USER MANUAL for Automata Simulator

Members:

Cristal, Katherine

De Guzman, Benedic

Uydiangco, Benedict

Victorio, Chris Daniel

June 16, 2023

Table of Contents

Introduction

We have developed a website that serves as a simulator for Automata theory specifically, the deterministic finite automata (DFA), context-free grammar (CFG) and pushdown automata (PDA). This web application facilitates interactive exploration of fundamental concepts in automata theory. Users are able to choose between two RegEx options, input strings for validation, and observe the step-by-step procedures involved in string validation. The program boasts a user-friendly interface that allows for customization of inputs, simulation of string processing, and delivers results for PDA and CFG. Moreover, it features an interactive display to enhance comprehension and provide practical learning opportunities for these powerful computational models.

Development of the web application

The web application is created by combining several technologies to offer a dynamic and visually appealing user experience. To accomplish this goal, we used HTML, CSS, JavaScript, jQuery, and Bootstrap 5.3 in our web application.

The HTML files such as index, calculator, number, and letter in our web application include the structure and content of the individual pages, constituting the foundation of your program.

Our application's CSS files, such as calculator, letters, styles, and numbers, are in charge of decorating the various components and pages, ensuring a visually pleasing and consistent appearance. We incorporated Bootstrap into our application to reduce development time and effort while maintaining a professional and polished appearance

The logic and functionality for numerous aspects, such as the calculator functionality, managing letters and numbers, and constructing a Deterministic Finite Automaton (DFA), are contained in the JavaScript files in our web application, including calculator, letters, numbers, and DFA. Additionally, we used jQuery to improve the functionality and user experience of our web application, making it easier to design dynamic and responsive features.

DFA.js was used to implement the Deterministic Finite Automaton (DFA) in the web application. We created a DFA class to establish the properties of the DFA object using the specified parameters: alphabet, states, initial state, end state, and transitions.

The execute method executes the DFA given an input string. It loops through the input string, modifying the current state according to the transition rules specified in the transitions property. The path array stores the visited states. Finally, it determines whether the final state has been reached and returns true if it has, indicating that the input string has been accepted by the DFA.

Overall, by combining these technologies and incorporating the DFA implementation, we were able to construct a dynamic and visually appealing online application with better interactivity and pattern recognition capabilities.

Automata

- Is the study of abstract computing devices, or "machines". A study of mathematical models of a computer which can determine whether a particular string is in a language. Concept that describes how machines mimic human behavior.
- Is an abstract model of a digital computer. Every automaton includes some essential features. It has a mechanism for reading input. It is also assumed to operate in a discrete time frame. A mathematical model for a finite state machine. A finite state machine is a machine that, given an input, jumps through a series of states according to a transition function that tells the automaton which state to go next given a current state and a current symbol.

Note: There are important notations that are not automaton-like, but play an important role in the study of automata and their application.

1. Grammars

– are useful models when designing software that processes data with a recursive structure, the best known example is a "parser", the component of a compiler that deals with recursively nested features of the typical programming languages.

2. Regular Expression

– denote the structure of data, especially text strings.

Alphabets

- An alphabet is any finite set of symbols.

∑ - it denotes alphabets

Common Alphabets include:

1. ∑ = { 0,1 } , binary alphabet
2. ∑ = { a,b, …., z }, the set of all lower-case letters
3. The set of all ASCII characters, or the set of all printable ASCII characters.

String

- Is a finite sequence of symbols from some alphabet, e.g. 01001. The empty string with zero occurrences of symbols is denoted by Λ

RegEx (Regular Expression)

A sequence of pattern that defines a string, are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string. It is the most effective way to represent any language.

The symbols that appear in regular expressions are the letters of the alphabet Σ, the symbol for the null string Λ, parentheses, the star operator, and the plus sign.

DFA (Deterministic Finite Automaton)

In a DFA, for a particular input character, the machine goes to one state only. A transition function is defined on every state for every input symbol. Also in DFA null (or Λ) move is not allowed, i.e., DFA cannot change state without any input character.

There can be many possible DFAs for a pattern. A DFA with minimum number of states is generally preferred.

A DFA consists of the following components:

1. Alphabet	**∑={1,0}or ∑={a,b}**
2. State	○
3. Transition Function	→
4. Start State	⊖
5. Final State	⊕
6. Trap State	Ⓣ

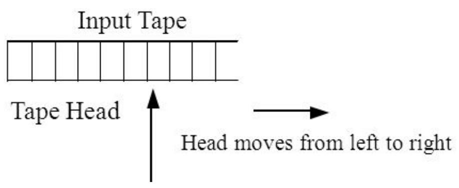
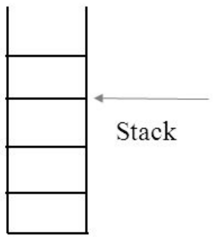
PDA (Pushdown Automata)

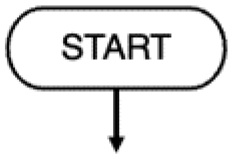
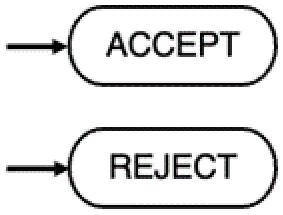
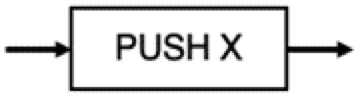
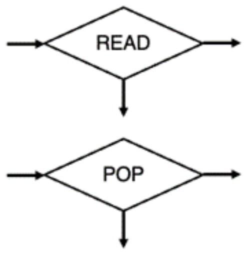
Is a finite automata with extra memory called stack which helps Pushdown automata to recognize Context Free Languages. Is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar. Can remember an infinite amount of information.

A pushdown automaton, PDA, is a collection of eight things:

1. An alphabet **Σ** of input letters.
2. 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.
3. An alphabet Γ of STACK characters.
4. A pushdown STACK (infinite in one direction). Initially the STACK is empty (contains all blanks).
5. One START state that has only out-edges, no in-edges.
6. Halt states of two kinds: some ACCEPT and some REJECT. They have in-edges and no out-edges.
7. Finitely many nonbranching PUSH states that introduce characters onto the top of the STACK. They are of the form.
8. Finitely many branching states of two kinds:
    a. States that read the next unused letter from the TAPEStates that read the top character of the STACK

A PDA consists of the following components:

1. **Alphabet**	**∑={1,0}or ∑={a,b}**
2. **Input tape**	
3. **Stack**	

4. Start State	
5. Halt States	
6. Push State	
7. Branching State	

Context Free Grammar

Are syntactic models of languages, similar to regular expressions. Through grammar rules, CFGs describe how variables can be replaced by variables and terminals to derive strings in the language of the grammar.

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

1. An alphabet Σ of letters called terminals from which we are going to make strings that will be the words of a language.
2. A set of symbols called non-terminals, one of which is the symbol S, standing for "start here".

