

# user manual

SUBMITTED BY:  
COHIT | LIVETA | ORTICIO | REJABA

BCS31

# table of contents

03	INTRODUCTION
04	DEVELOPMENT OF SIMULATOR
05	HOW DOES IT SIMULATE
06	SIMULATOR USER GUIDE
07	DFA SIMULATION AND VALIDATION
09	PDA DISPLAY
10	CFG DISPLAY
12	GLOSSARY

# introduction

The contents of the manual refer to the study of automata, specifically, finite automata, context-free language and pushdown automata. Through JavaScript, CSS and HTML, we have come up with a web application to aid people who are interested in the subject.

The web application simulates the different automata to provide better understanding on the topic.

# **development of simulator**

The simulator was developed with the use of HTML, CSS, and JavaScript. Bootstrap was used in the development and design of the website, and JQuery was used in the handling of the different functions. and events.

These languages and libraries were used in order to have an easier development process as the developers already had HTML, CSS, and JavaScript experience prior. Bootstrap was added to have a more striking visual base than plain HTML and CSS. JQuery was used as it simplified the syntax of some of the JavaScript code.

# simulate

## how does it simulate?

The compiler simulates using a JavaScript class named "DFA". the class represents the alphabet, states, initial and final state, and transitions.

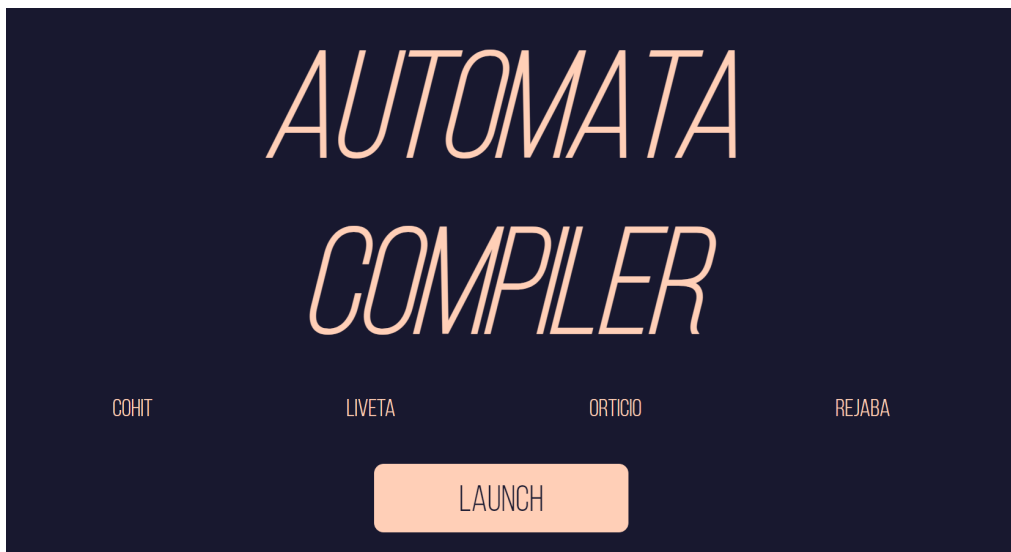
When the "Simulate" button is pressed, the DFA is executed and validated using the input string. The visited states are visually highlighted by applying and removing CSS classes to highlight the current state.

The animation is done through a for loop that iterates over the visited states and applies CSS classes with specific delays.

