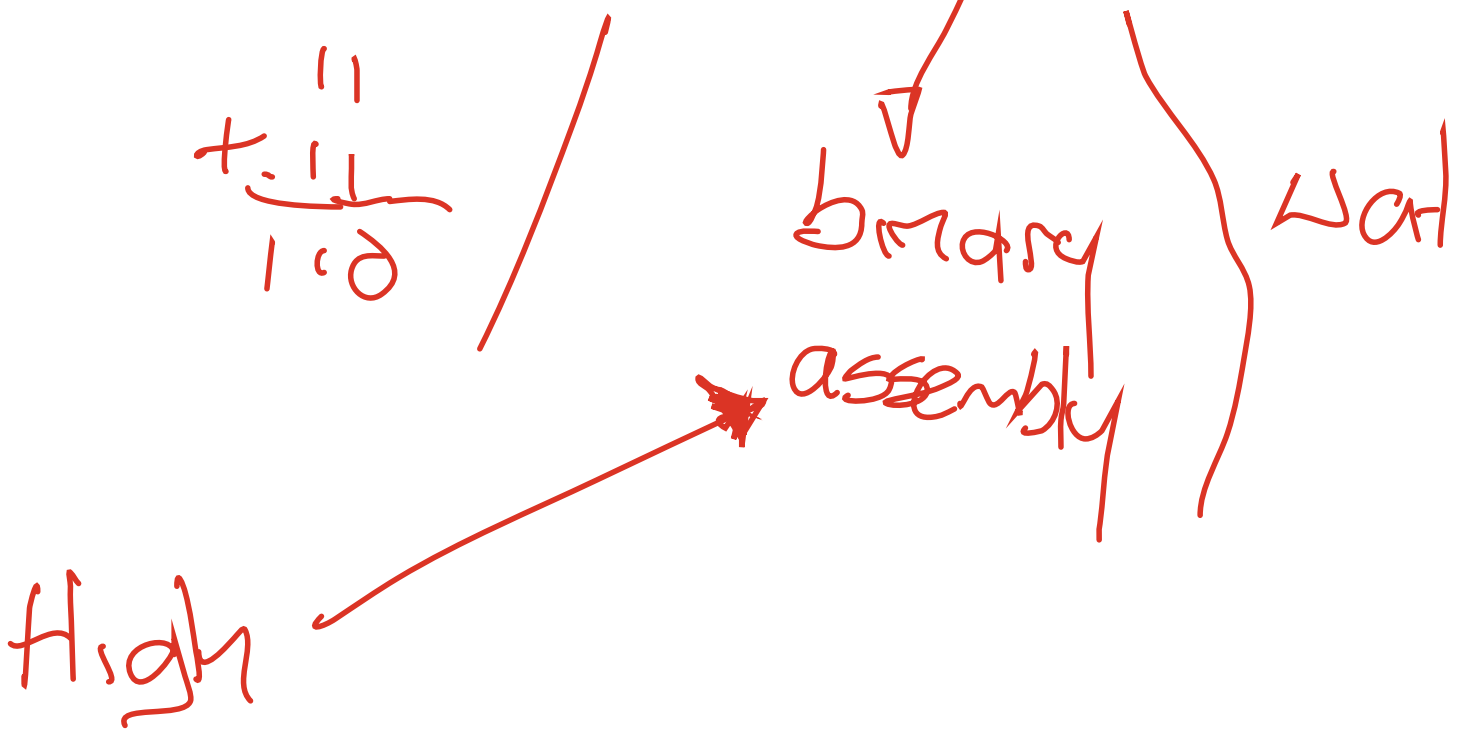
Von Neumann Architecture

- The **Von Neumann Architecture** is a computer architecture based on the stored-program computer concept.
- The **design** is based on the concept of an instruction set.
- The **program** and **data** are both stored in the same memory unit.

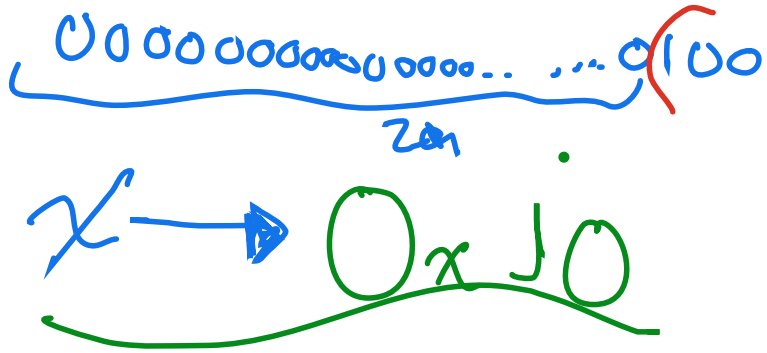


Bit Operations

- **Bitwise operations** are operations that directly manipulate bits.
- They are used in low-level programming for performing calculations, file processing, and data compression.



Memory and Bit Storage



→ HEAP-SIZE

Garbage Collector → scheduled

Outline

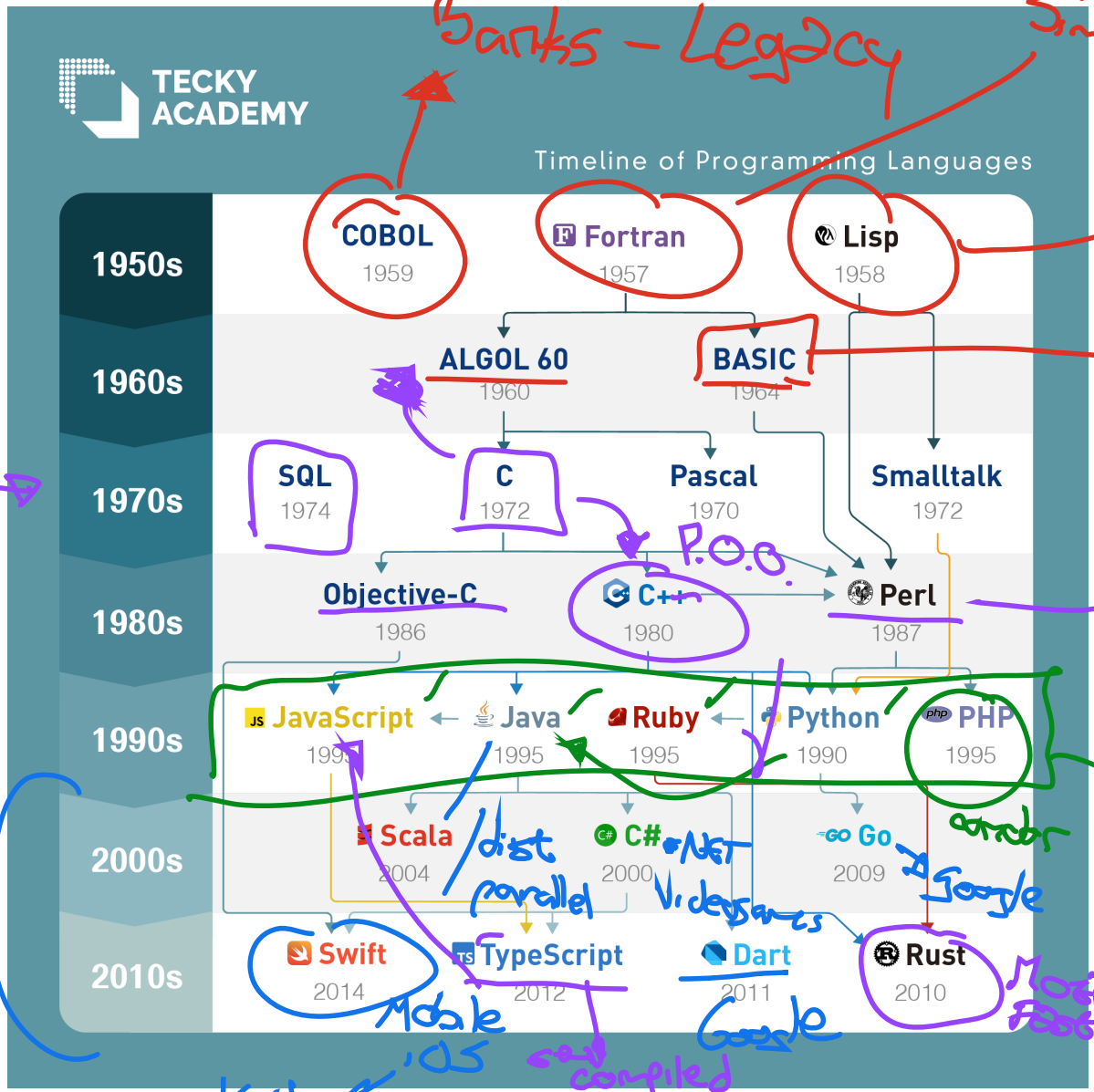
UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Assembly Programming Language

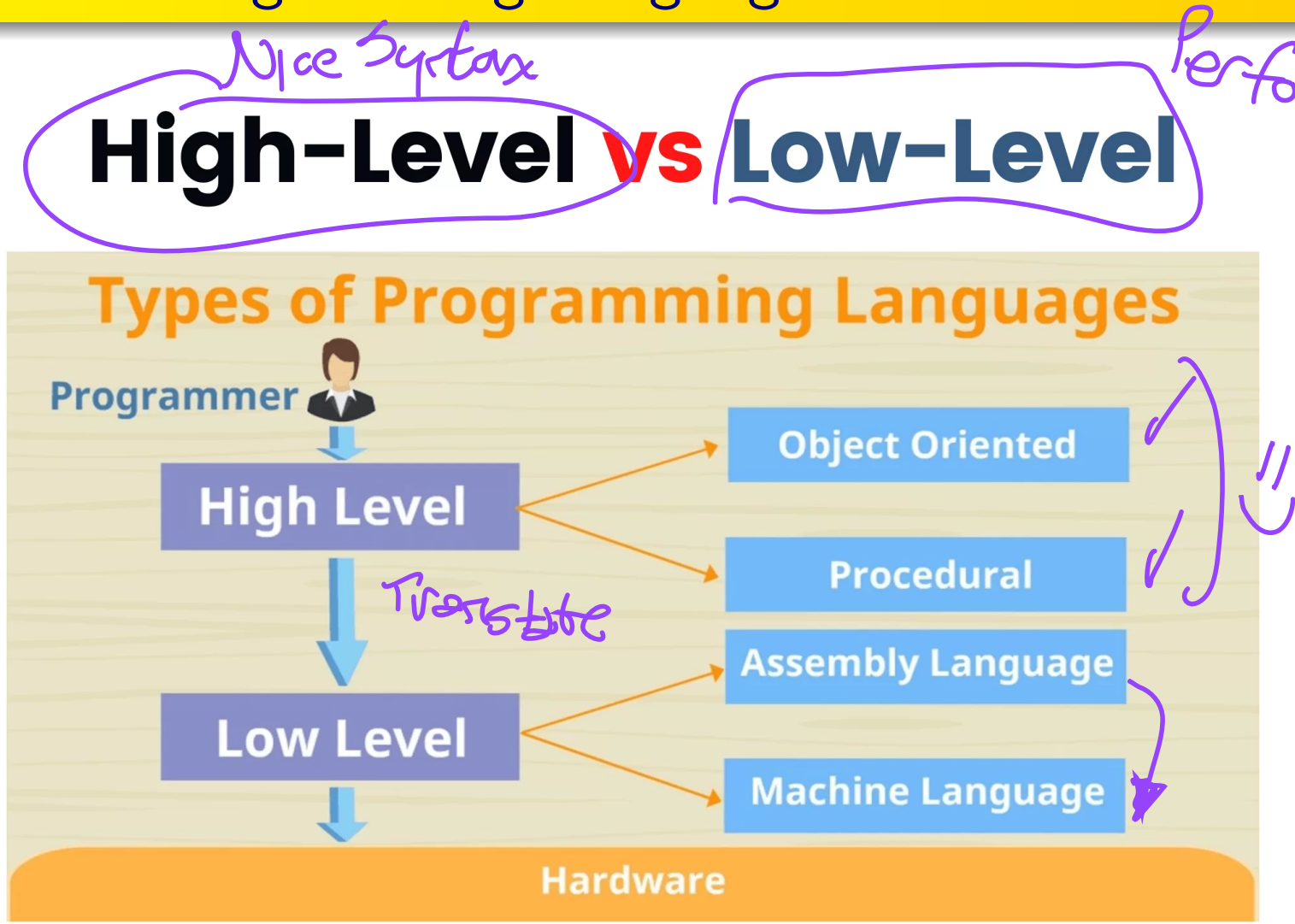
```
1  ; Example of a basic conditional structure in x86 assembly
2
3  cmp eax, 10     ; Compare the value in eax with 10
4  je equal_label  ; Jump to equal_label if eax is equal to 10
5
6  ; Code for not equal case
7  jmp end_label   ; Jump to end_label to avoid executing the equal case code
8
9  equal_label:
10 ; Code for equal case
11
12 end_label:
13 ; Continue execution
14
```

Listing 1: Basic Conditional Structure in Assembly

History of Programming Languages



High-Level Programming Languages I



High-Level Programming Languages II

Purpose of High-Level Languages

Ease of Use

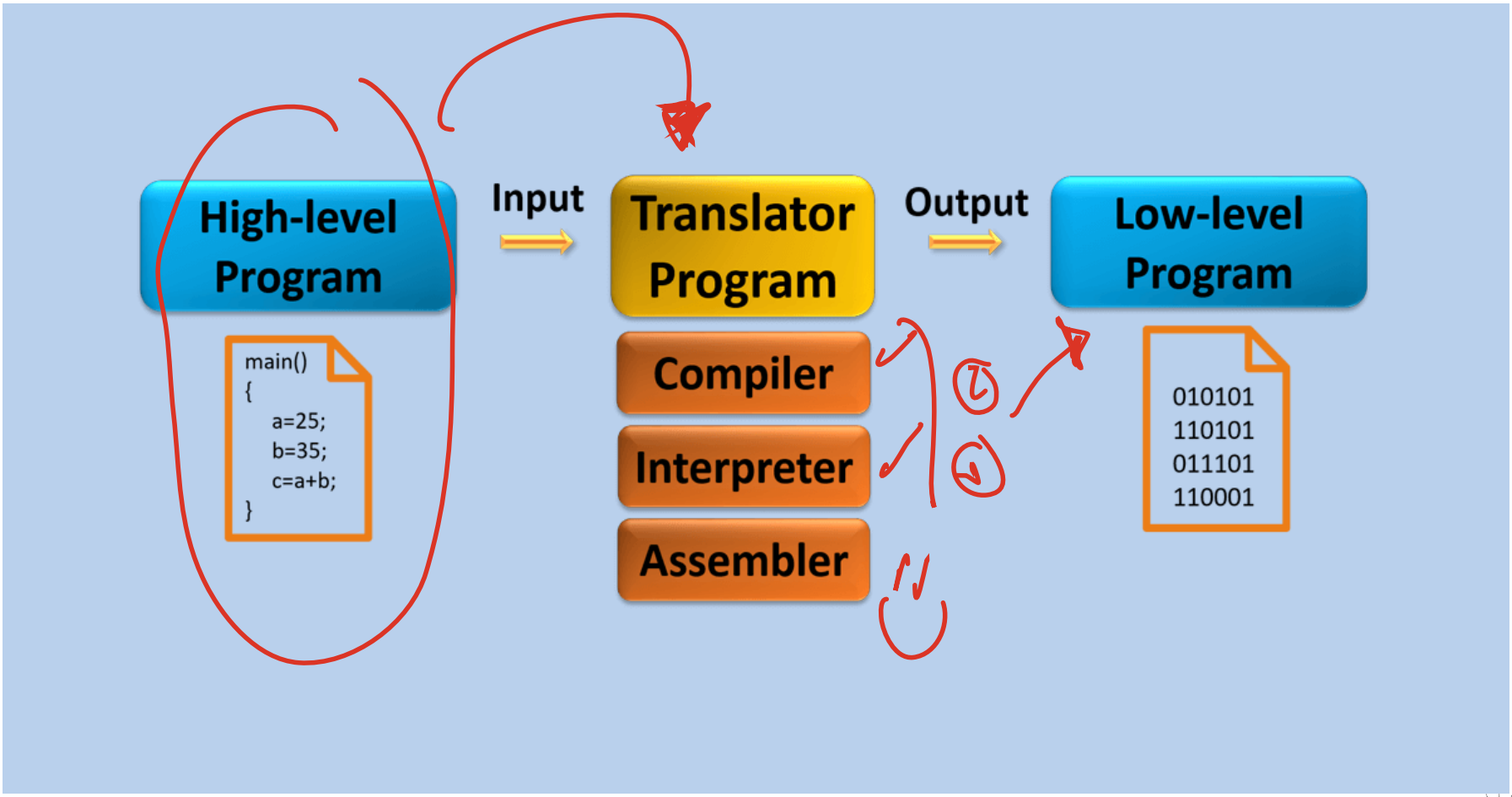
- Simplifying programming
- Minimizing learning curve
- Enhancing productivity
- Automated memory management
- Clear syntax
- Readability and maintainability

Portability Across Systems

- Cross-platform compatibility
- Utilization of compilers and interpreters
- Seamless execution on various platforms
- Reduction of platform-specific modifications
- Enhanced flexibility across environments

Techopedia

Translation Process



Efficiency and Readability

- **Efficiency** is the ability to avoid wasting materials, energy, efforts, money, and time in doing something or in producing a desired result.
- **Readability** is the ease with which a human reader can understand the purpose, control flow, and operation of source code.