3. A finite set of productions or substitution rules of the form A à a.

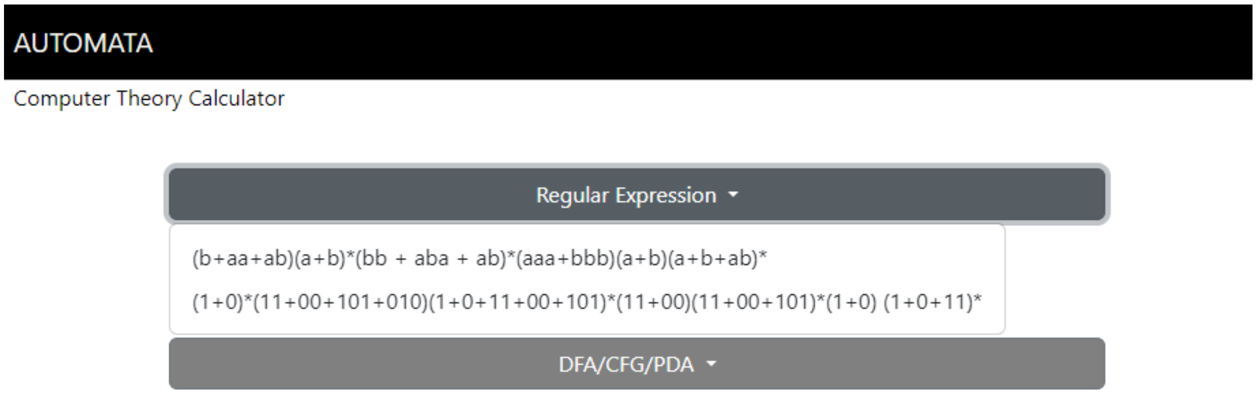
A CFG consists of the following components:

1. Non-terminal symbols	**XY**
2. Terminal symbols	**a | b**
3. Production rules	$S \rightarrow XY$ $X \rightarrow aX \mid bX$ $Y \rightarrow aY \mid bY$
4. Start icon	$S \rightarrow$

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.

Using the DFA Simulator

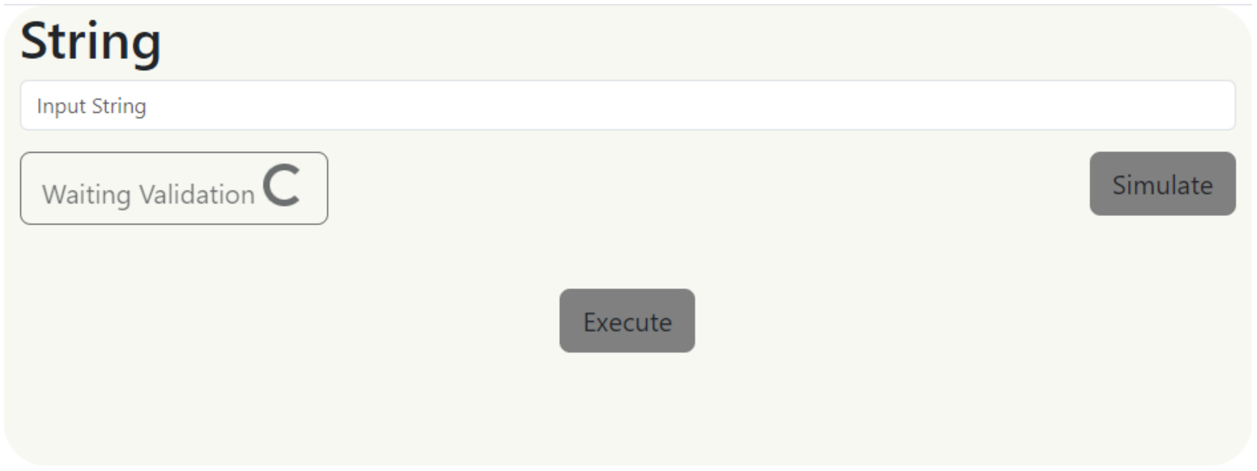
1. Select a Regular Expression and choose between DFA/CFG/PDA to simulate.



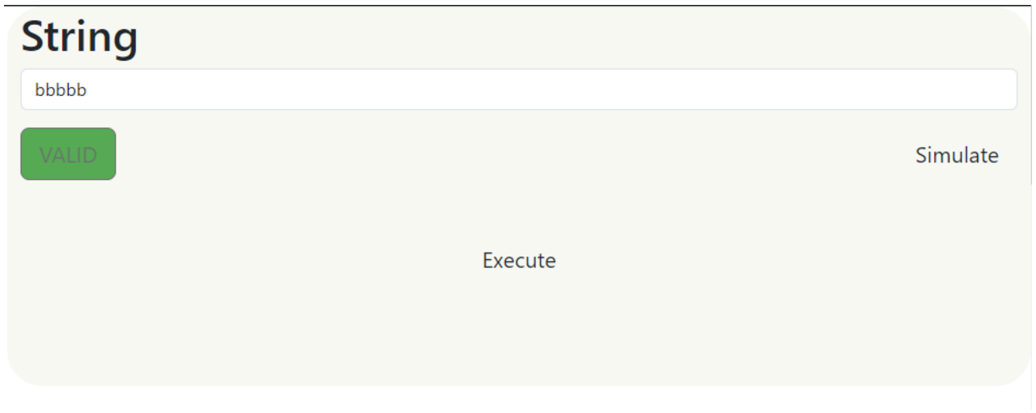
(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

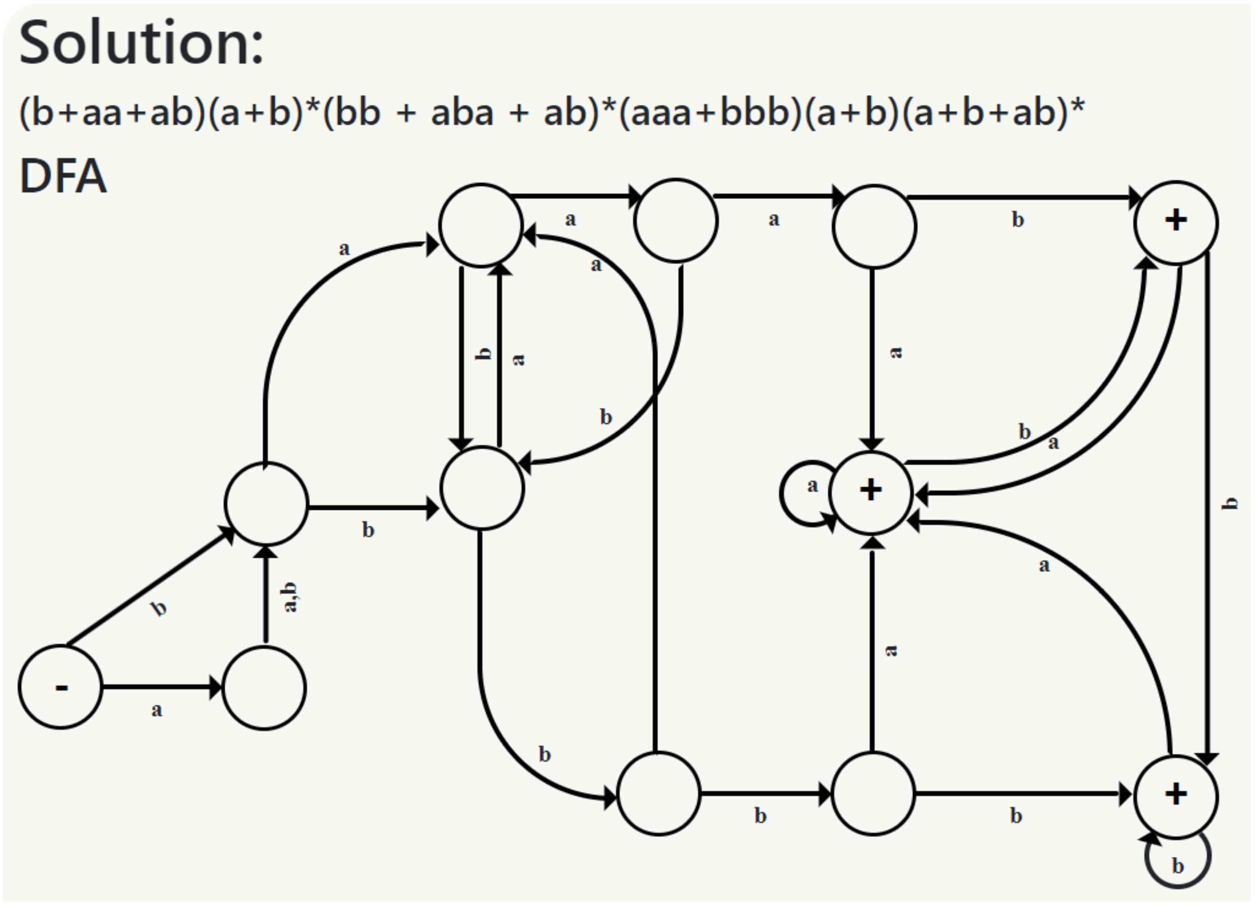
(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*