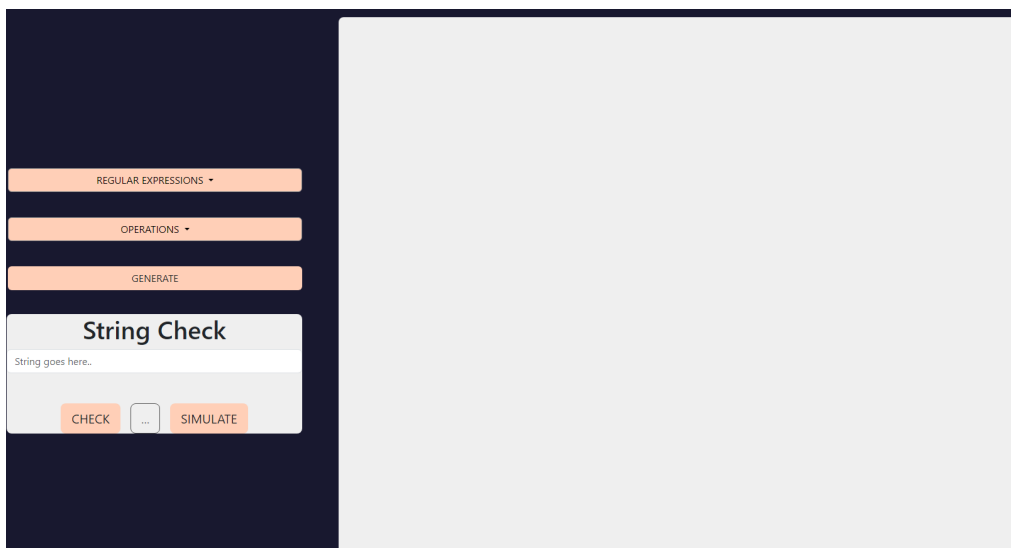
This approach makes a user-friendly visualization of the DFA validation process, which enhances the user's understanding of the DFA model.

# simulator user guide

The simulator has a landing page displaying the names of the group members and the launch button.

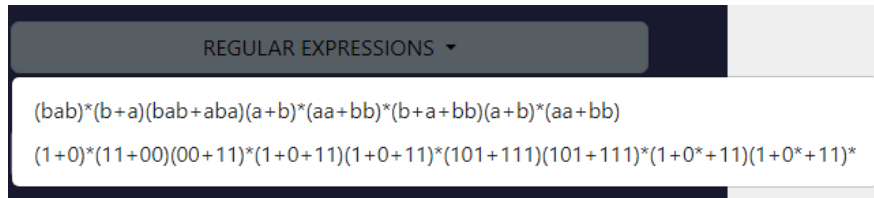


Pressing the "LAUNCH" button will send the user to the empty calculator.

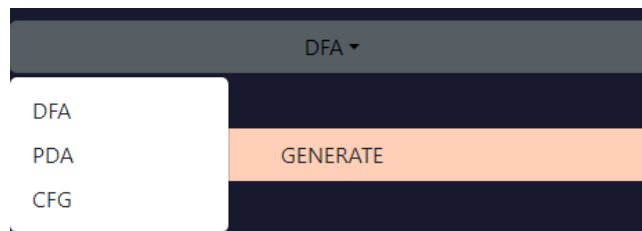


# DFA Simulation and Validation

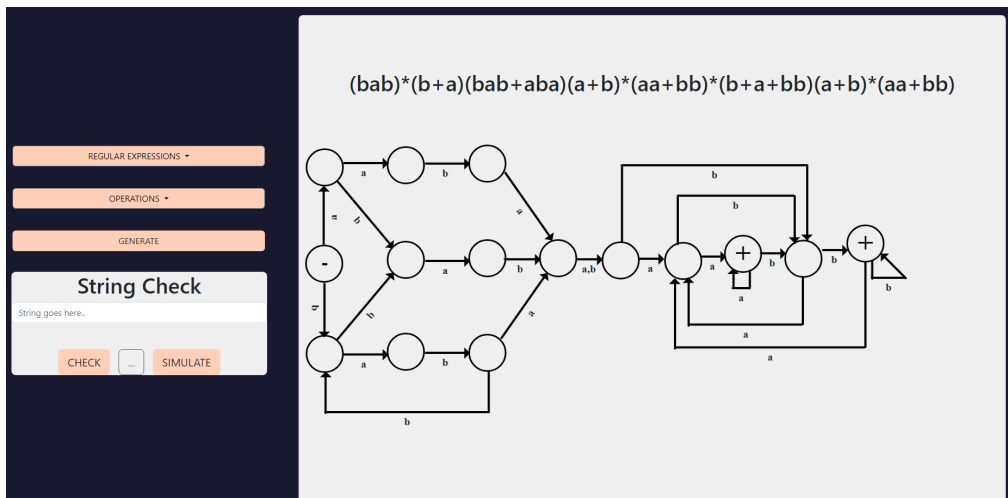
1. Select a Regular Expression from the first dropdown box.