Time

Memory

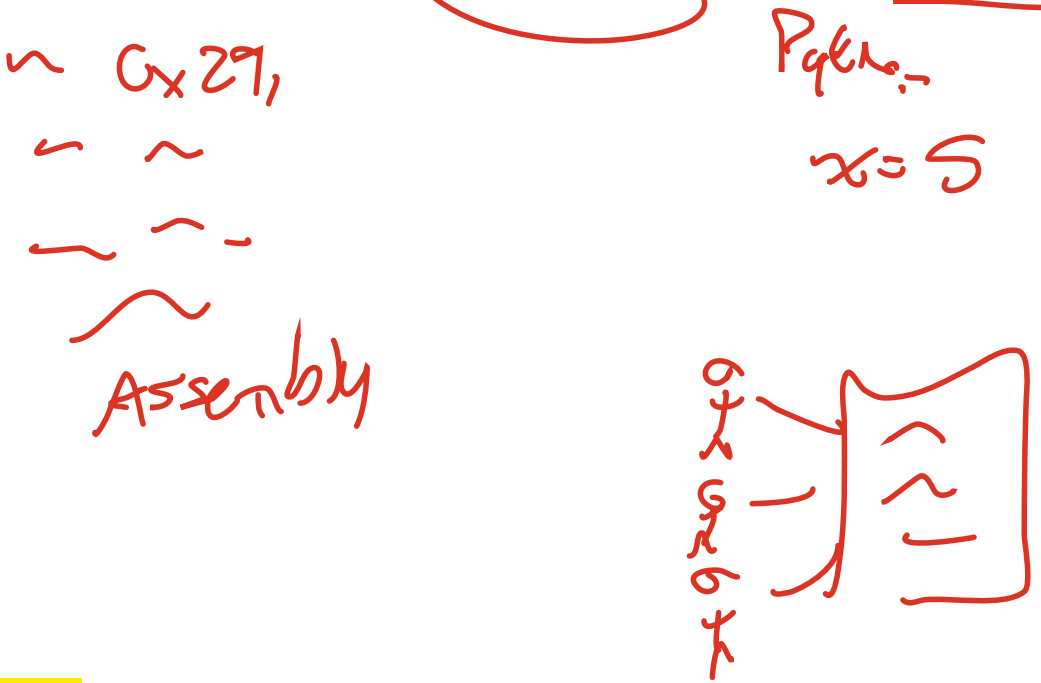
C++ / C | 0 ms

Python | 60 ms

Java | 120 ms

Efficiency and Readability

- **Efficiency** is the ability to avoid wasting materials, energy, efforts, money, and time in doing something or in producing a `desired result`.
- **Readability** is the ease with which a human reader can understand the purpose, control flow, and operation of `source code`.



Programming Languages Levels

32-bit (4-byte) `ADD` instruction:

100000	00101	00010	00011	00000000000
opcode	**rc**	**ra**	**rb**	**(unused)**

Could be something like: `Reg[4] <- Reg[2] + Reg[3]`

In `assembler`:

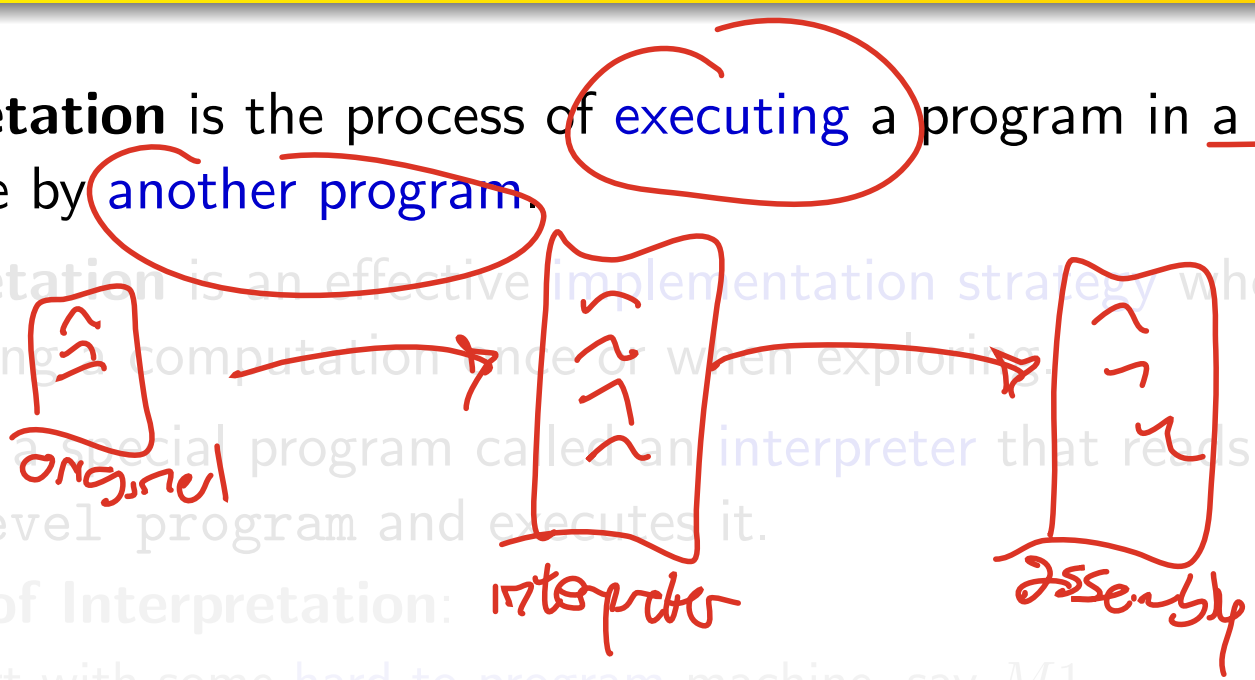
```
1    ADD(R2, R3, R4)
2
```

In any high-level language like `C`:

```
1    a = b + c;
2
```

Interpretation

- **Interpretation** is the process of executing a program in a high-level language by another program.
- **Interpretation** is an effective implementation strategy when performing a computation once or when exploring.
- There is a special program called an interpreter that reads a `high-level program` and executes it.
- **Model of Interpretation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P1$ for $M1$ that mimics the operation of another easier machine $M2$.
  - Result, $P1$ is an interpreter for $M2$, it means, a virtual $M2$.
- **Advantages**:
  - Portability.
  - Flexibility.
  - Ease of debugging.



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Interpretation

- **Interpretation** is the process of executing a program in a high-level language by another program.
- **Interpretation** is an effective implementation strategy when performing a computation once or when exploring.
- There is a special program called an interpreter that reads a high-level program and executes it.
- **Model of Interpretation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P1$ for $M1$ that mimics the operation of another easier machine $M2$.
  - Result, $P1$ is an interpreter for $M2$, it means, a virtual $M2$.
- **Advantages**:
  - Portability.
  - Flexibility.
  - Ease of debugging.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Interpretation

- **Interpretation** is the process of executing a program in a high-level language by another program.
- **Interpretation** is an effective implementation strategy when performing a computation once or when exploring.
- There is a special program called an interpreter that reads a `high-level program` and executes it.
- **Model of Interpretation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P1$ for $M1$ that mimics the operation of another easier machine $M2$.
  - Result, $P1$ is an interpreter for $M2$, it means, a virtual $M2$.
- **Advantages**:
  - Portability.
  - Flexibility.
  - Ease of debugging.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Interpretation

- **Interpretation** is the process of executing a program in a high-level language by another program.
- **Interpretation** is an effective implementation strategy when performing a computation once or when exploring.
- There is a special program called an interpreter that reads a `high-level program` and executes it.
- **Model of Interpretation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P1$ for $M1$ that mimics the operation of another easier machine $M2$.
  - Result: $P1$ is an interpreter for $M2$, it means, a virtual $M2$.
- **Advantages**:
  - Portability.
  - Flexibility.
  - Ease of debugging.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Interpretation

- **Interpretation** is the process of executing a program in a high-level language by another program.
- **Interpretation** is an effective implementation strategy when performing a computation once or when exploring.
- There is a special program called an interpreter that reads a `high-level program` and executes it.
- **Model of Interpretation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P1$ for $M1$ that mimics the operation of another easier machine $M2$.
  - Result, $P1$ is an interpreter for $M2$, it means, a virtual $M2$.
- **Advantages**:
  - Portability.
  - Flexibility.
  - Ease of debugging.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

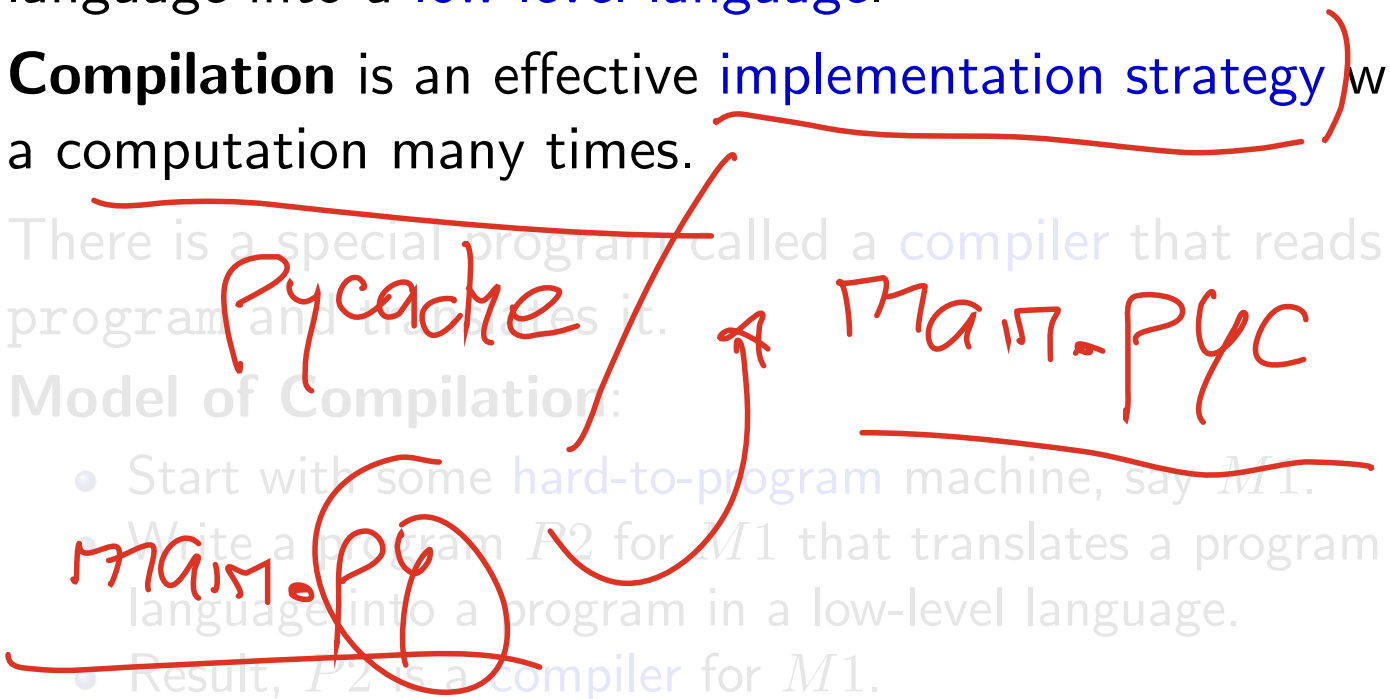
Compilation

- **Compilation** is the process of translating a program in a high-level language into a low-level language.
- **Compilation** is an effective implementation strategy when performing a computation many times.
- There is a special program called a compiler that reads a `high-level program` and translates it.
- **Model of Compilation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P2$ for $M1$ that translates a program in a high-level language into a program in a low-level language.
  - Result, $P2$ is a compiler for $M1$.
- **Advantages**:
  - Fast Execution.
  - Efficiency.
  - Portability.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Compilation

- **Compilation** is the process of translating a program in a high-level language into a low-level language.
- **Compilation** is an effective implementation strategy when performing a computation many times.
- There is a special program called a compiler that reads a `high-level program` and translates it.
- **Model of Compilation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P2$ for $M1$ that translates a program in a high-level language into a program in a low-level language.
  - Result, $P2$ is a compiler for $M1$.
- **Advantages**:
  - Fast Execution.
  - Efficiency.
  - Portability.



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Compilation

- **Compilation** is the process of translating a program in a high-level language into a low-level language.
- **Compilation** is an effective implementation strategy when performing a computation many times.
- There is a special program called a compiler that reads a `high-level program` and translates it.
- **Model of Compilation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P2$ for $M1$ that translates a program in a high-level language into a program in a low-level language.
  - Result, $P2$ is a compiler for $M1$.
- **Advantages**:
  - Fast Execution.
  - Efficiency.
  - Portability.

Compilation

- **Compilation** is the process of translating a program in a high-level language into a low-level language.
- **Compilation** is an effective implementation strategy when performing a computation many times.
- There is a special program called a compiler that reads a `high-level program` and translates it.
- **Model of Compilation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P2$ for $M1$ that translates a program in a high-level language into a program in a low-level language.
  - Result, $P2$ is a compiler for $M1$.
- **Advantages**:
  - Fast Execution.
  - Efficiency.
  - Portability.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Compilation

- **Compilation** is the process of translating a program in a high-level language into a low-level language.
- **Compilation** is an effective implementation strategy when performing a computation many times.
- There is a special program called a compiler that reads a `high-level program` and translates it.
- **Model of Compilation**:
  - Start with some hard-to-program machine, say $M1$.
  - Write a program $P2$ for $M1$ that translates a program in a high-level language into a program in a low-level language.
  - Result, $P2$ is a compiler for $M1$.
- **Advantages**:
  - Fast Execution.
  - Efficiency.
  - Portability.