2. Input a string to validate.



3. Simulate the string in the DFA.

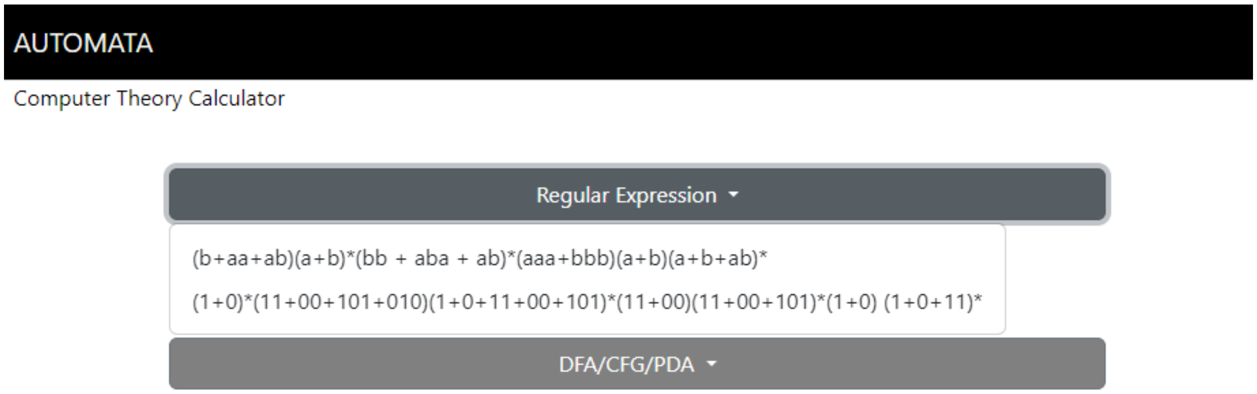



Solution:

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

DFA

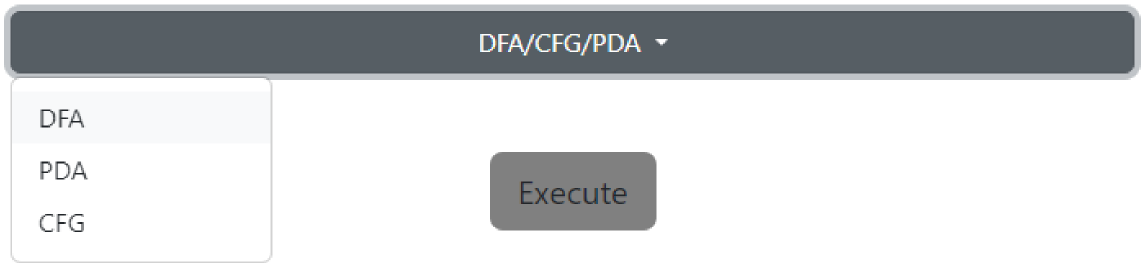
4. In order to execute the CFG/PDA, you must pick first a RegEx accordingly,


AUTOMATA

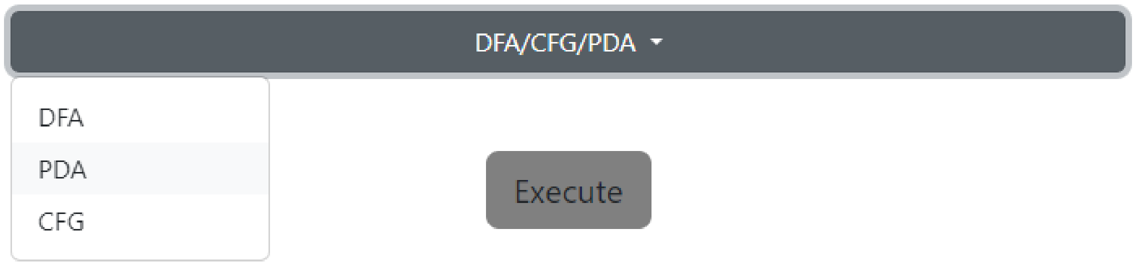
Computer Theory Calculator

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)* then secondly choose DFA then click simulate, to display the DFA.



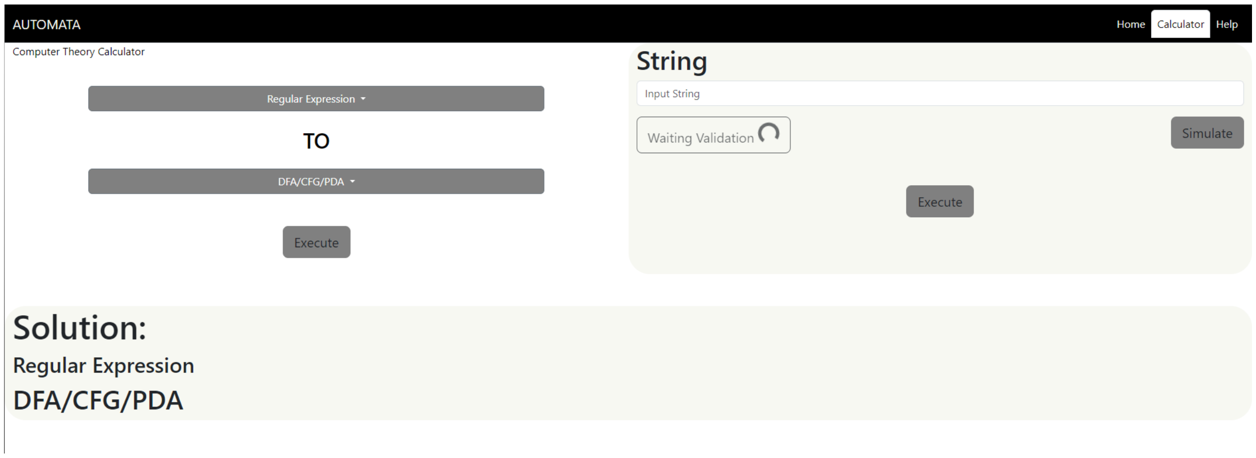
then click again a RegEx that you have chosen before, and then click PDA to display the DFA of the chosen RegEx.