2. Select the DFA operation:

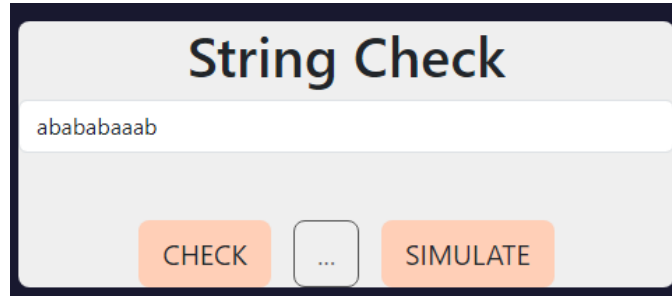


3. Press generate to show the DFA of the chosen Regular Expression.



# DFA Simulation and Validation

4. Input a string in the String Check box.

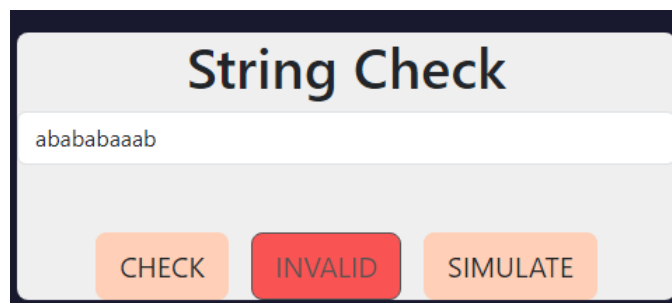


String Check

abababaaab

CHECK ... SIMULATE

5. Press the Check button to see if your inputted string is valid or invalid.

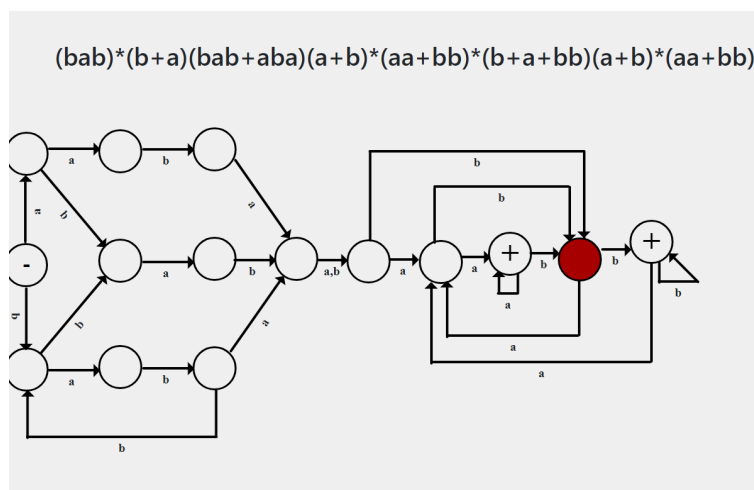


String Check

abababaaab

CHECK INVALID SIMULATE

6. Press the Simulate button to see a visual representation of the process in your DFA.

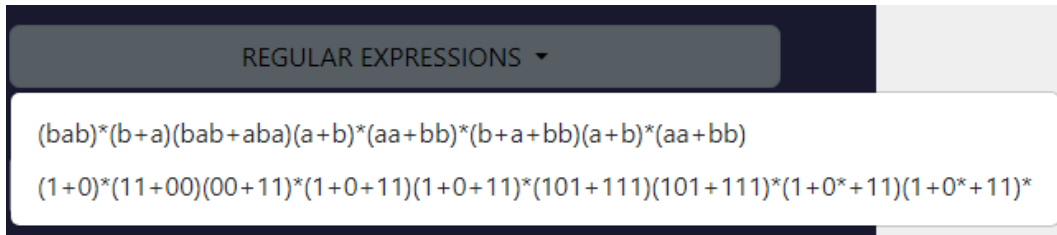


(Image above shows the final state only)