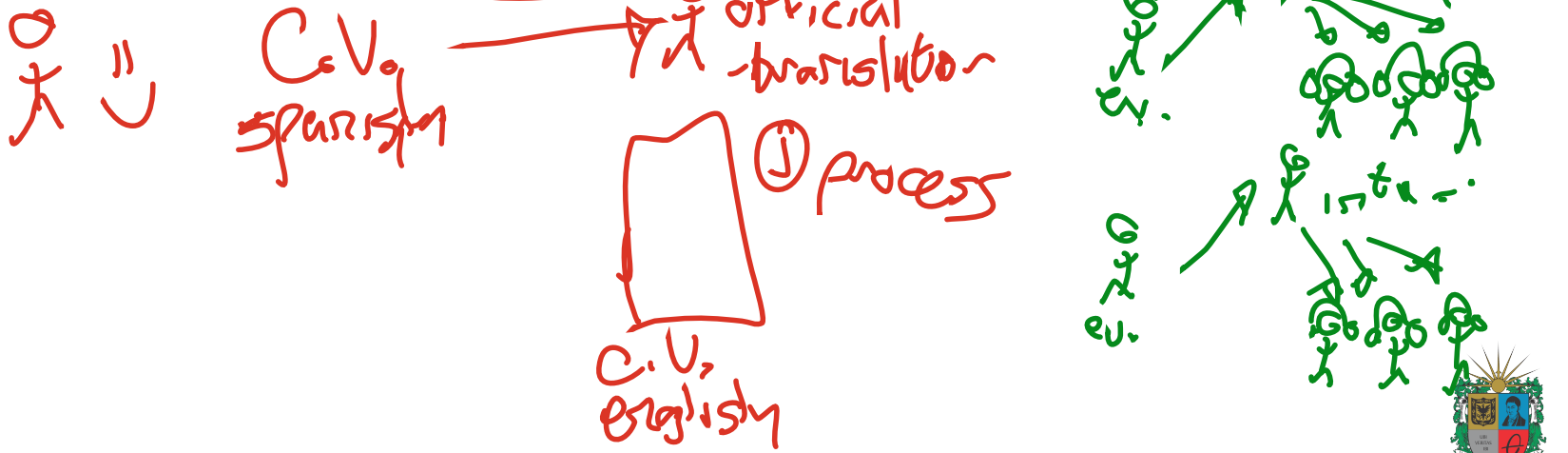
Pointers

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Interpretation Vs. Compilation

Characteristics differences:

	Compilation	Interpretation
How does it treat input `x + 2`?	Generate a program that computes `x + 2`	Computes `x + 2`
When it happens?	Before Execution	During Execution
What it complicates/slows?	Program Development	Program Execution
Decisions made at	Compile Time	Run Time



Outline

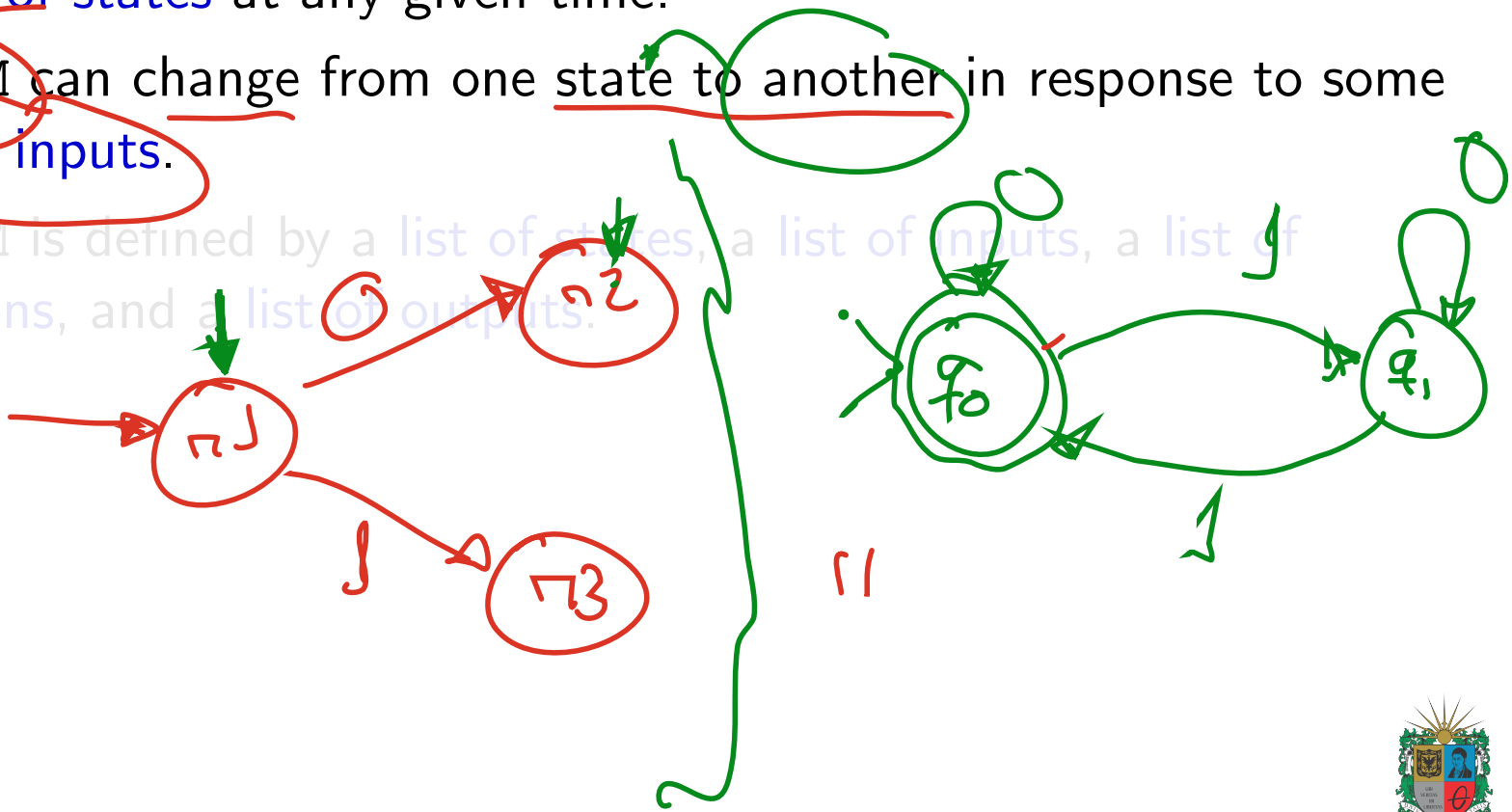
UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Finite-State Machines

- A **finite-state machine** (FSM) is a mathematical model of computation.
- It is an abstract machine that can be in exactly one of a finite number of states at any given time.
- The FSM can change from one state to another in response to some external inputs.
- The FSM is defined by a list of states, a list of inputs, a list of transitions, and a list of outputs.

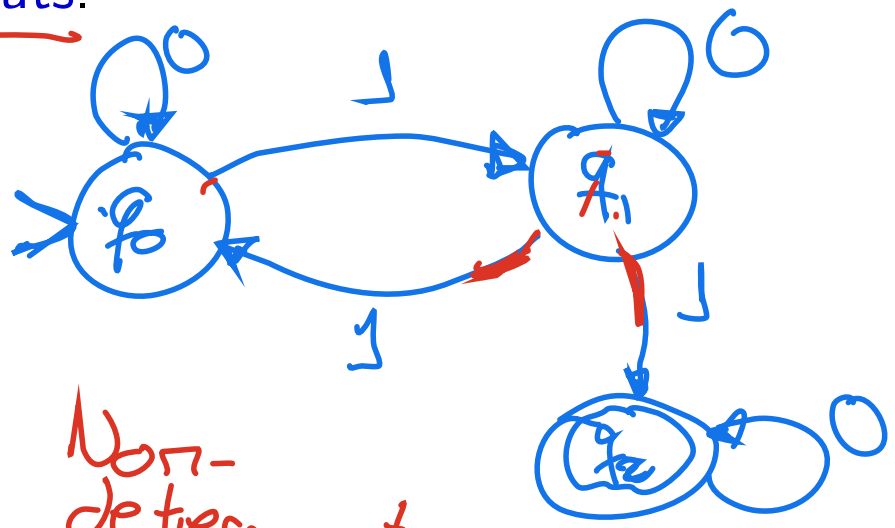
Finite-State Machines

- A **finite-state machine** (FSM) is a mathematical model of computation.
- It is an abstract machine that can be in exactly one of a finite number of states at any given time.
- The FSM can change from one state to another in response to some external inputs.
- The FSM is defined by a list of states, a list of inputs, a list of transitions, and a list of outputs.

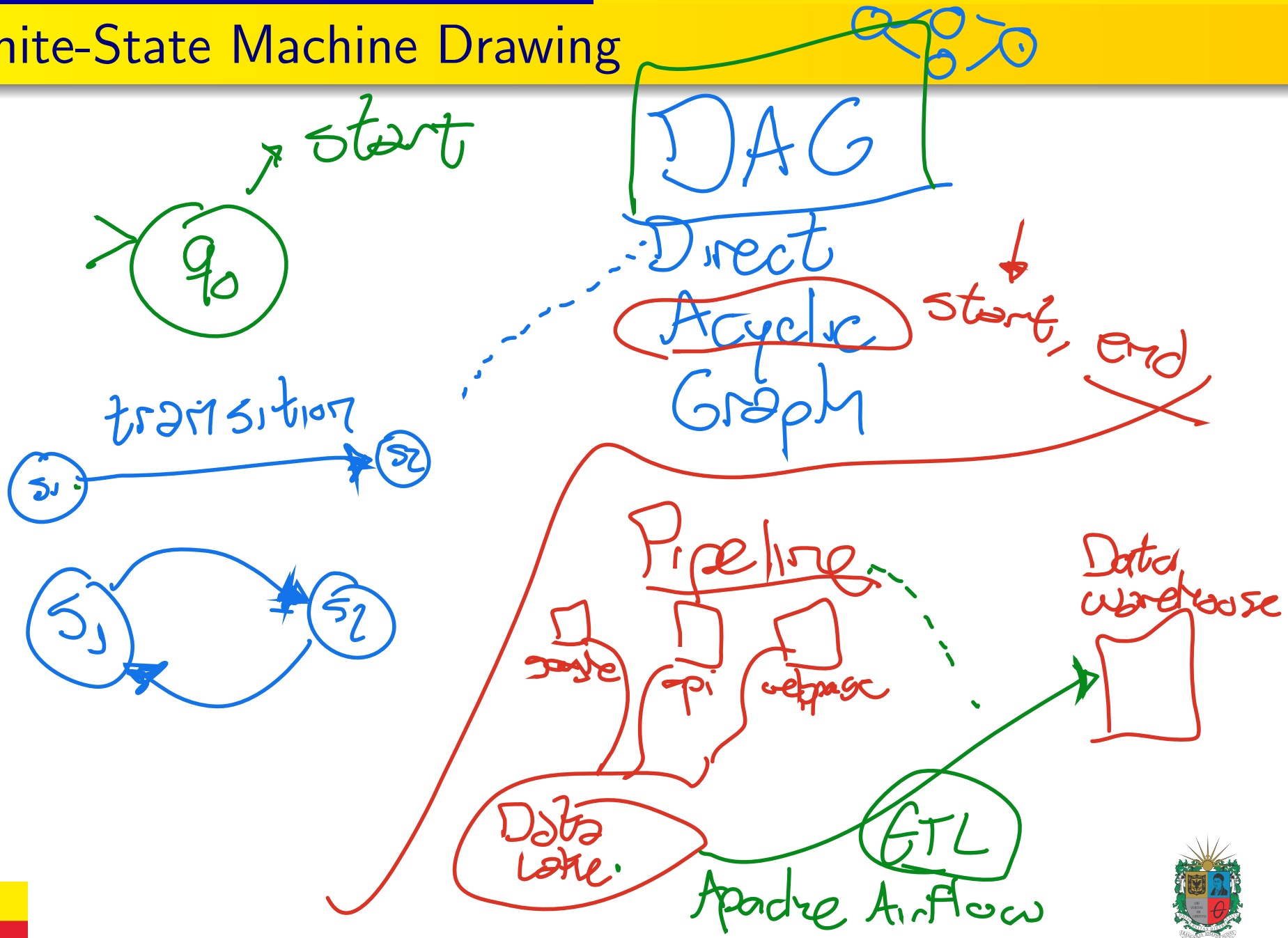


Finite-State Machines

- A **finite-state machine** (FSM) is a mathematical model of computation.
- It is an abstract machine that can be in exactly one of a finite number of states at any given time.
- The FSM can change from one state to another in response to some external inputs.
- The FSM is defined by a list of states, a list of inputs, a list of transitions, and a list of outputs.



Finite-State Machine Drawing



Alonzo Church

- **Alonzo Church** (1903 — 1995) was an American mathematician and logician.
- He is best known for the **Lambda Calculus**, which he developed in the 1930s.
- The **Lambda Calculus** is a formal system in mathematical logic for expressing computation based on function abstraction and application.

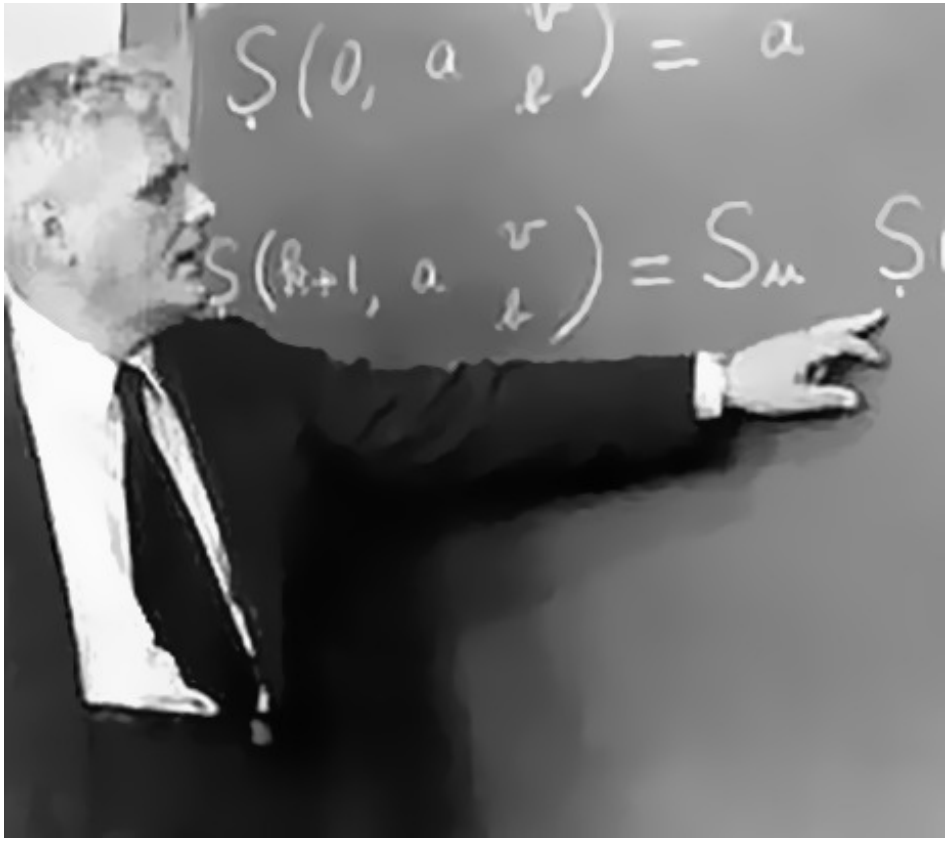
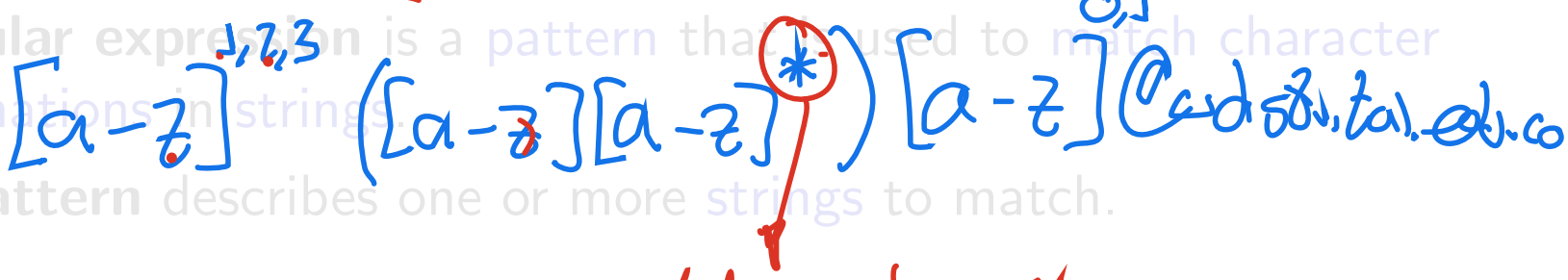


Figure: Alonzo Church

Regular Expressions

A **regular expression** is a sequence of characters that define a search pattern.

- **Regular expressions** are used in search engines, search and replace dialogs of word processors, and in text processing utilities.
- A **regular expression** is a pattern that is used to match character combinations in strings.
- The **pattern** describes one or more strings to match.