To Display the CFG, you must click again the chosen RegEx before, and then click CFG.

Sample Outputs

- Click first your desired RegEx to display the DFA/CFG/PDA of it.


AUTOMATA

Computer Theory Calculator

String

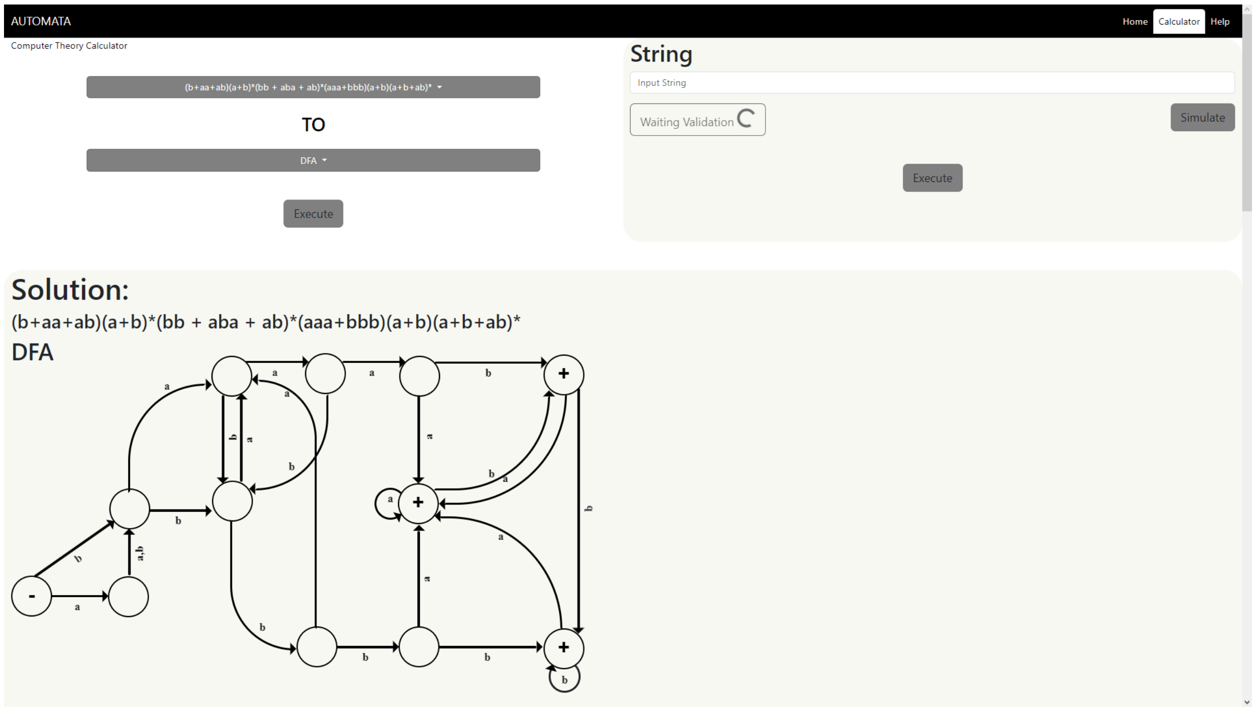
Solution:

Regular Expression

DFA/CFG/PDA

- Display the DFA for RegEx

  (b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*


AUTOMATA

Computer Theory Calculator

String

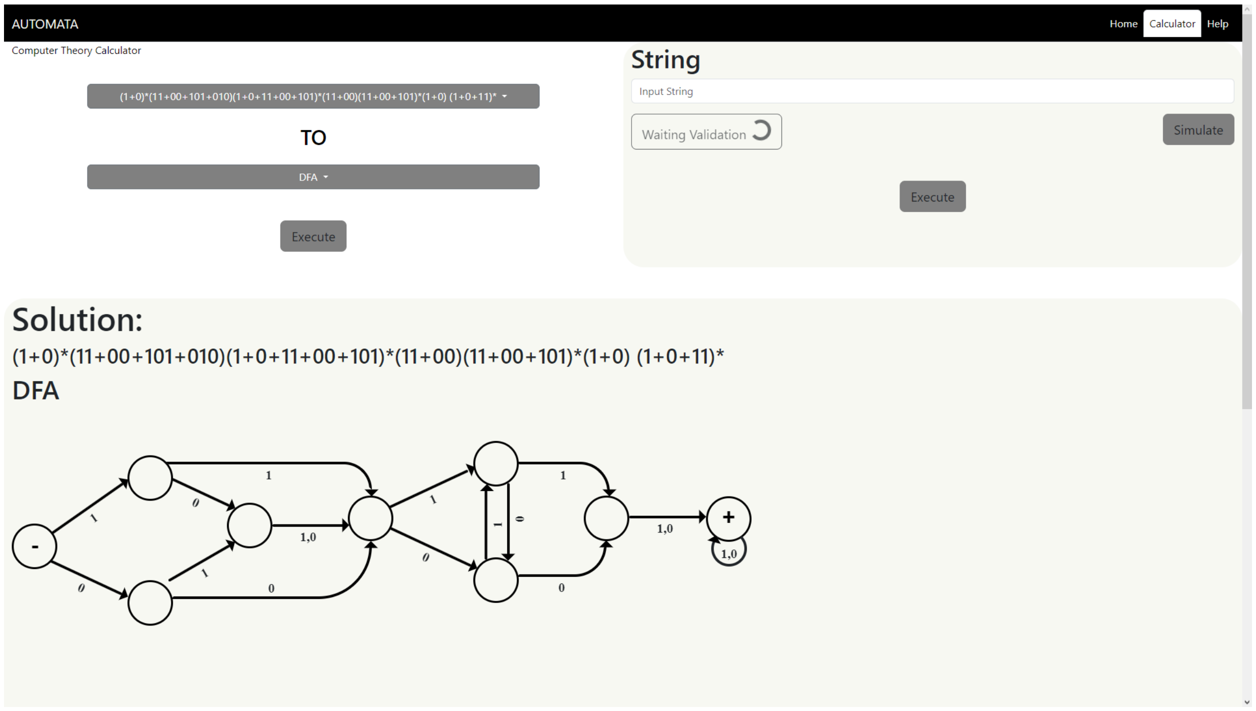
Solution:

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

DFA

- Display the DFA for RegEx

  (1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*



AUTOMATA

Computer Theory Calculator

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*

TO

DFA

String

Input String

Waiting Validation

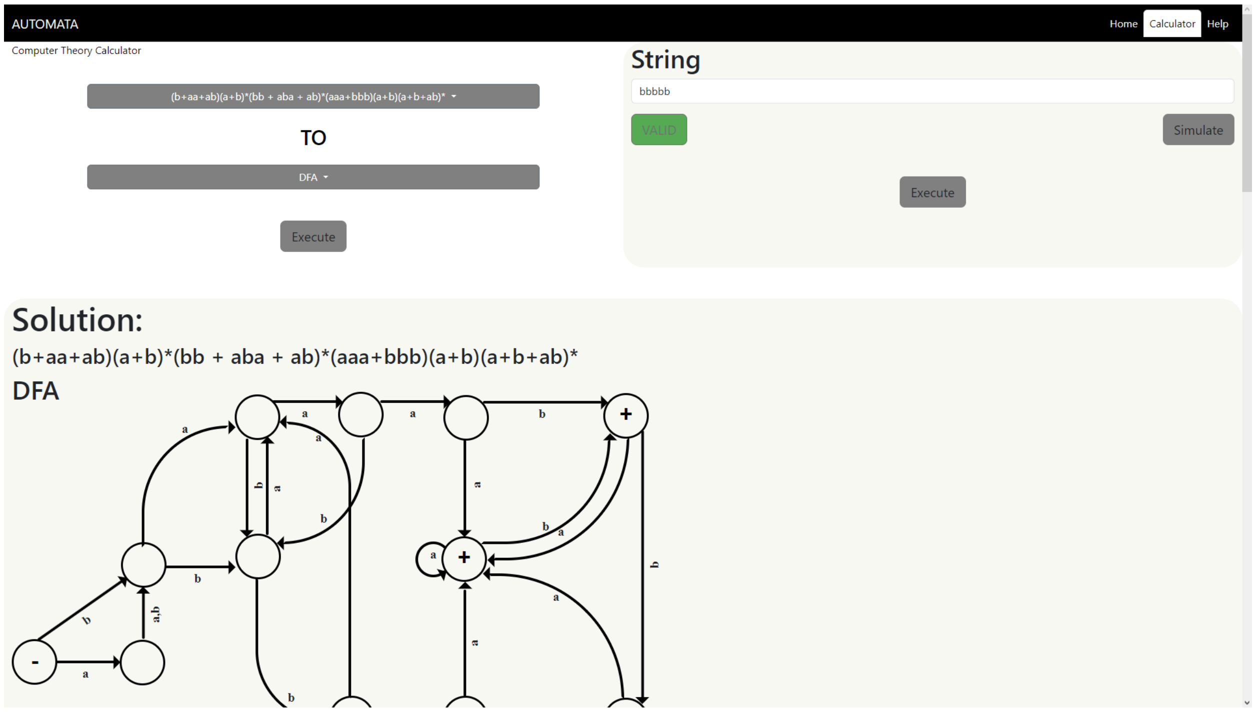
Solution:

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*

DFA

- Checking for valid and invalid strings for RegEx

  (b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*



AUTOMATA

Computer Theory Calculator

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

TO

DFA

String

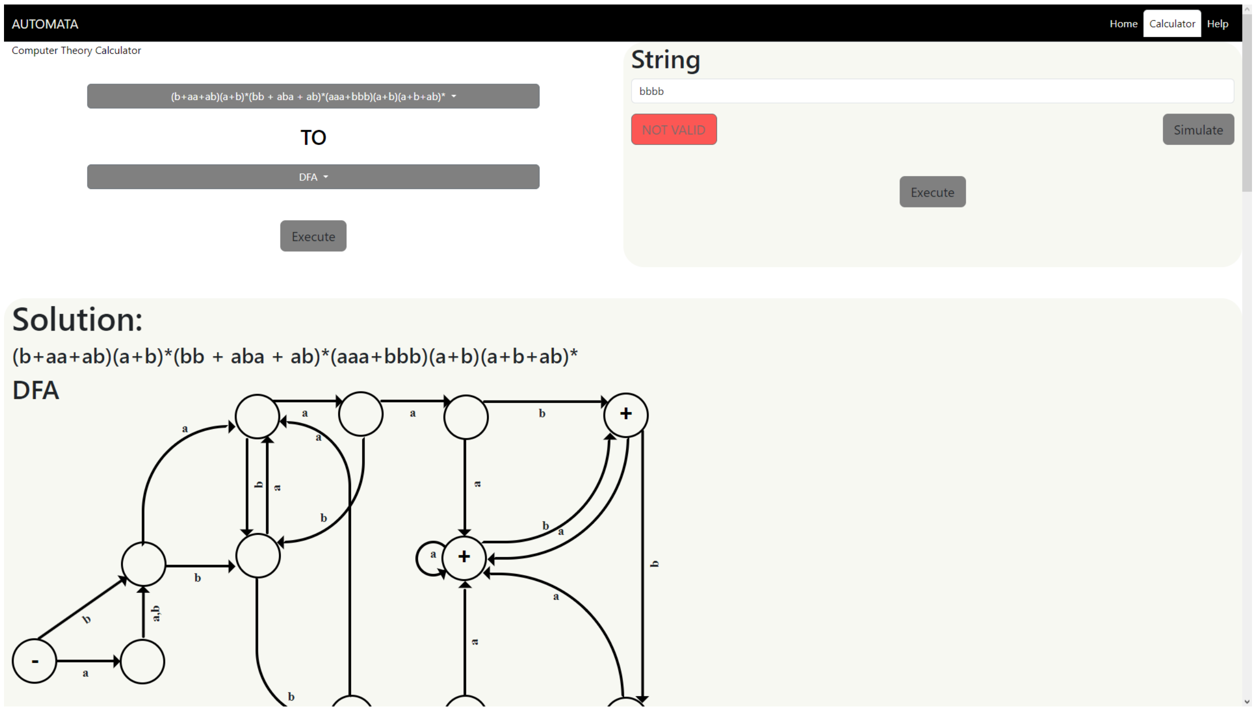
bbbbb

VALID

Solution:

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

DFA


AUTOMATA

Computer Theory Calculator

Home Calculator Help

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

TO

DFA

Execute

String

bbbb

NOT VALID

Simulate

Execute

Solution:

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

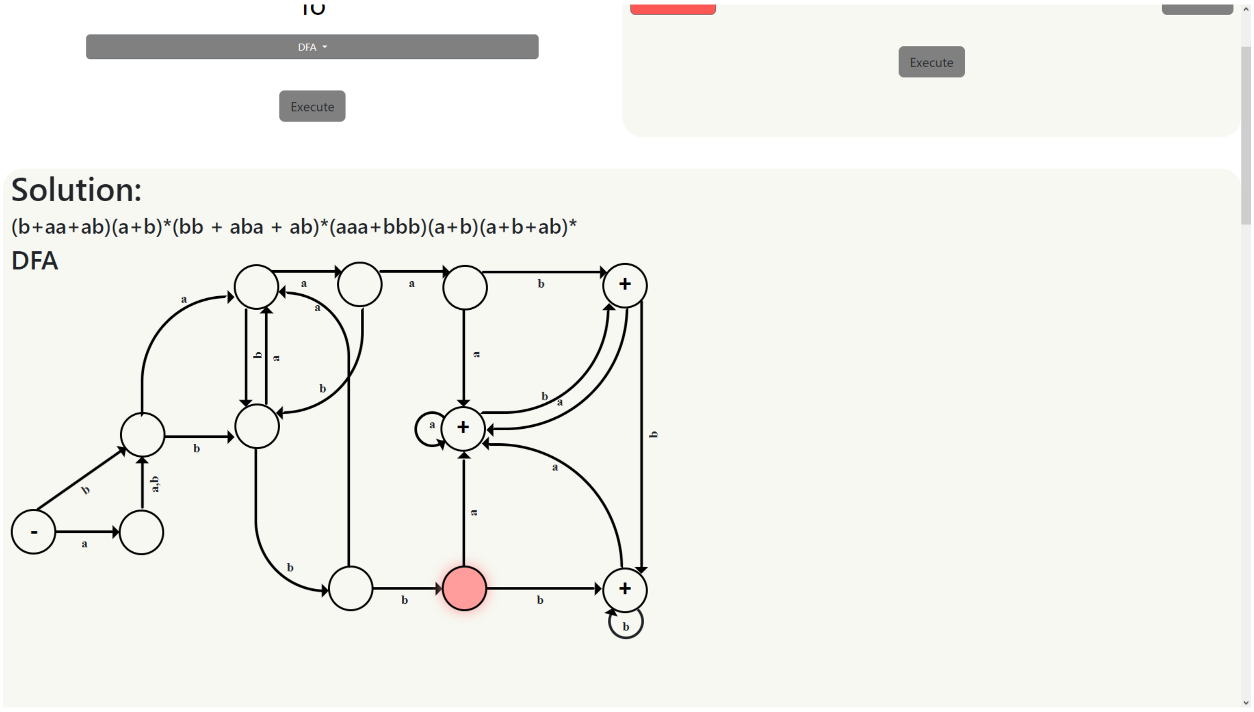
DFA

- Checking the valid and invalid strings to simulate in the DFA

For RegEx

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

a. Invalid


TO

DFA

Execute

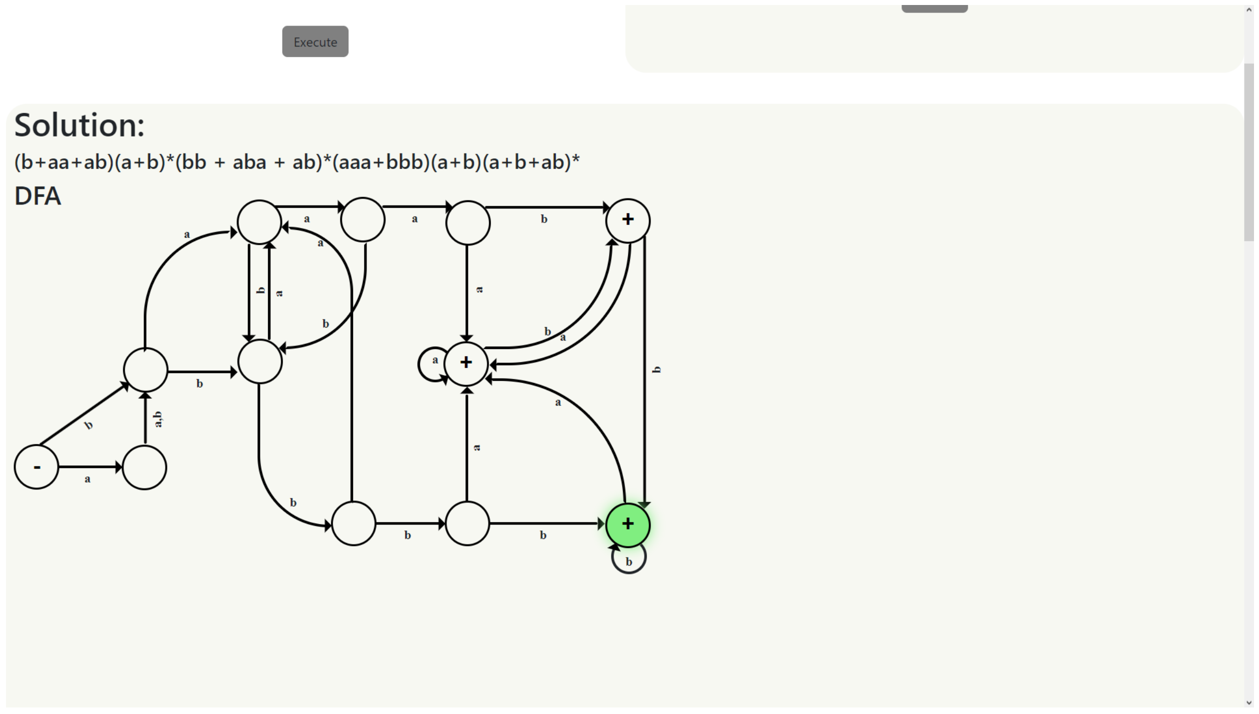
Execute

Solution:

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

DFA

b. Valid



Solution:

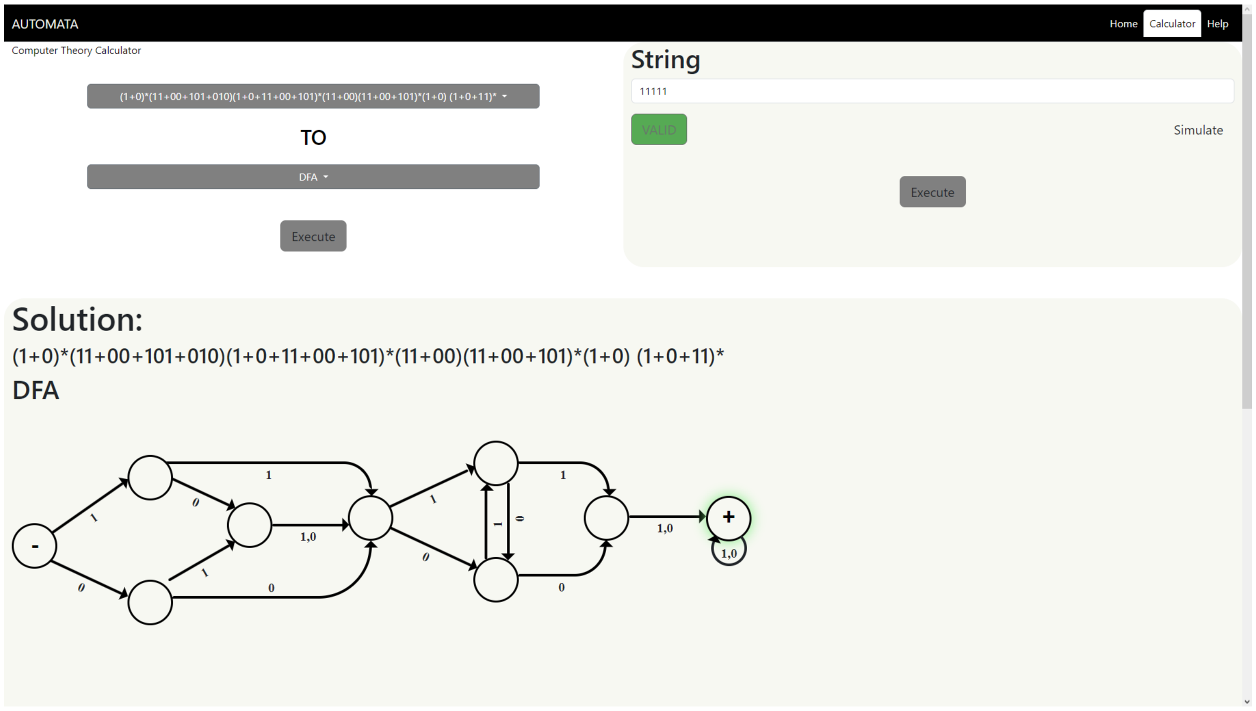
(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*