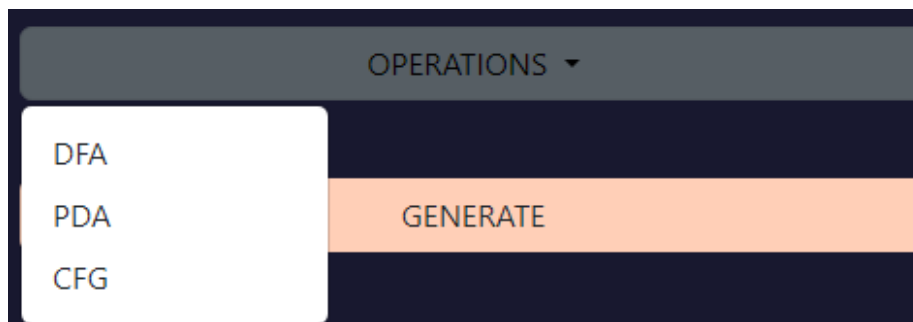


# PDA Display

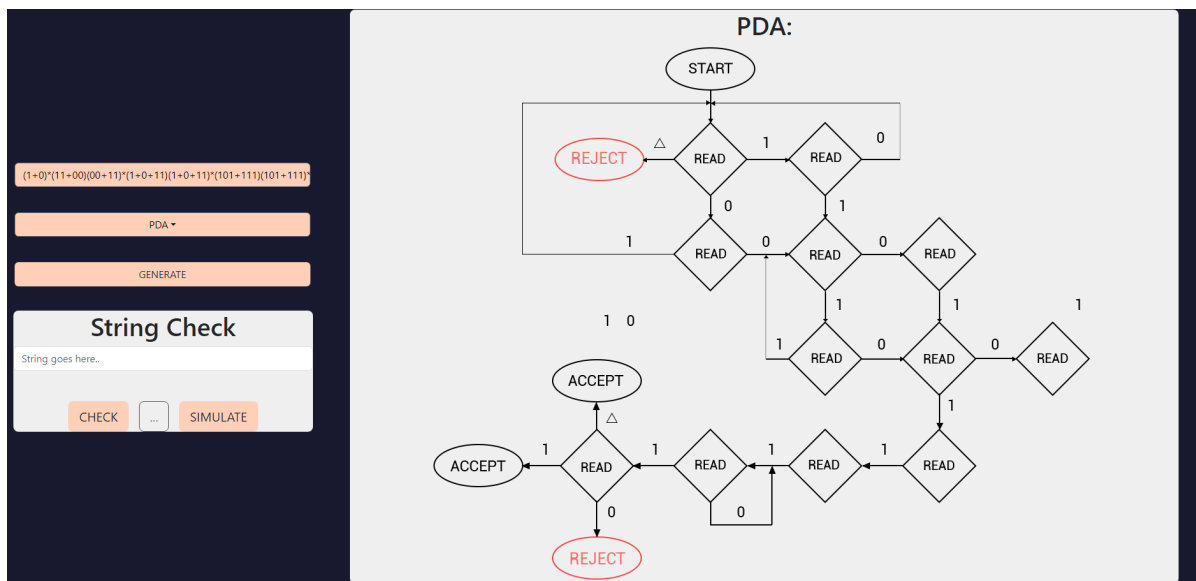
1. Select a Regular Expression from the first dropdown box.



2. Select PDA from the second dropdown box.

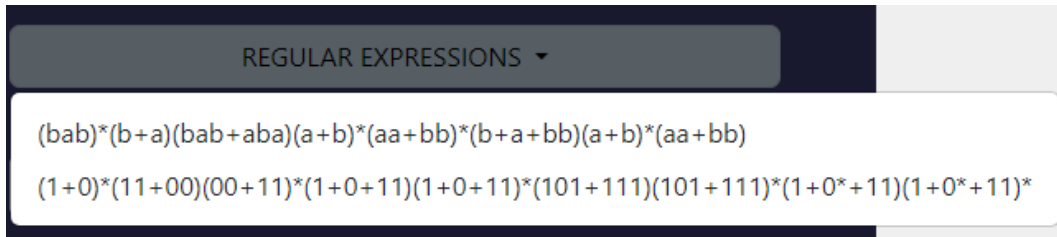


3. Press generate to display the PDA.



# CFG Display

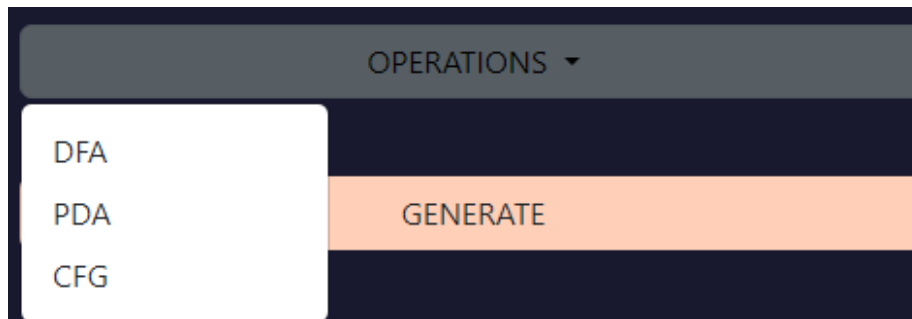
1. Select a Regular Expression from the first dropdown box.