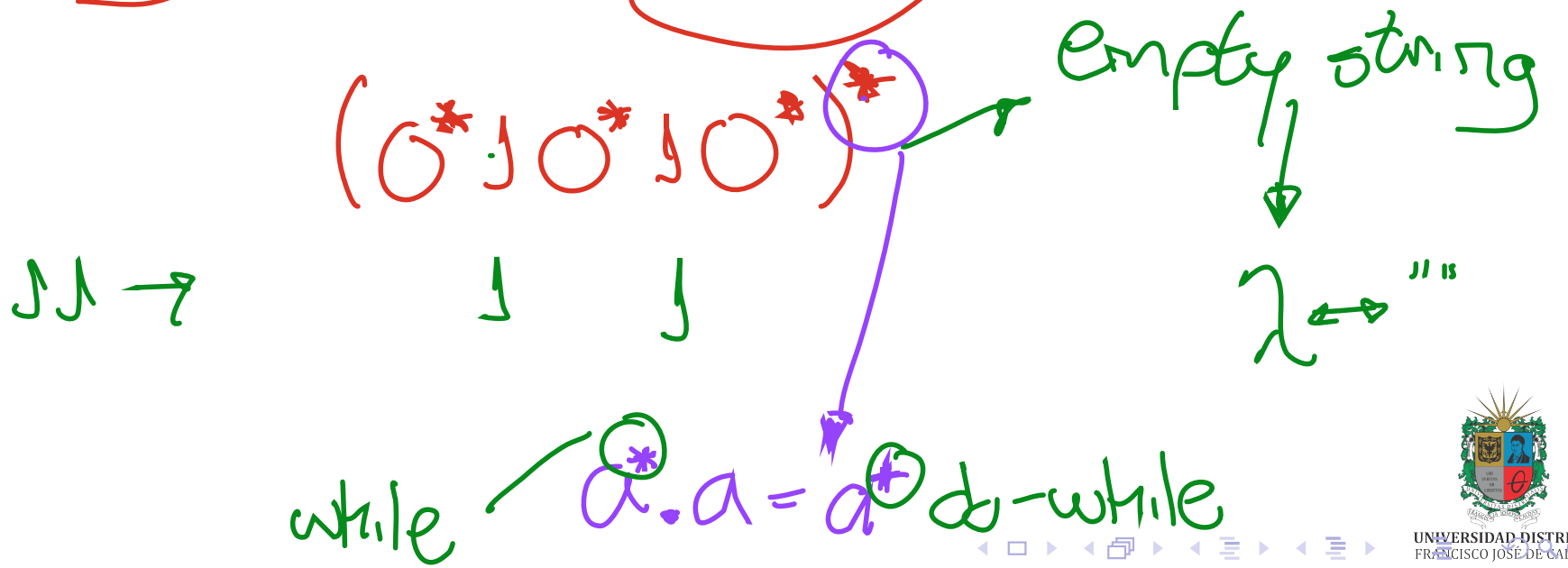


UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Regular Expressions

A **regular expression** is a sequence of characters that define a search pattern.

- **Regular expressions** are used in search engines, search and replace dialogs of word processors, and in text processing utilities.
- A **regular expression** is a pattern that is used to match character combinations in strings.
- The **pattern** describes one or more strings to match.



Finite Automata: Concatenation

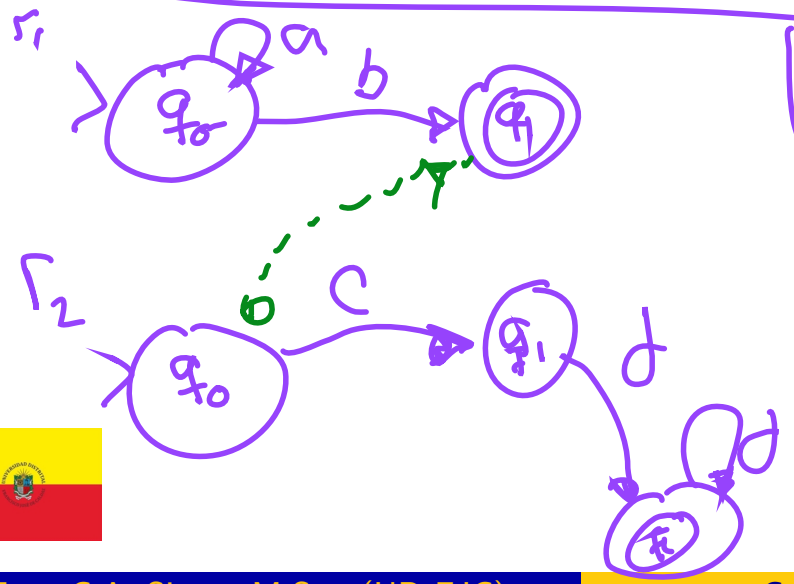
The **concatenation** of two regular expressions R_1 and R_2 is a regular expression that matches the concatenation of strings that are matched by R_1 followed by R_2.

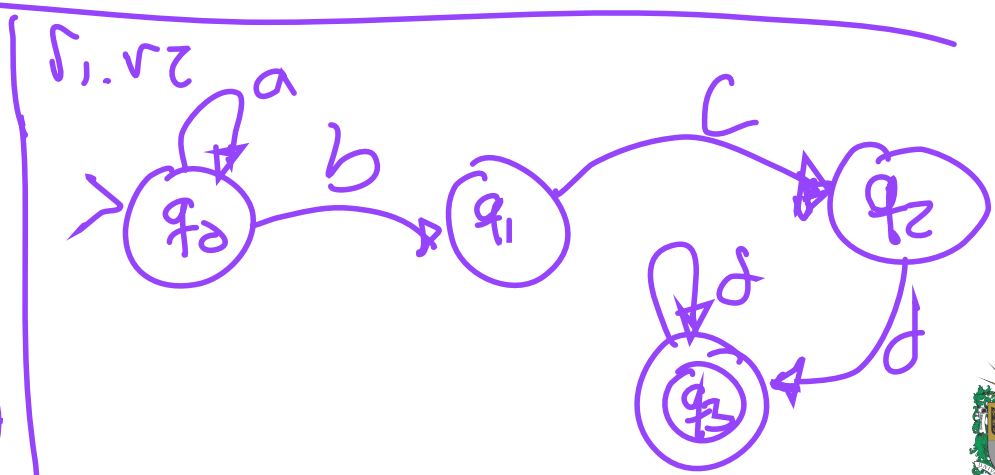
$r_1 = a^* b$

$r_2 = cd^+$

$r_1 \cdot r_2 = a^* b c d^+$

$r_1 r_2 = a^* b c d^+$





Finite Automata: Union

The **union** of two regular expressions R_1 and R_2 is a regular expression that matches the union of strings that are matched by R_1 **or** R_2.

$r_1 = a^*b$

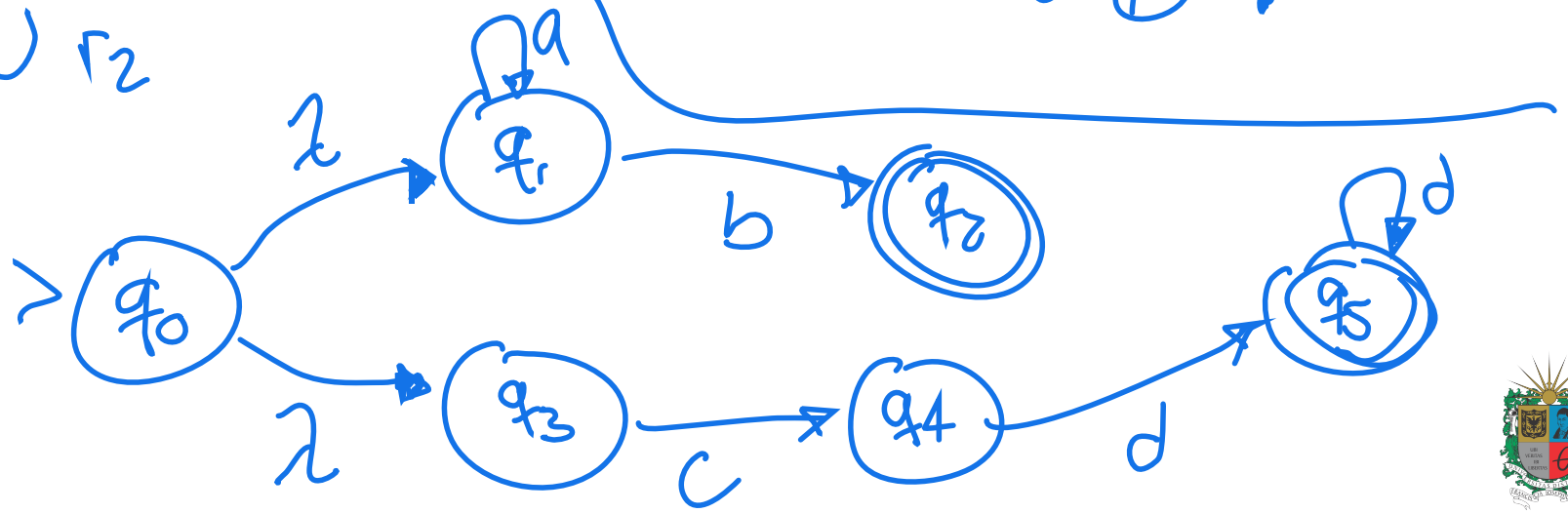
$r_2 = cd^+$

$r_1 \cup r_2 = a^*b \cup cd^+$

$= a^*b \text{ or } cd^+$

$= a^*b \mid cd^+$

$r_1 \cup r_2$



Finite Automata: Kleene's Star

The **Kleene's star** of a regular expression R is a regular expression that matches the concatenation of zero or more strings that are matched by R.

$a^* = a$

$a^2 = a.a$

$a^0 = \lambda$

$a^* = aaaa \ldots a$

$a^+ = a.a^*$

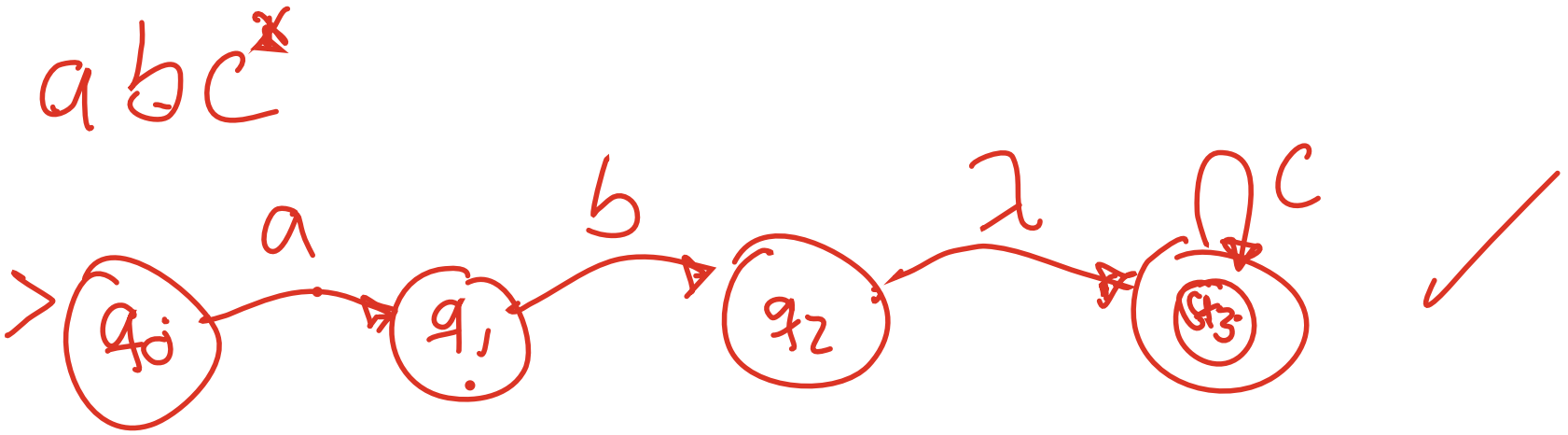
$a^+ = a^*.a$

$a.\lambda = a$

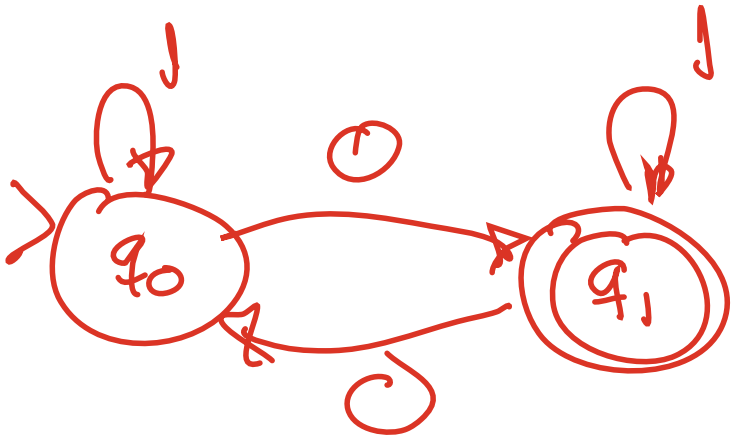
$abc.\lambda = abc$

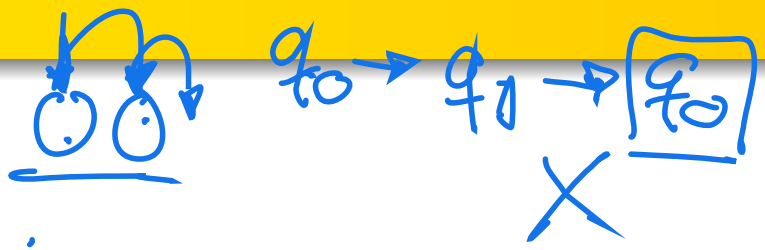
Finite Automata: λ-transition

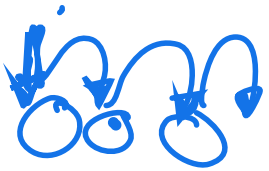
The λ-**transition** is a transition that can be taken without consuming any input.

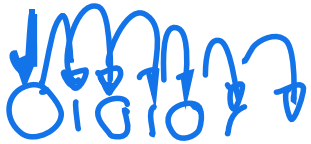


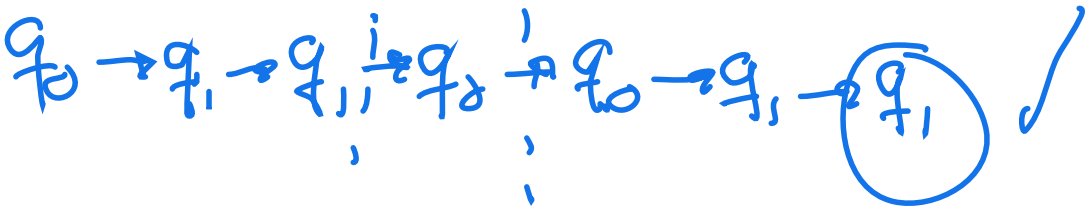
Strings Processing











Alan Turing

- **Alan Turing** (1912 — 1954) was an English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist.
- He is widely considered to be the **father of theoretical computer science** and **artificial intelligence**.
- He was highly influential in the development of theoretical computer science, providing a formalization of the concepts of algorithm and computation with the Turing machine.