DFA

For RegEx

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*

Valid


AUTOMATA

Home Calculator Help

Computer Theory Calculator

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*

TO

DFA

Execute

String

11111

VALID

Simulate

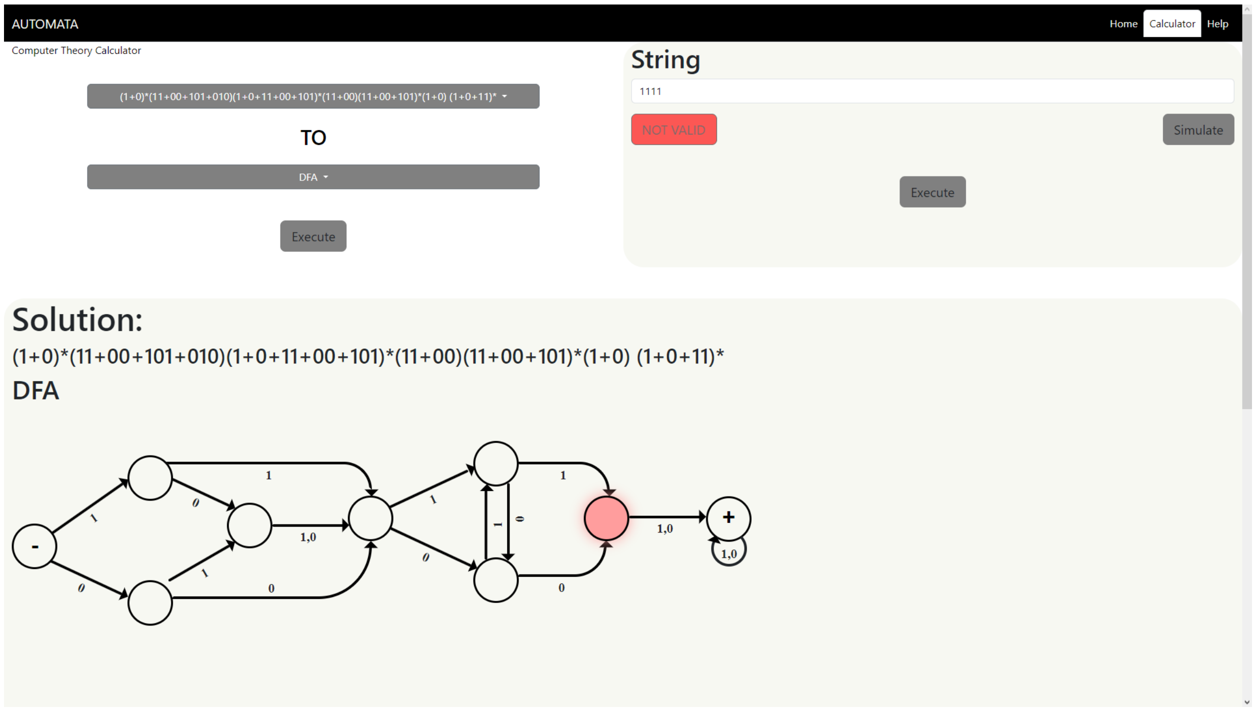
Execute

Solution:

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*

DFA

Invalid


AUTOMATA

Home Calculator Help

Computer Theory Calculator

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*

TO

DFA

Execute

String

1111

NOT VALID

Simulate

Execute

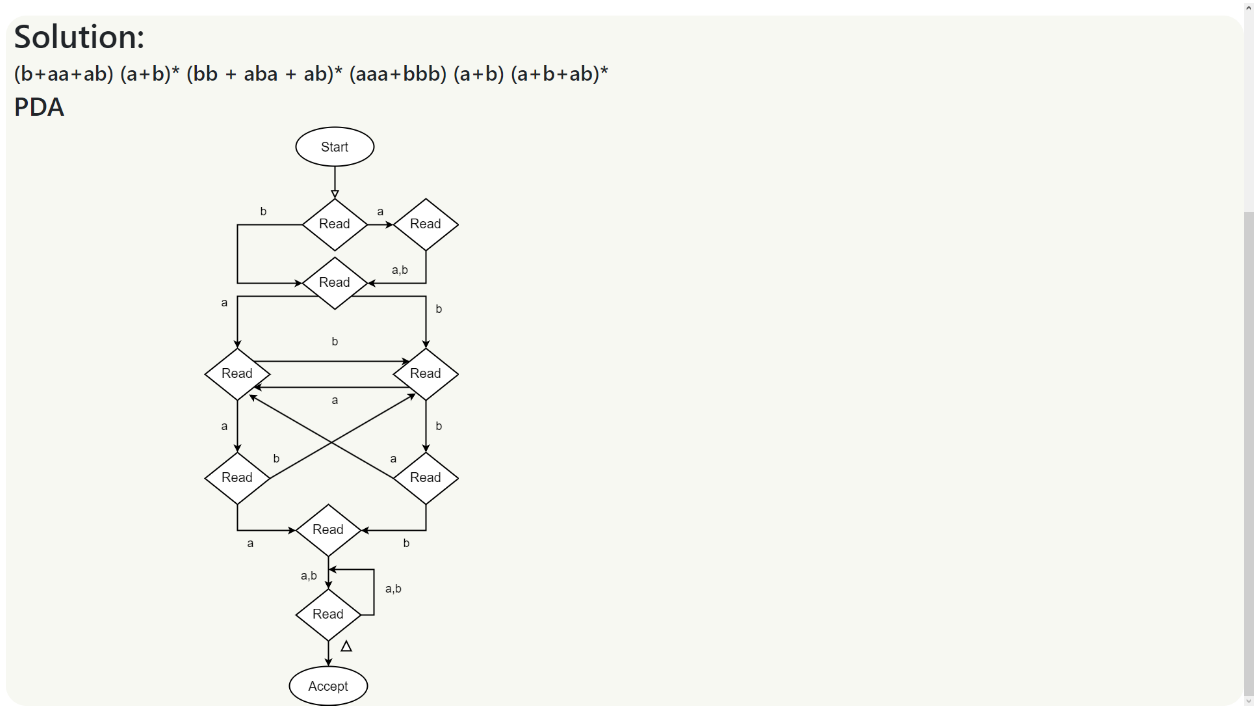
Solution:

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*

DFA

Displaying the PDA for RegEx

(b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*



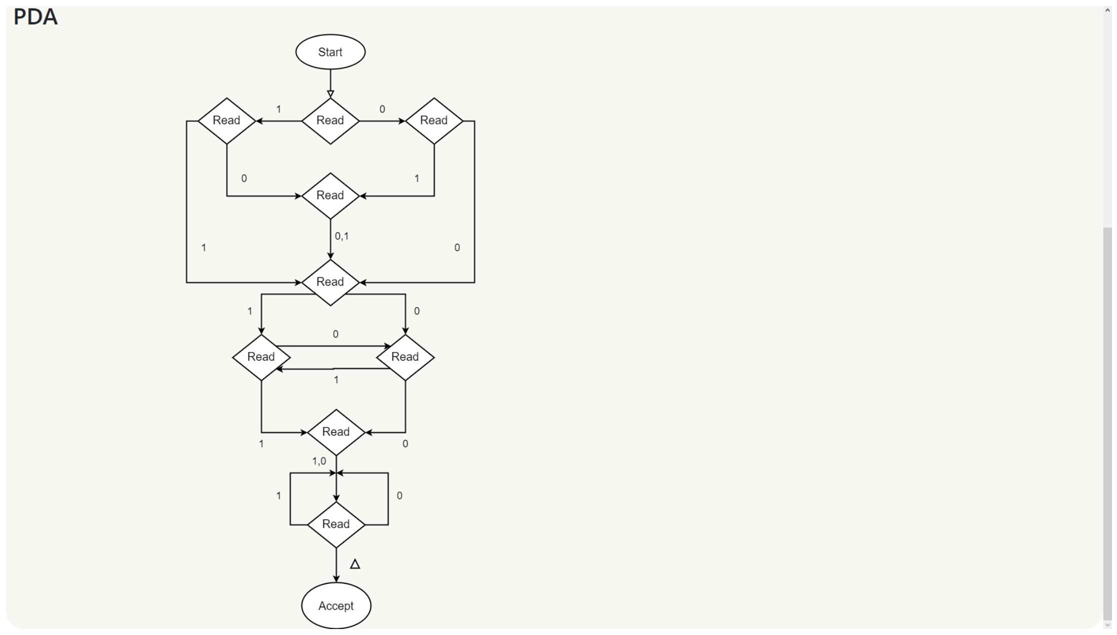
Solution:

(b+aa+ab) (a+b)* (bb + aba + ab)* (aaa+bbb) (a+b) (a+b+ab)*

PDA

Displaying the PDA for RegEx

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0) (1+0+11)*

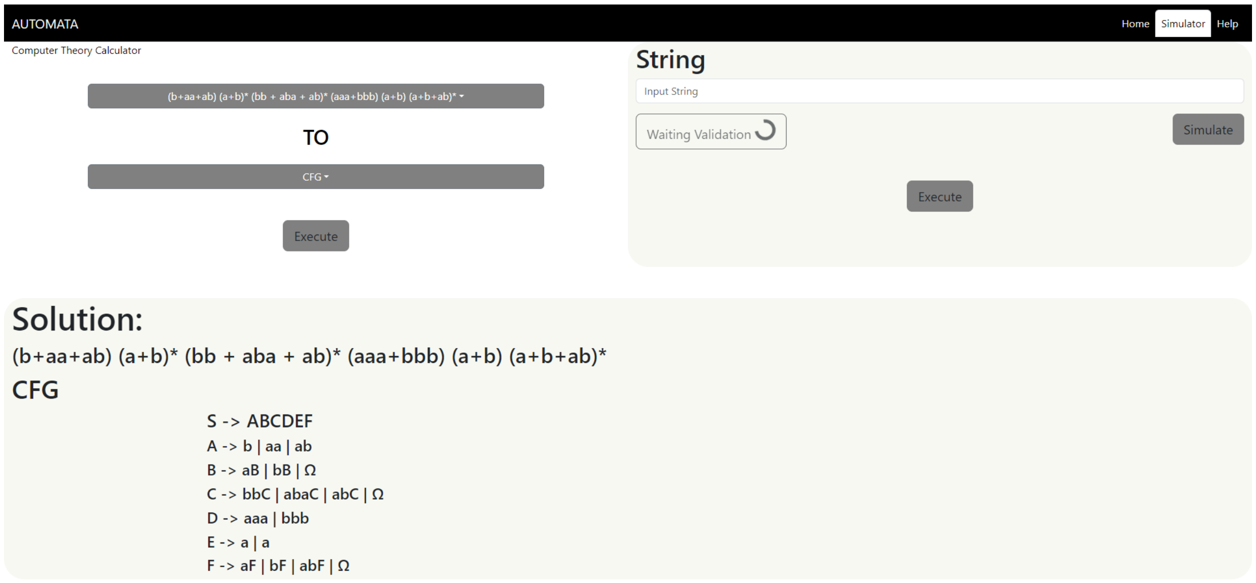


PDA

- Displaying the CFG

  For RegEx

  (b+aa+ab)(a+b)*(bb + aba + ab)*(aaa+bbb)(a+b)(a+b+ab)*



Solution:

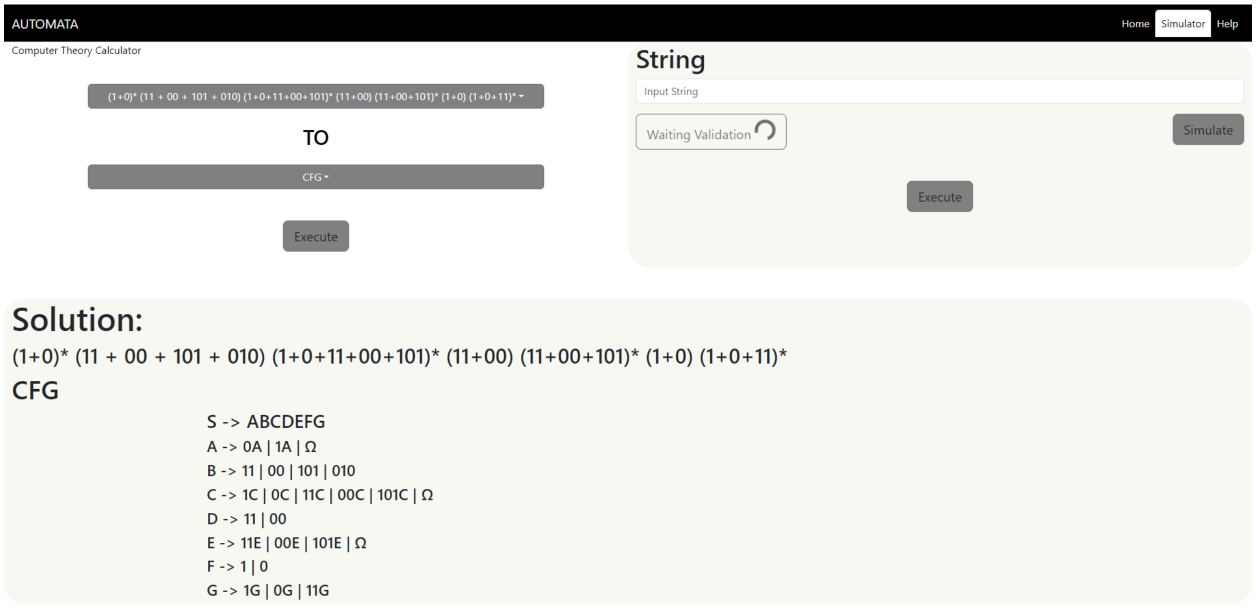
(b+aa+ab) (a+b)* (bb + aba + ab)* (aaa+bbb) (a+b) (a+b+ab)*

CFG

```
S -> ABCDEF
A -> b | aa | ab
B -> aB | bB | Ω
C -> bbC | abaC | abC | Ω
D -> aaa | bbb
E -> a | a
F -> aF | bF | abF | Ω
```

For RegEx

(1+0)*(11+00+101+010)(1+0+11+00+101)*(11+00)(11+00+101)*(1+0)(1+0+11)*



Solution:

(1+0)* (11 + 00 + 101 + 010) (1+0+11+00+101)* (11+00) (11+00+101)* (1+0) (1+0+11)*

CFG

```
S -> ABCDEFG
A -> 0A | 1A | Ω
B -> 11 | 00 | 101 | 010
C -> 1C | 0C | 11C | 00C | 101C | Ω
D -> 11 | 00
E -> 11E | 00E | 101E | Ω
F -> 1 | 0
G -> 1G | 0G | 11G
```