REGULAR EXPRESSIONS ▾

- (bab)\*(b+a)(bab+aba)(a+b)\*(aa+bb)\*(b+a+bb)(a+b)\*(aa+bb)
- (1+0)\*(11+00)(00+11)\*(1+0+11)(1+0+11)\*(101+111)(101+111)\*(1+0\*+11)(1+0\*+11)\*

2. Select CFG from the second dropdown box.

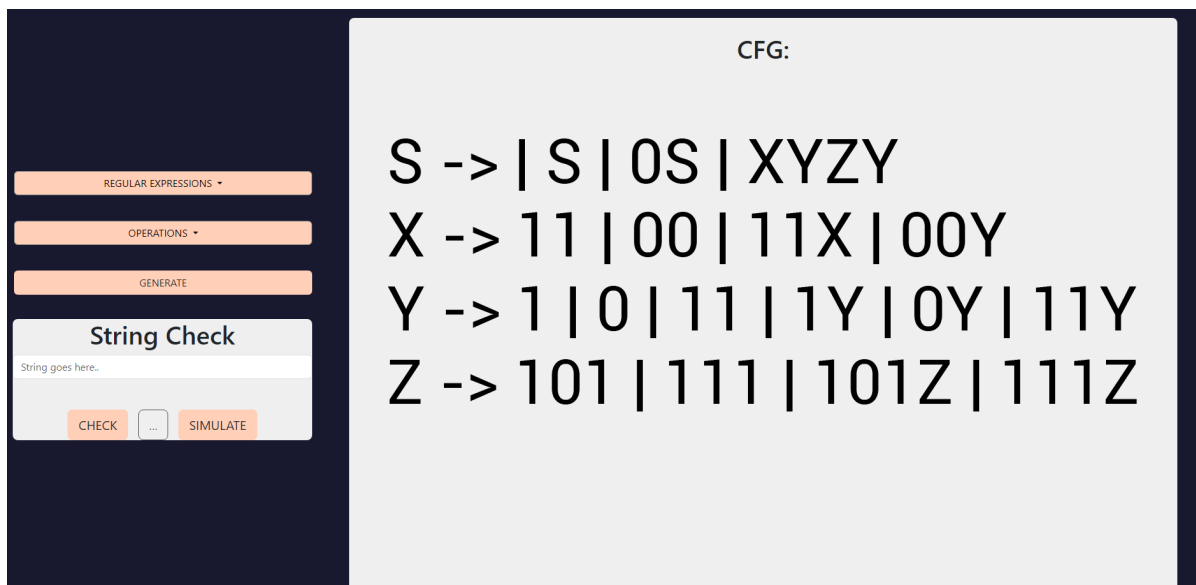


OPERATIONS ▾

- DFA
- PDA
- CFG

GENERATE

3. Press generate to display the CFG.



CFG:

S -> | S | OS | XYZY

X -> 11 | 00 | 11X | 00Y

Y -> 1 | 0 | 11 | 1Y | 0Y | 11Y

Z -> 101 | 111 | 101Z | 111Z

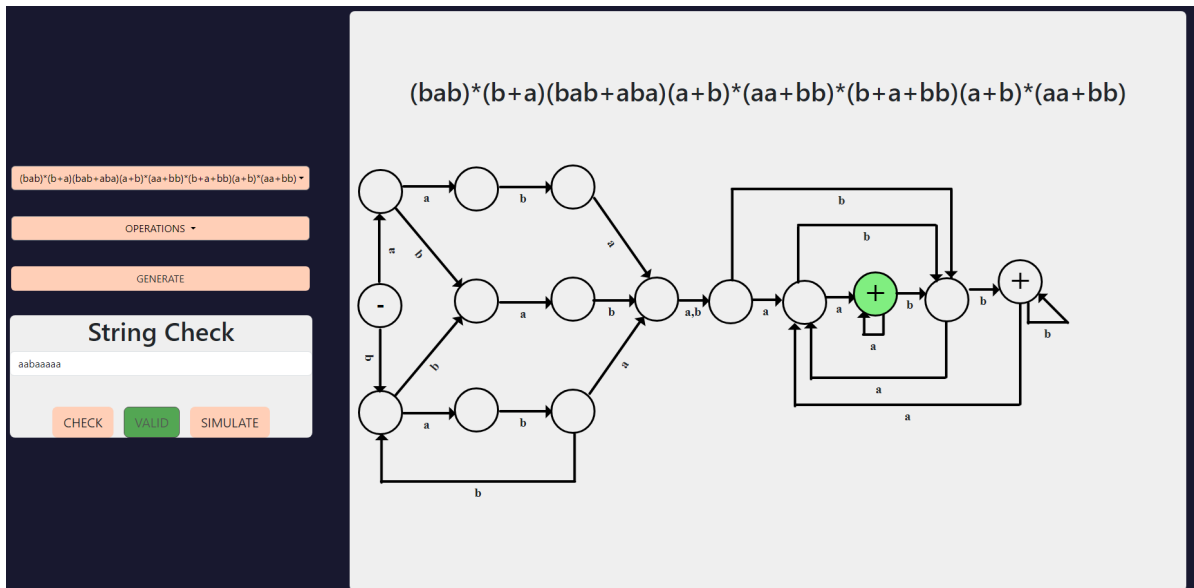
String Check

String goes here.

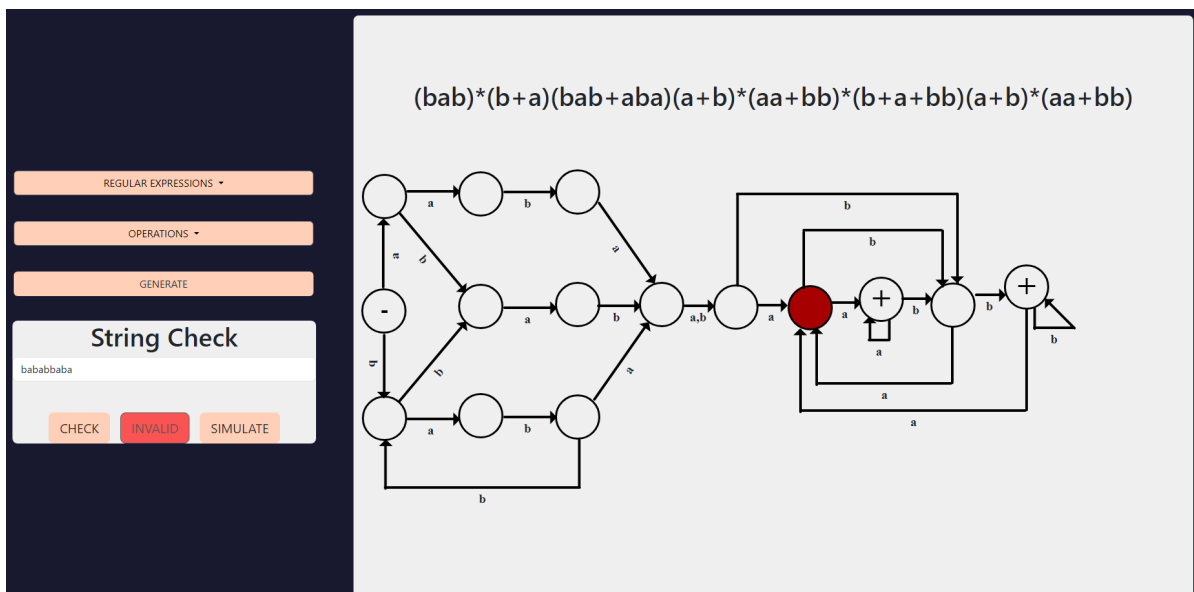
CHECK ... SIMULATE

# sample outputs

Valid string and DFA for first RegEx:



Invalid string and DFA for first RegEx:



# glossary

**Alphabets** refer to any finite set of symbols. e.g.  $\Sigma = \{0, 1\}$

**Automata** - an abstract model of a digital computer.

**Context-Free Grammar** - are syntactic models of languages, similar to regular expressions

**Context-Free Language** - is the language generated by the CFG. The set of all strings of terminals that can be produced from the start symbol  $S$  using the productions as substitutions.