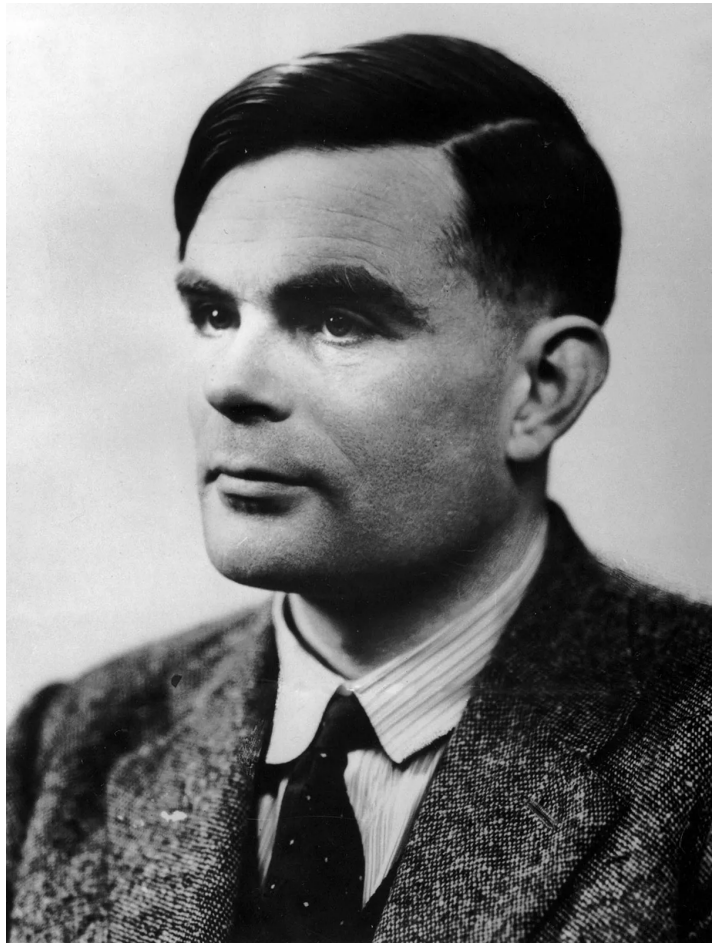
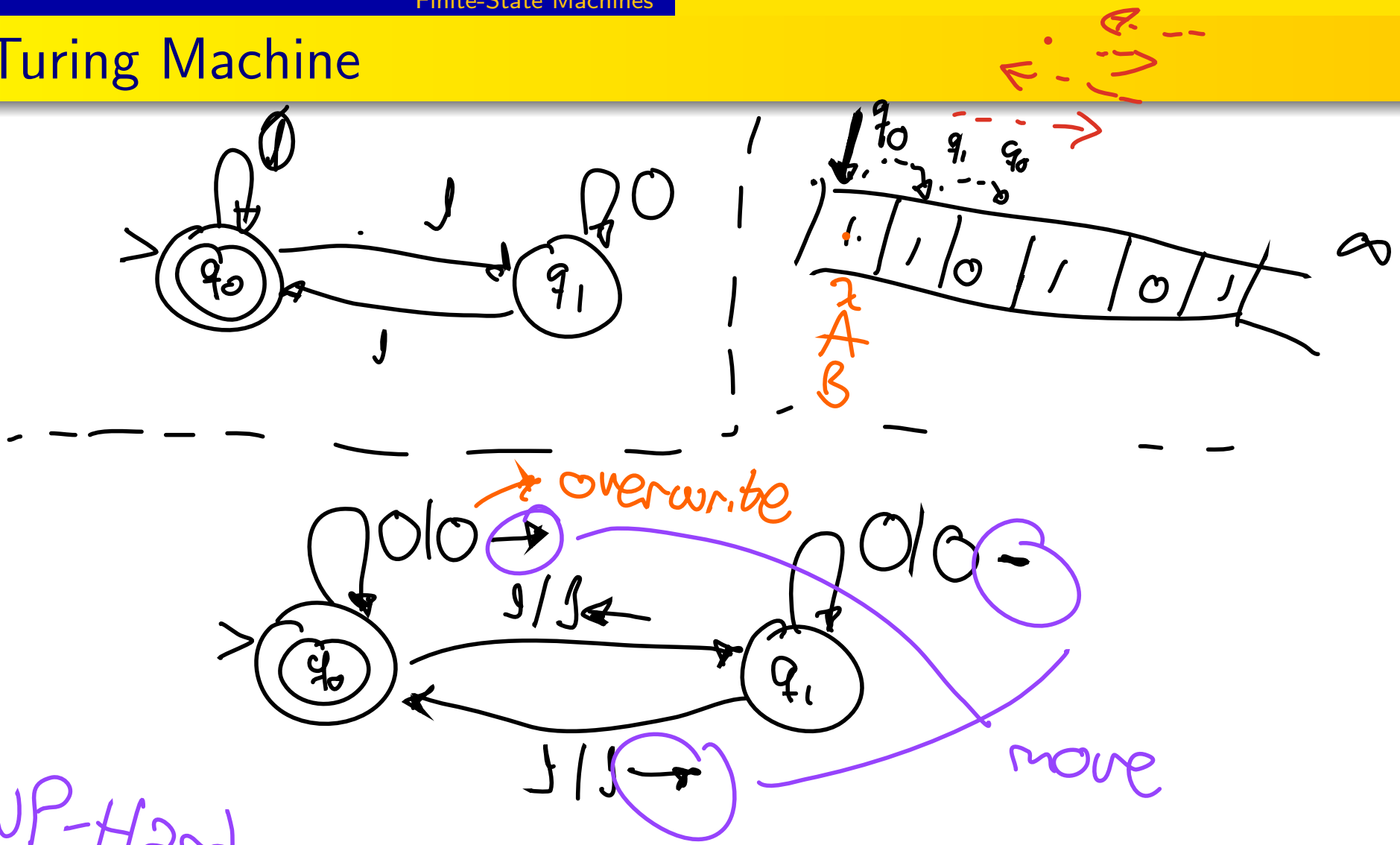


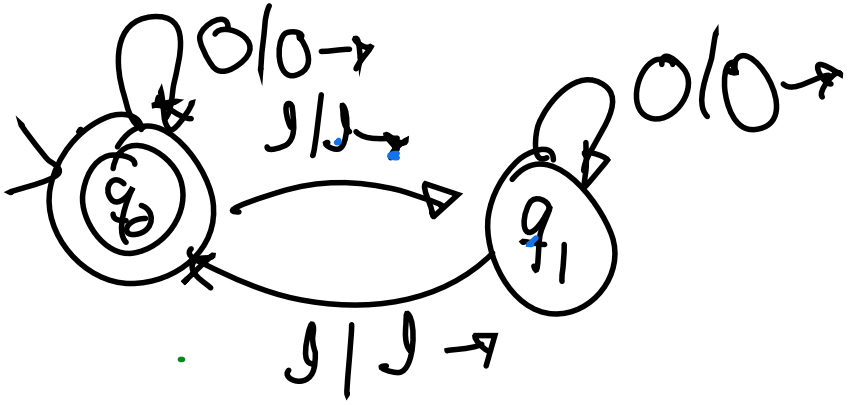
Figure: Alan Turing

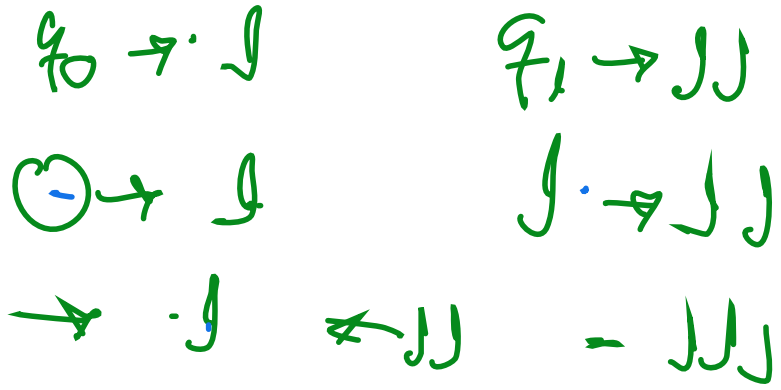
Turing Machine

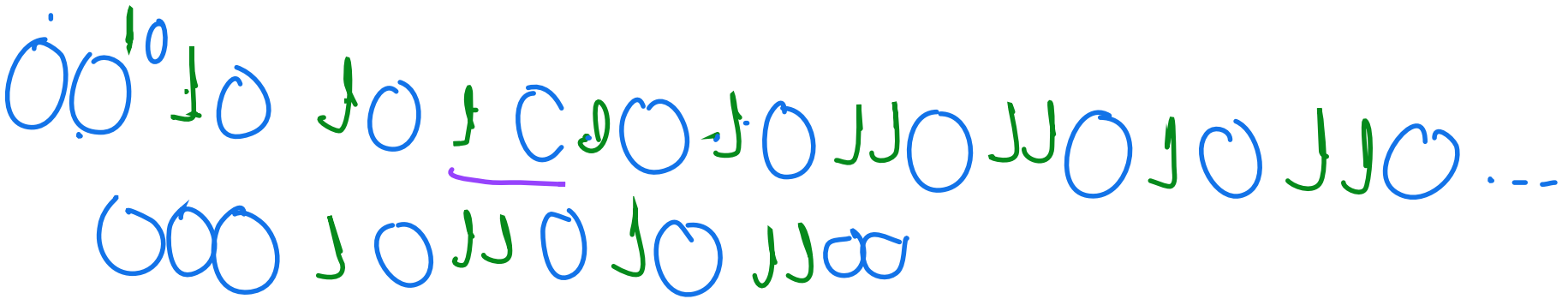


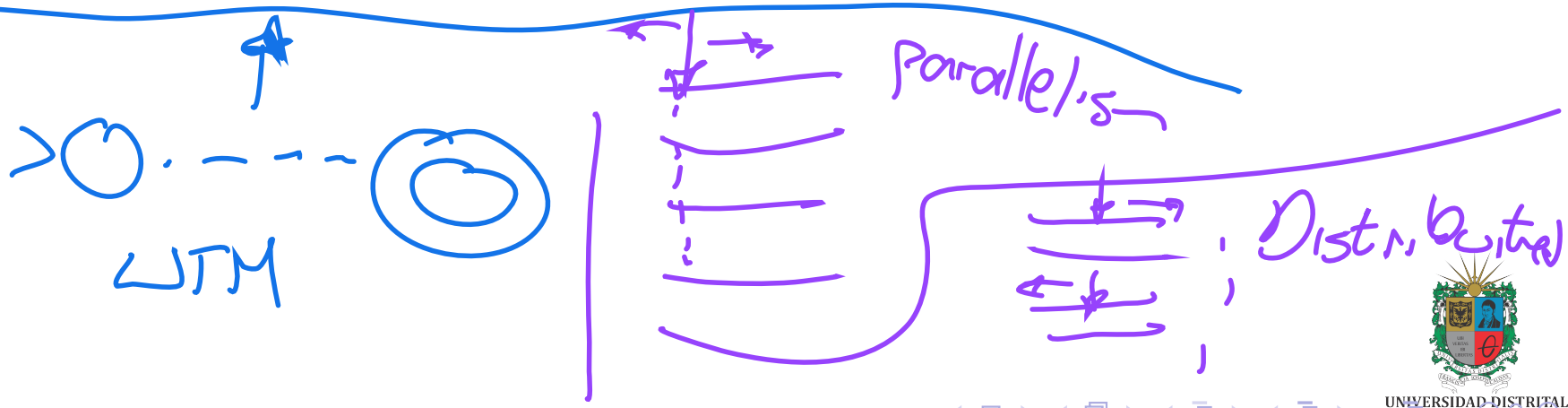
NP-Hard
NP-Complete

Universal Turing Machine



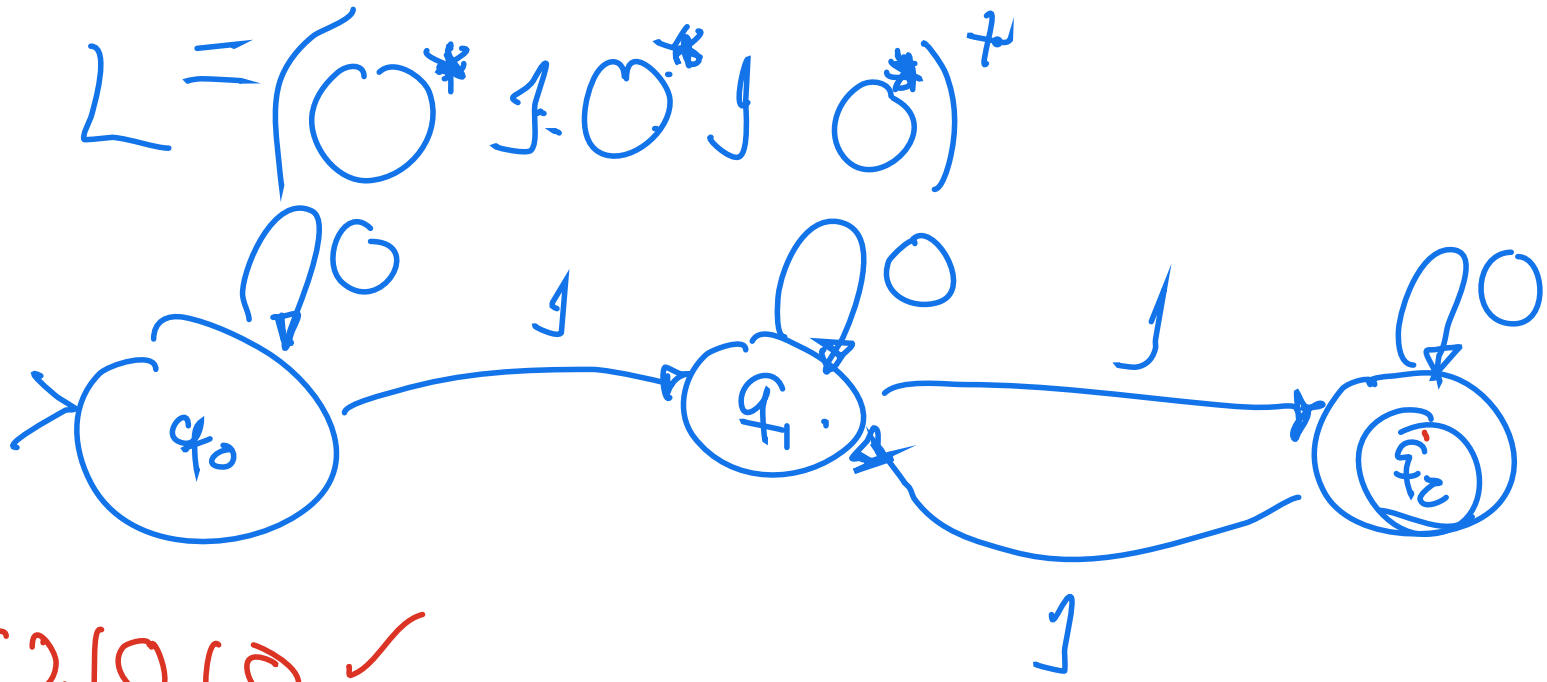


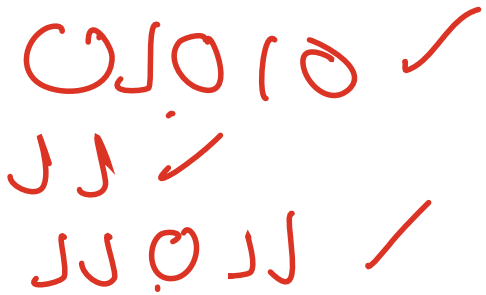




Regular Expression & Finite Automata: *Example i*

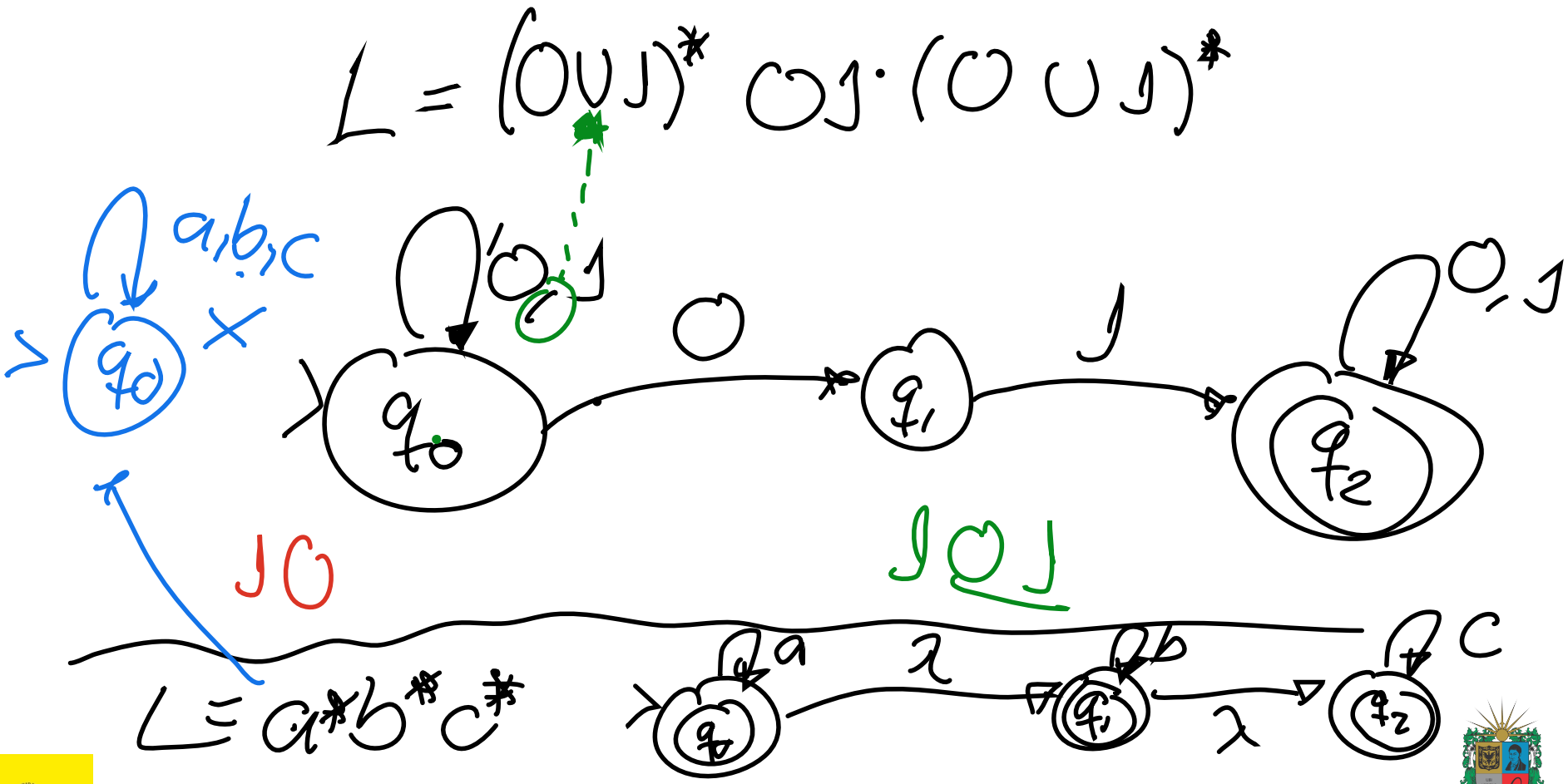
Be L a language over the alphabet $\Sigma = \{0, 1\}$, such that L is the set of all strings that contain an even number of 1s. The regular expression for L is:





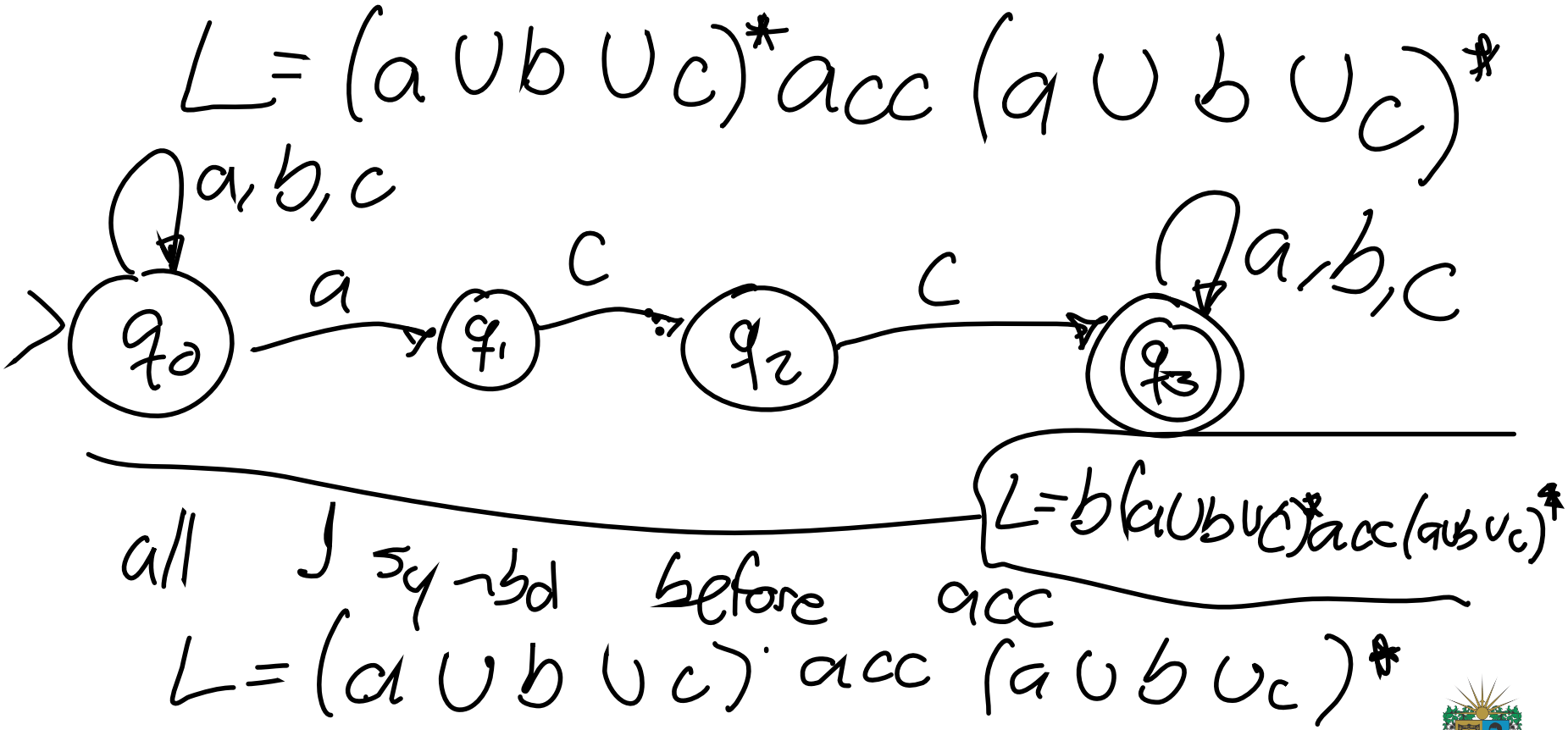
Regular Expression & Finite Automata: *Example ii*

Be L a language over the alphabet $\Sigma = \{0, 1\}$, such that L is the set of all strings that contain the substring 01. The regular expression for L is:



Regular Expression & Finite Automata: *Example iii*

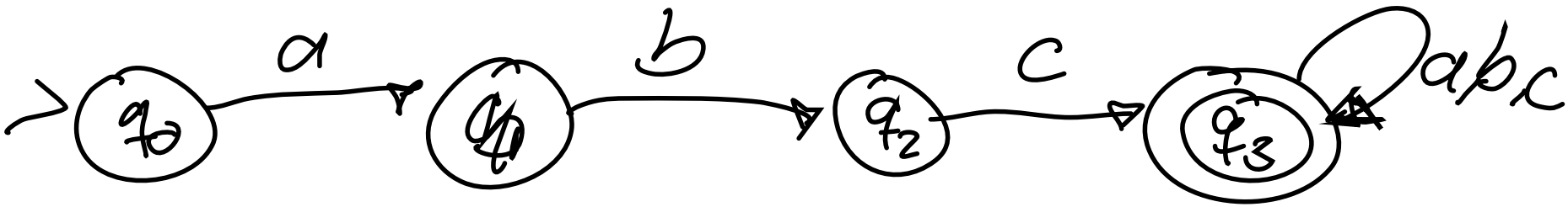
Be L a language over the alphabet $\Sigma = \{a, b, c\}$, such that L is the set of all strings that contain the substring acc. The regular expression for L is:



Regular Expression & Finite Automata: *Example iv*

Be L a language over the alphabet $\Sigma = \{a, b, c\}$, such that L is the set of all strings that start with the substring abc. The regular expression for L is:

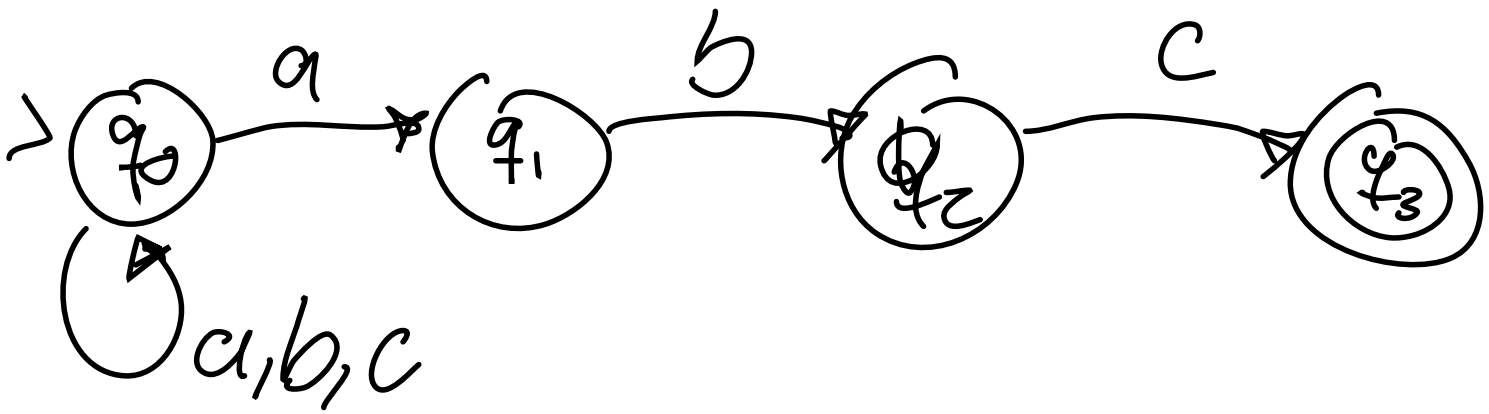
$$L = abc\,(a \cup b \cup c)^*$$



Regular Expression & Finite Automata: *Example v*

Be L a language over the alphabet $\Sigma = \{a, b, c\}$, such that L is the set of all strings that end with the substring abc. The regular expression for L is:

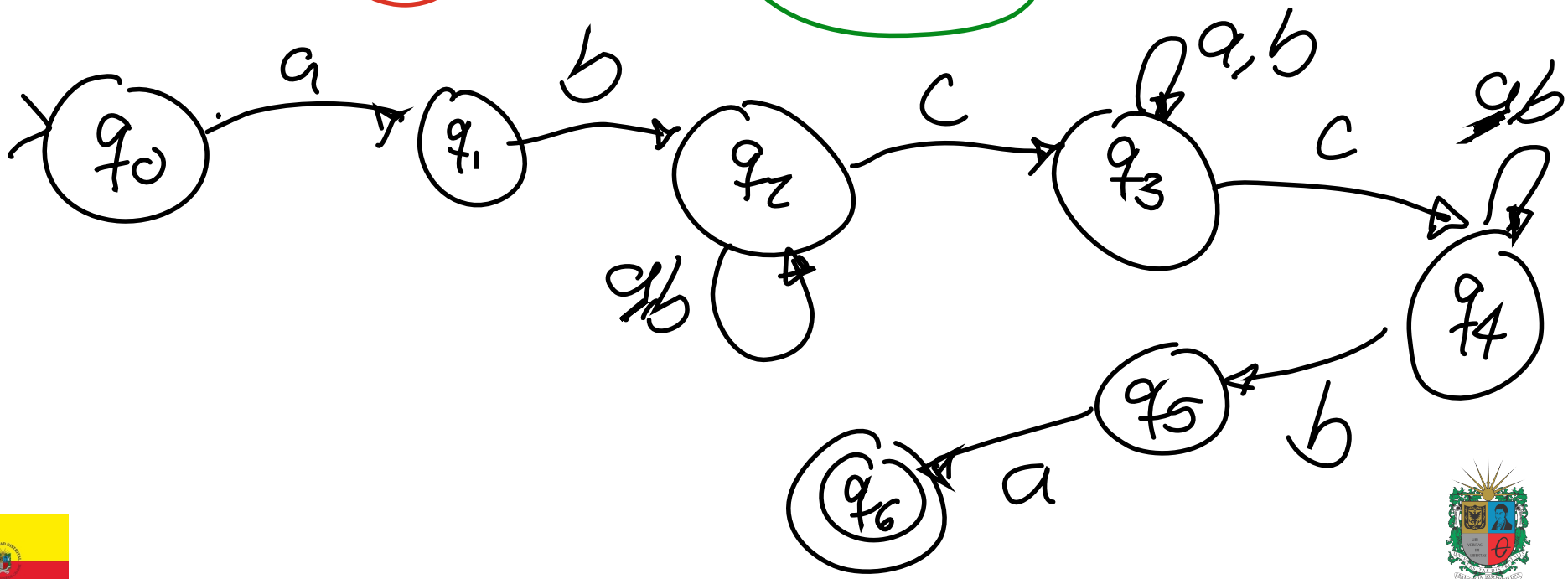
$$L = (a \cup b \cup c)^* abc$$



Regular Expression & Finite Automata: *Example vi*

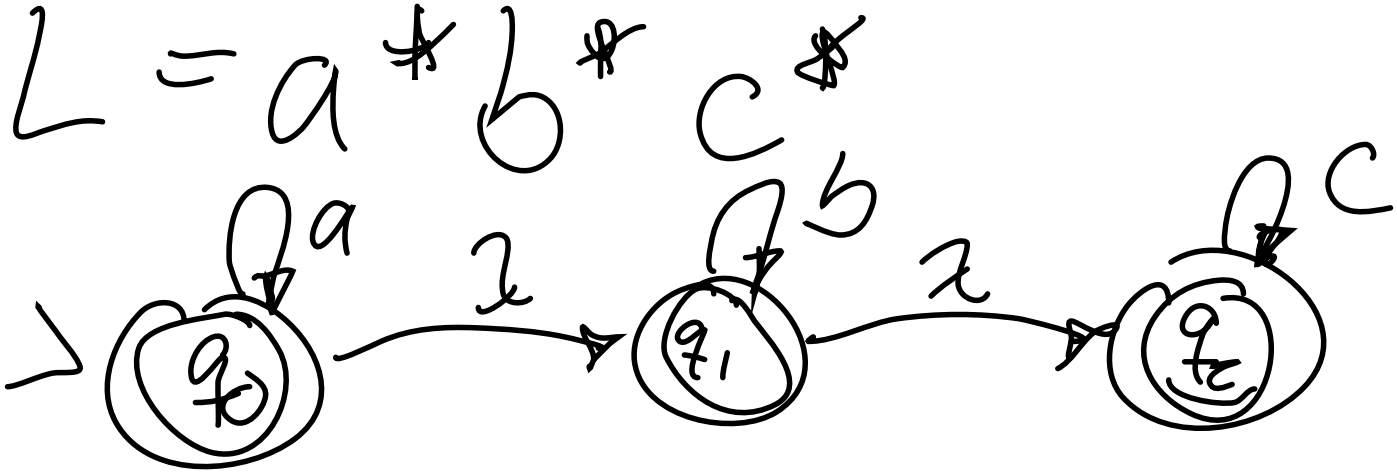
Be L a language over the alphabet $\Sigma = \{a, b, c\}$, such that L is the set of all strings that start with the substring ab, contain just two c's and end with the substring ba. The regular expression for L is:

$$L = ab(a \cup b)^* c (a \cup b)^* c (a \cup b)^* ba$$



Regular Expression & Finite Automata: *Example vii*

Be L a language over the alphabet $\Sigma = \{a, b, c\}$, such that L is the set of all strings that start with any number of a's (could be 0), followed by any number of b's, and end with any number of c's. The regular expression for L is:

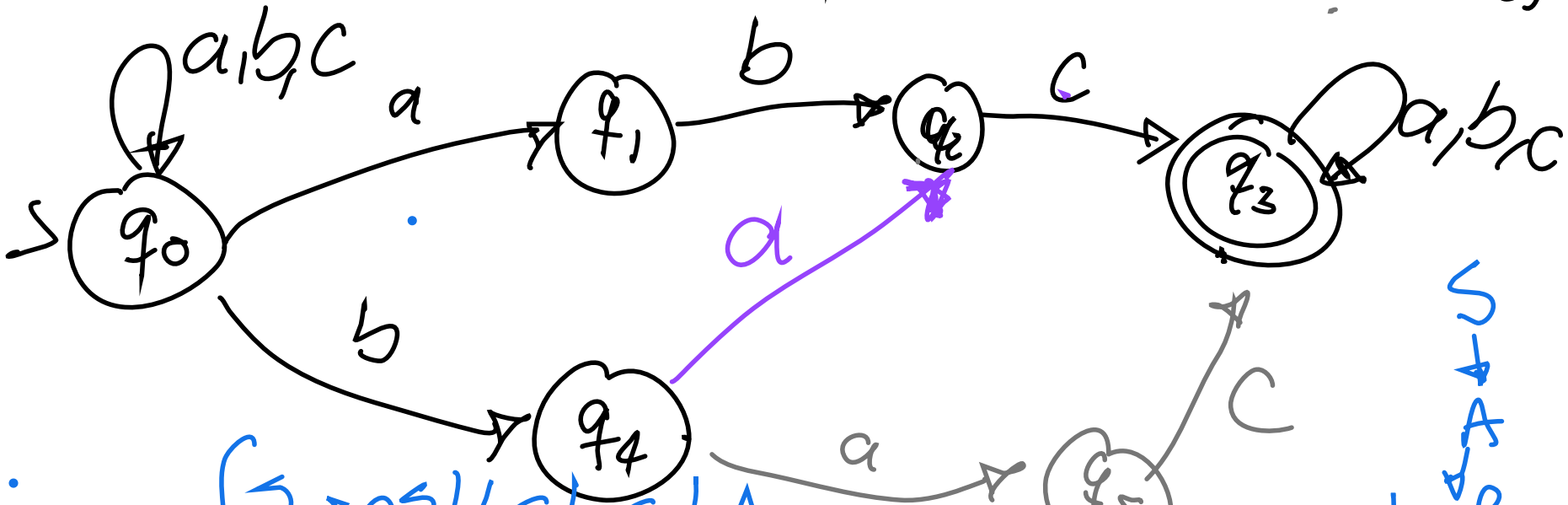


UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Regular Expression & Finite Automata: *Example viii*

Be L a language over the alphabet $\Sigma = \{a, b, c\}$, such that L is the set of all strings that contain the substring abc or bac. The regular expression for L is:

$$L = (a \cup b \cup c)^* (abc \cup bac)(a \cup b \cup c)^*$$



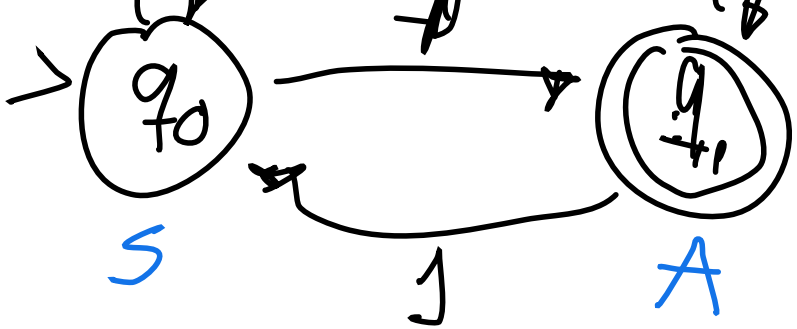
$$G = \begin{cases} S \to aS \mid bS \mid cS \mid A \\ A \to abcB \mid bacB \\ B \to aB \mid bB \mid cB \mid \lambda \end{cases}$$

S → A → abcB

Regular Expression & Finite Automata: *Example ix*

Be L a language over the alphabet $\Sigma = \{0, 1\}$, such that L is the set of all strings that contain an odd number of 1s. The regular expression for L is:

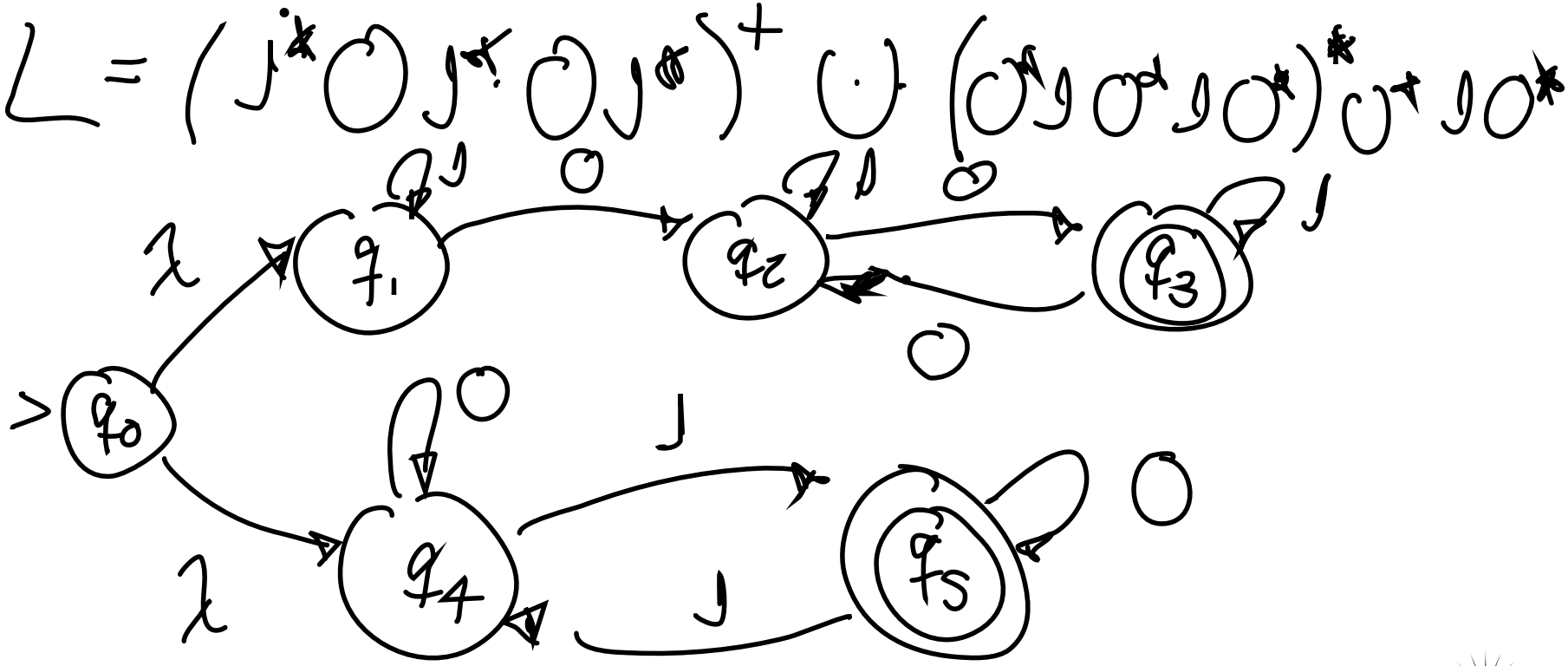
$$L = (0^*10^*10^*)^*0^*10^*$$



$$\Rightarrow G = \begin{cases} S \to 0S \mid 1A \\ A \to 1S \mid 0A \mid \lambda \end{cases}$$

Regular Expression & Finite Automata: *Example x*

Be L a language over the alphabet $\Sigma = \{0, 1\}$, such that L is the set of all strings that contain an even number of 0s or an odd number of 1s. The regular expression for L is:



Outline

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Noam Chomsky

- **Noam Chomsky** (1928 —) is an American linguist, philosopher, cognitive scientist, historian, social critic, and political activist.
- He is considered the **father of modern linguistics**.
- He introduced the **Chomsky hierarchy**, a classification of formal languages.

Figure: Noam Chomsky

Natural Processing Language

- **Natural Language Processing** (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human languages.
- **NLP** is used to apply algorithms to text and speech.
- **NLP** is used to understand the meaning of text and speech.
- **NLP** is used to generate human language text.

vibe code

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Natural Processing Language

- **Natural Language Processing** (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human languages.
- **NLP** is used to apply algorithms to text and speech.
- **NLP** is used to understand the meaning of text and speech.
- **NLP** is used to generate human language text.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Natural Processing Language

- **Natural Language Processing** (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human languages.
- **NLP** is used to apply algorithms to text and speech.
- **NLP** is used to understand the meaning of text and speech.
- **NLP** is used to generate human language text.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Formal Languages

- A **formal language** is a set of strings of symbols.
- The **alphabet** of a formal language is the set of symbols that can be used to form strings.
- A **formal grammar** is a set of rules for generating strings in a formal language.
- A **generative grammar** is a formal system for describing the structure of strings in a formal language.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Formal Languages

- A **formal language** is a set of strings of symbols.
- The **alphabet** of a formal language is the set of symbols that can be used to form strings.
- A **formal grammar** is a set of rules for generating strings in a formal language.
- A **generative grammar** is a formal system for describing the structure of strings in a formal language.

Grammars Foundations

- A **grammar** is a set of rules for generating strings in a formal language.
- A **grammar** is a formal system for describing the structure of strings in a formal language.
- A **generative grammar** is a set of rules for generating strings in a formal language.

Grammars Foundations

- A **grammar** is a set of rules for generating strings in a formal language.
- A **grammar** is a formal system for describing the structure of strings in a formal language.
- A **generative grammar** is a set of rules for generating strings in a formal language.

Chomsky Hierarchy

- The **Chomsky hierarchy** is a classification of formal languages.
- The **Chomsky hierarchy** is named after the linguist and cognitive scientist **Noam Chomsky**.
- The **Chomsky hierarchy** consists of four types of formal grammars:
  - **Type 0**: Unrestricted grammars.
  - **Type 1**: Context-sensitive grammars.
  - **Type 2**: Context-free grammars.
  - **Type 3**: Regular grammars.

Chomsky Hierarchy

- The **Chomsky hierarchy** is a classification of formal languages.
- The **Chomsky hierarchy** is named after the linguist and cognitive scientist **Noam Chomsky**.
- The **Chomsky hierarchy** consists of four types of formal grammars:
  - **Type 0**: Unrestricted grammars.
  - **Type 1**: Context-sensitive grammars.
  - **Type 2**: Context-free grammars.
  - **Type 3**: Regular grammars.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Context-Free Grammars

- A **context-free grammar** is a formal grammar and generative grammar in which every production rule is of the form:

$$A \rightarrow \alpha \cdot \quad (1)$$

replacement

variable

- Where A is a nonterminal symbol and α is a string of terminals and nonterminals.
- A **context-free grammar** is a generative grammar that can generate a context-free language.

variable

replacement

pronoun → I, you, he, she, it, they, we

verb → sleep, eat, run, jump, ...

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Context-Free Grammars

- A **context-free grammar** is a formal grammar and generative grammar in which every production rule is of the form:

$$A \to \alpha \tag{1}$$

- Where A is a nonterminal symbol and α is a string of terminals and nonterminals.
- A **context-free grammar** is a generative grammar that can generate a context-free language.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Context-Free Grammars

- A **context-free grammar** is a formal grammar and generative grammar in which every production rule is of the form:

$$A \to \alpha \tag{1}$$

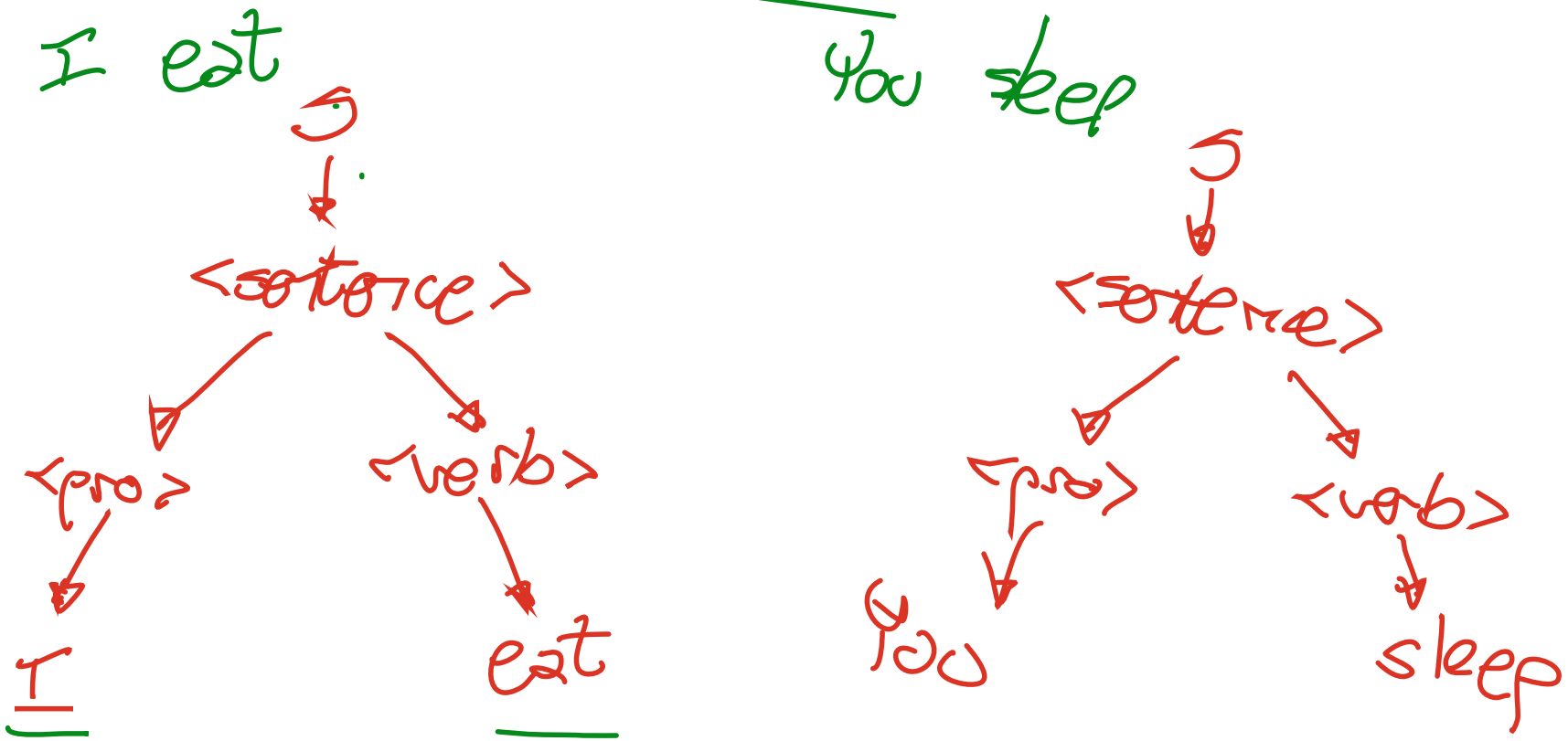
- Where A is a nonterminal symbol and α is a string of terminals and nonterminals.
- A **context-free grammar** is a generative grammar that can generate a context-free language.

start

$$G = \begin{cases} S \to \langle \text{sentence} \rangle \\ \langle \text{sentence} \rangle \to \langle \text{pro} \rangle \langle \text{verb} \rangle \\ \langle \text{pro} \rangle \to I \mid you \mid they \\ \langle \text{verb} \rangle \to eat \mid sleep \end{cases}$$

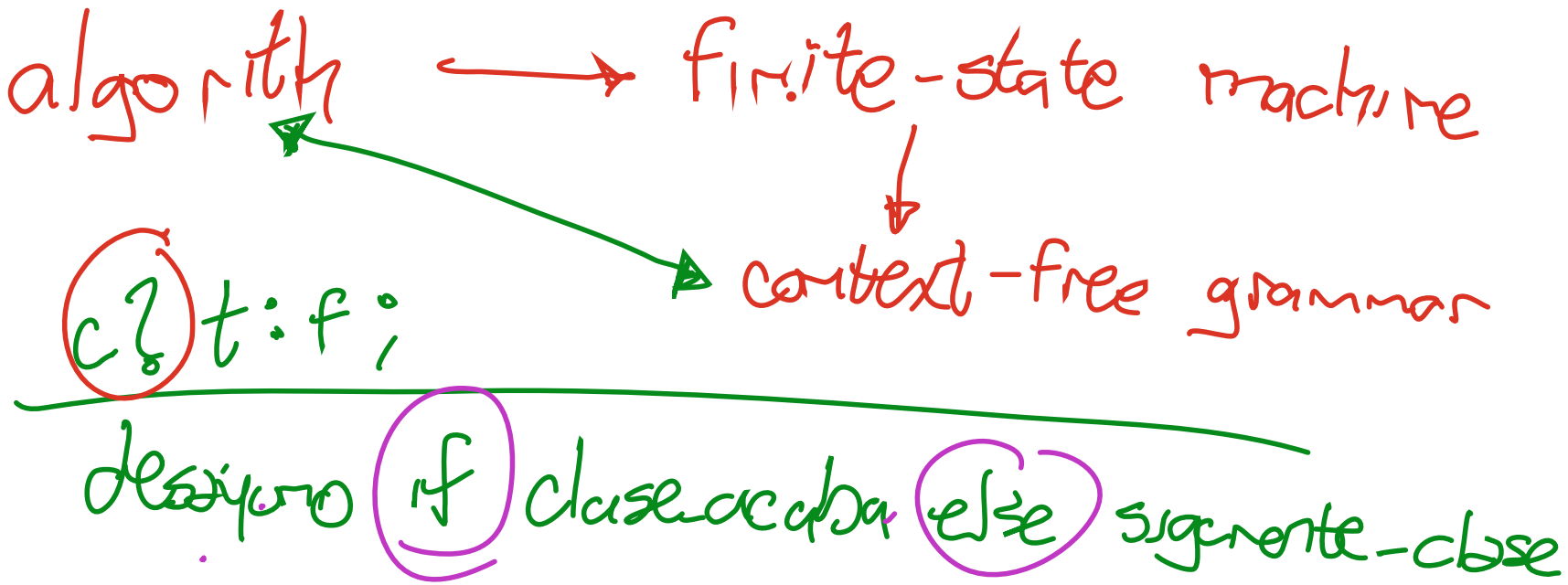
Derivation Trees

A **derivation tree** is a tree that represents the sequence of production rules used to generate a string in a formal language.