**Deterministic Finite Automaton (DFA)** - for a particular input character, the machine goes to one state only. Null move is not allowed.

**Finite Automata** - simplest abstract machine to recognize patterns with input taken from the given alphabet.

**Grammars** - are useful models when designing software that processes data with a recursive structure.

**Lambda** - the language consisting of only the empty string.

**Language** - is the set of strings chosen from some alphabet.

**Non-deterministic Finite Automaton** - similar to DFA except null move is allowed and its ability to transmit to any number of states for a particular input.

**Pushdown Automata** - is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar.

**Regular Expression** - denote the structure of data, especially text strings.

**String** - is a finite sequence of symbols from some alphabet e.g. 01001

**A context-free grammar is a collection of three things:**

- > An alphabet  $\Sigma$  of letters called **terminals** from which we are going to make strings that will be the words of a language
- > A set of symbols called nonterminals, one of which is the symbol  $S$ , standing for "start here"
- > A finite set of productions or substitution rules of the form  $A \rightarrow a$

**$S \rightarrow aX$**

Nonterminals:  $S, X$

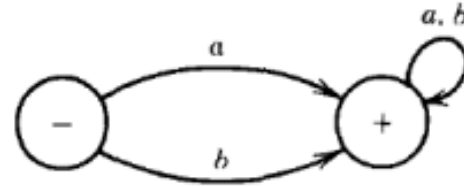
Terminal:  $a$

**A finite automaton is a collection of three things:**

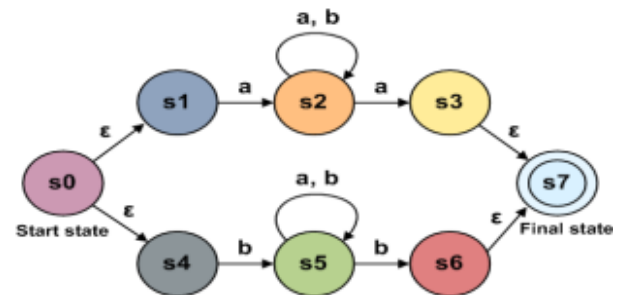
- > A finite set of states, one of which is designated as the initial state, called the start state, and some of which are designated as final states.
- > An alphabet  $\Sigma$  of possible input letters, from which are formed strings, that are to be read one letter at a time
- A finite set of transitions that tell for each state and for each letter of the input alphabet which state to go to next

## Deterministic Finite Automaton

DFA with  $\Sigma = \{a, b\}$  accepts all strings starting with either  $a$  or  $b$ .



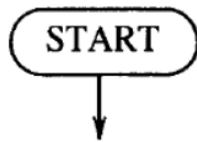
## Non-Deterministic Finite Automaton



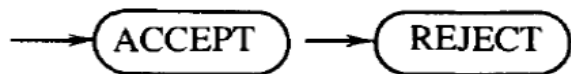
**A pushdown automaton is a collection of eight things:**

- > An alphabet  $\Sigma$  of input letters
- > An input TAPE (infinite in one direction). Initially the string of input letters is placed on the TAPE starting in cell  $i$ . The rest of the TAPE is blank
- > An alphabet  $\Gamma$  of STACK characters.
- > A pushdown STACK (infinite in one direction). Initially the STACK is empty (contains all blanks).

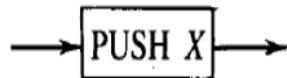
> One START state that has only out-edges, no in-edges



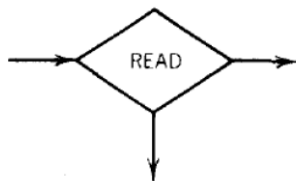
> Halt states of two kinds: some ACCEPT and some REJECT. They have in-edges and no out-edges.



> Finitely many nonbranching PUSH states that introduce characters onto the top of the STACK. They are of the form



> Finitely many branching states of two kinds: States that read the next unused letter from the TAPE



States that read the top character of the STACK