Equivalence between Grammars and Finite Automatas

A **context-free grammar** can be converted into a **finite automaton** and a **finite automaton** can be converted into a **context-free grammar**.



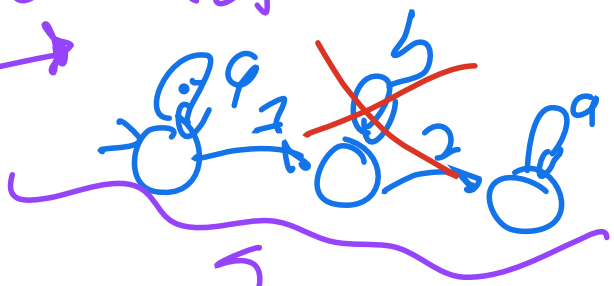
Equivalence Between Grammars and Regular Expressions

A **regular expression** can be converted into a **finite automaton** and a **finite automaton** can be converted into a **regular expression**.

regex → machine → grammar

$L = a^i b^j a^i,\ \Sigma = \{a, b\},\ i, j \geq 0$

$$G = \begin{cases} S \to aSa \mid B \\ B \to bB \mid \lambda \end{cases}$$



S

aSa

aaSaa

aaBaa

aabBaa

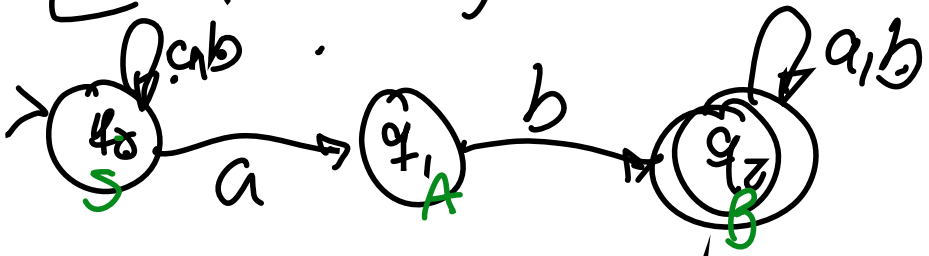
aabλaa = aabaa

aaba a

Free-Context Grammar: *Example i*

Be L a language over the alphabet $\Sigma = \{a, b\}$, such that L is the set of all strings that contain the substring ab. The context-free grammar for L is:

$$L = (a \cup b)^* ab (a \cup b)^*$$



$$G = \begin{cases} S \to aS \mid bS \mid aA \\ A \to bB \\ B \to aB \mid bB \mid \lambda \end{cases}$$

Free-Context Grammar: *Example ii*

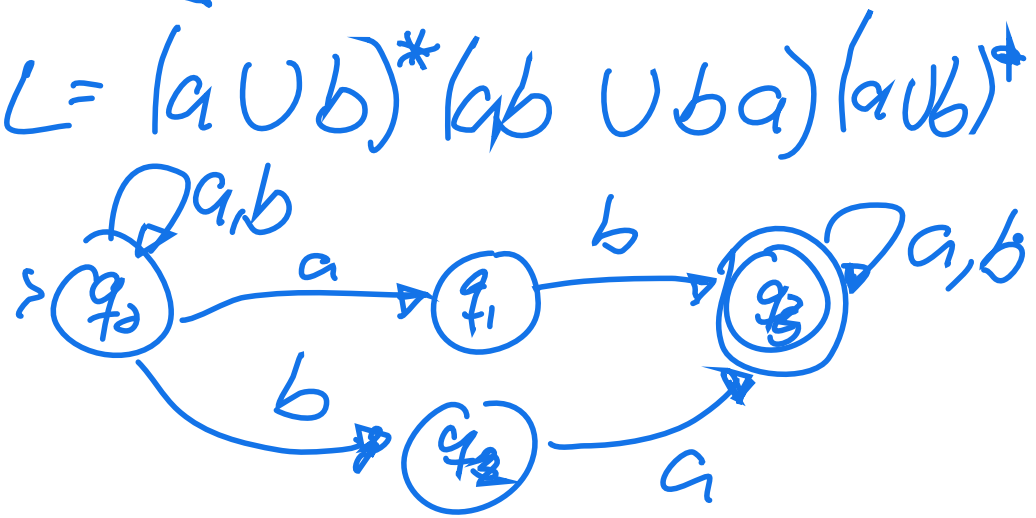
Be L a language over the alphabet $\Sigma = \{a, b\}$, such that L is the set of all strings that contain the substring ab or ba. The context-free grammar for L is:

$$G = \begin{cases} S \to aS \mid bS \mid A \\ A \to abB \mid baB \\ B \to aB \mid bB \mid \lambda \end{cases}$$

$S \Rightarrow aA \Rightarrow abC \Rightarrow ab\lambda = ab$

equivalent

$L = (a \cup b)^*(ab \cup ba)(a \cup b)^*$

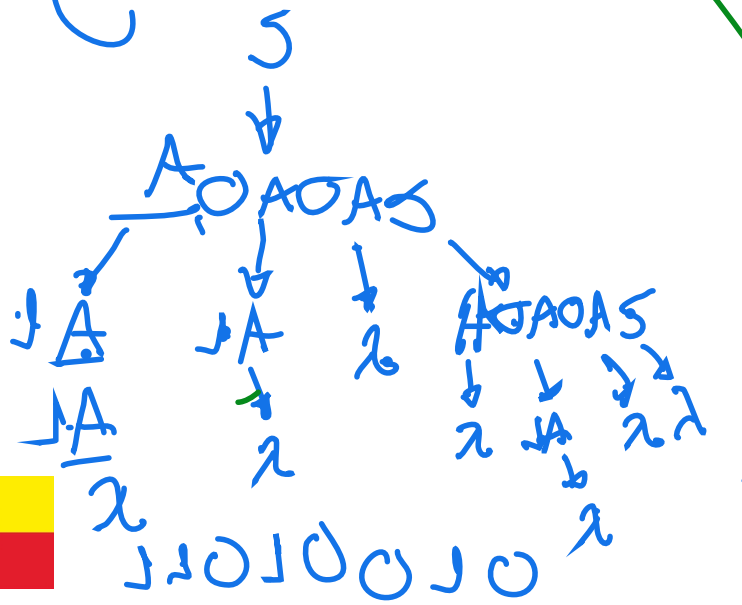


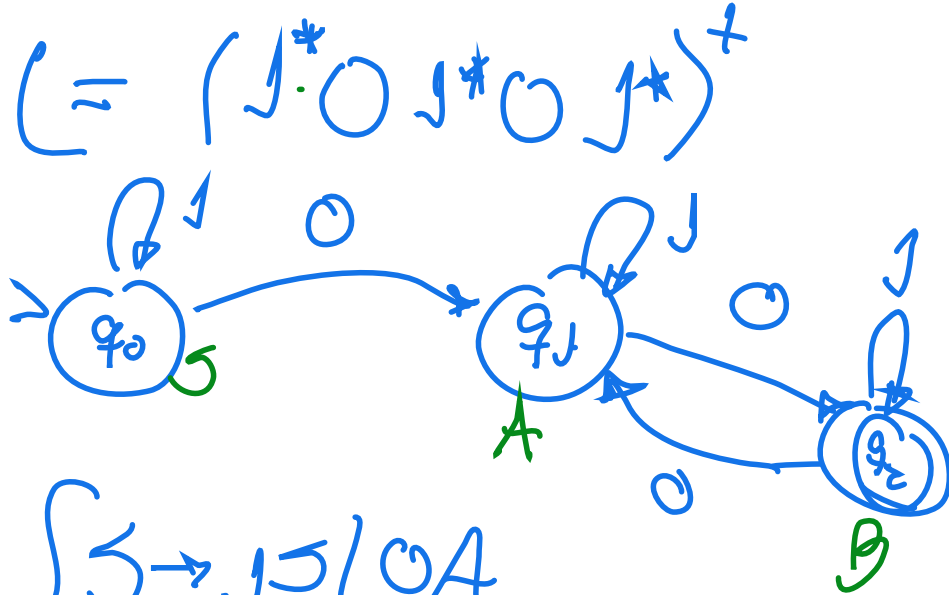
$$G = \begin{cases} S \to aS \mid bS \mid aA \mid bB \\ A \to bC \\ B \to aC \\ C \to aC \mid bC \mid \lambda \end{cases}$$

Free-Context Grammar: *Example iii*

Be L a language over the alphabet $\Sigma = \{0, 1\}$, such that L is the set of all strings that contain an even number of 0s. The context-free grammar for L is:

$$G = \begin{cases} S \to A0A0A \mid \lambda \\ A \to 1A \mid \lambda \end{cases}$$

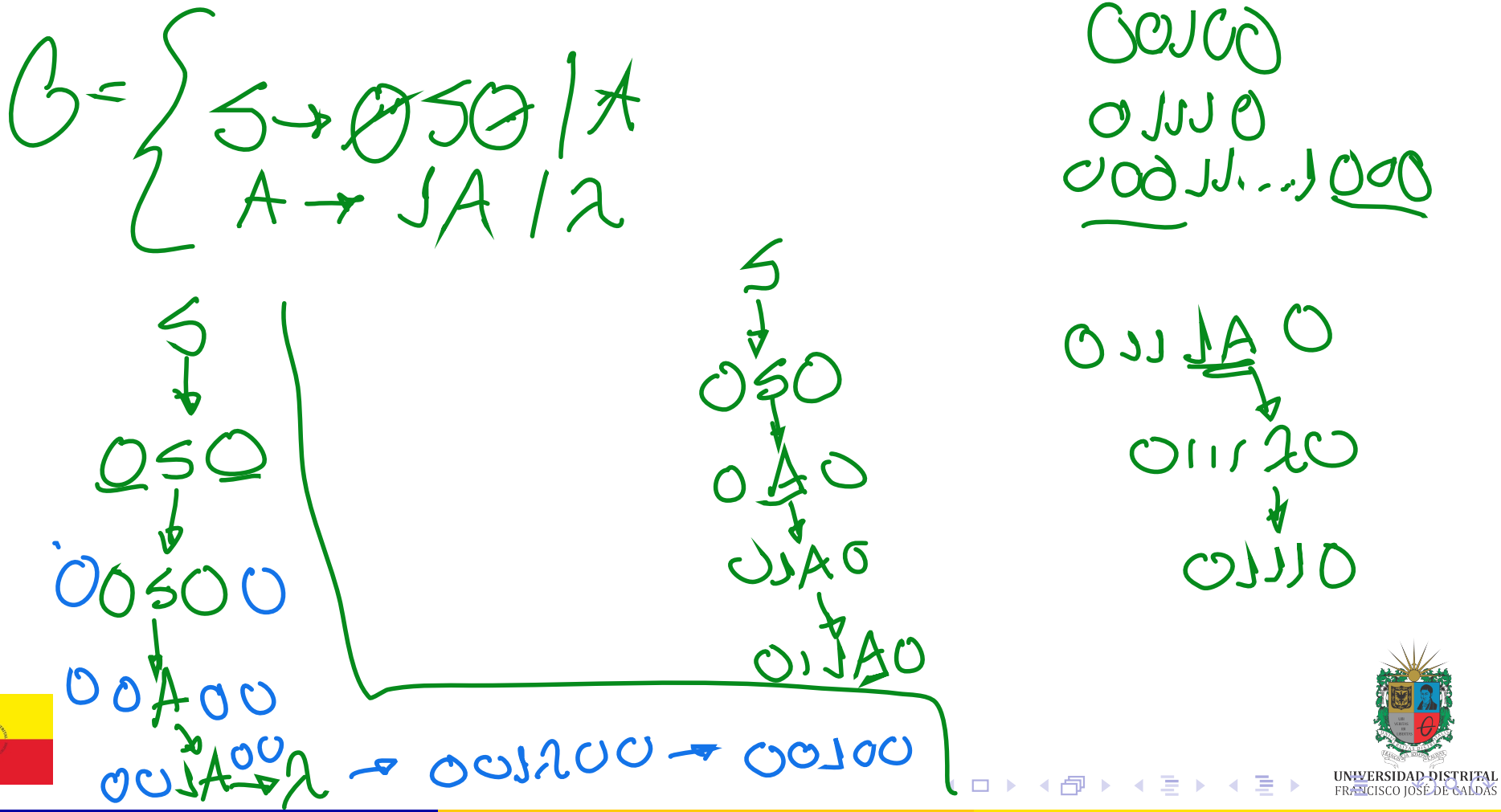




$$G = \begin{cases} S \to 1S \mid 0A \\ A \to 1A \mid 0B \\ B \to 1B \mid 0A \mid \lambda \end{cases}$$

Free-Context Grammar: *Example iv*

Be L a language over the alphabet $\Sigma = \{0, 1\}$, such that $L = 0^i 1^j 0^i$, where $i, j \geq 0$. The context-free grammar for L is:



Free-Context Grammar: *Example v*

Be L a language over the alphabet $\Sigma = \{a, b, c\}$, such that $L = a^i b^j c^{i+j}$ where $i, j \geq 1$. The context-free grammar for L is:

$$G \rightarrow \begin{cases} S \rightarrow aSc \mid aAc \\ A \rightarrow bAc \mid bc \end{cases}$$

abbccc

S

aAc

bAc

bc

abcx

abccc x

abbccc

Outline

$$G = \begin{cases} S \rightarrow \langle sujeto \rangle \langle verbo \rangle \langle complemento \rangle \\ \langle sujeto \rangle \rightarrow Yo \mid Tu \mid Él \mid Ella \mid Nosotros \mid Ellos \\ \langle Verbo \rangle \rightarrow comer \mid dormir \mid estudiar \\ \langle complemento \rangle \rightarrow mucho \mid poco \end{cases}$$

Yo dormir poco

Él comer mucho

Nosotros estudiar poco

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Thanks!

Questions?

Repo: *https://github.com/EngAndres/ud-public/tree/main/courses/computer-science-iii*

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS