

Theory of Computation (TIC2151)

Trimester 2 2020/2021

Assignment

**Group G14(TT2V)**

|  |  |  |
| --- | --- | --- |
| Name | ID | Participation |
| HOSSAIN MOHAMMAD MUBDIUL | 1161303847 | 40% |
| HAIDAR ALI NASSER ALI | 1171302004 | 30% |
| IMRAN UZAIR | 1171102458 | 30% |

# Table of Contents

[**Table of Contents**](#_gpb7hffz2k5u) **2**

[**Introduction**](#_7p62oxez89d8) **3**

[**Design Flowcharts**](#_uh976omkhed) **4**

[Part 1 Convert Regular Grammar to NFA with Epsilon](#_cp591jeikkjb) 4

[Part 2 Convert Regular Grammar to NFA without Epsilon](#_vie7ktgg824a) 5

[Part 3 Convert Regular Grammar to DFA](#_mlds2kj65kfp) 6

[Part 4 Convert Regular Grammar to Minimized DFA](#_9ucao56ywavp) 7

[Part 5 Test Strings](#_m54zyljzkfp9) 8

[Figure 5 Flowchart for Testing String](#_dgcj2ocln8iq) 8

[**Algorithm**](#_tlhlpe8lfae7) **9**

[Part 1 Convert Regular Grammar to NFA with Epsilon](#_z8kh2tn9ddkb) 9

[Part 2 Convert Regular Grammar to NFA without Epsilon](#_njyat5i8ssyq) 10

[Part 3 Convert Regular Grammar to DFA](#_y3ec4gl3fhex) 11

[Part 4 Convert Regular Grammar to Minimized DFA](#_izkvprve1tw7) 12

[Part 5 Check Machine](#_3u0odb3asmfj) 13

[**Screenshots**](#_cdqvqsrvi6cv) **14**

[Home page](#_5leoytln4uq6) 14

[NFA](#_1cihm2i5n0x) 15

[NFA w/o ɛ](#_aul23mbe5noh) 16

[DFA](#_eug6cddv3ggu) 17

[Min DFA](#_vlbqm9wnsbj0) 18

[Test](#_k0k8aa9h2pxn) 19

[**Manual**](#_qfvfmmdbcc8s) **20**

[**Limitations**](#_ok5r8jtsw9wy) **21**

[**Important Codes**](#_9apx75hcsbt1) **22**

# **Introduction**

In this assignment we were tasked to make a program that would be able to do the following function:

1. Regular Grammar (RG) to eNFA
2. RG to NFA without epsilon
3. RG to DFA
4. RG to minimized DFA
5. Testing strings

# **Design Flowcharts**

## **Part 1 Convert Regular Grammar to NFA with Epsilon**

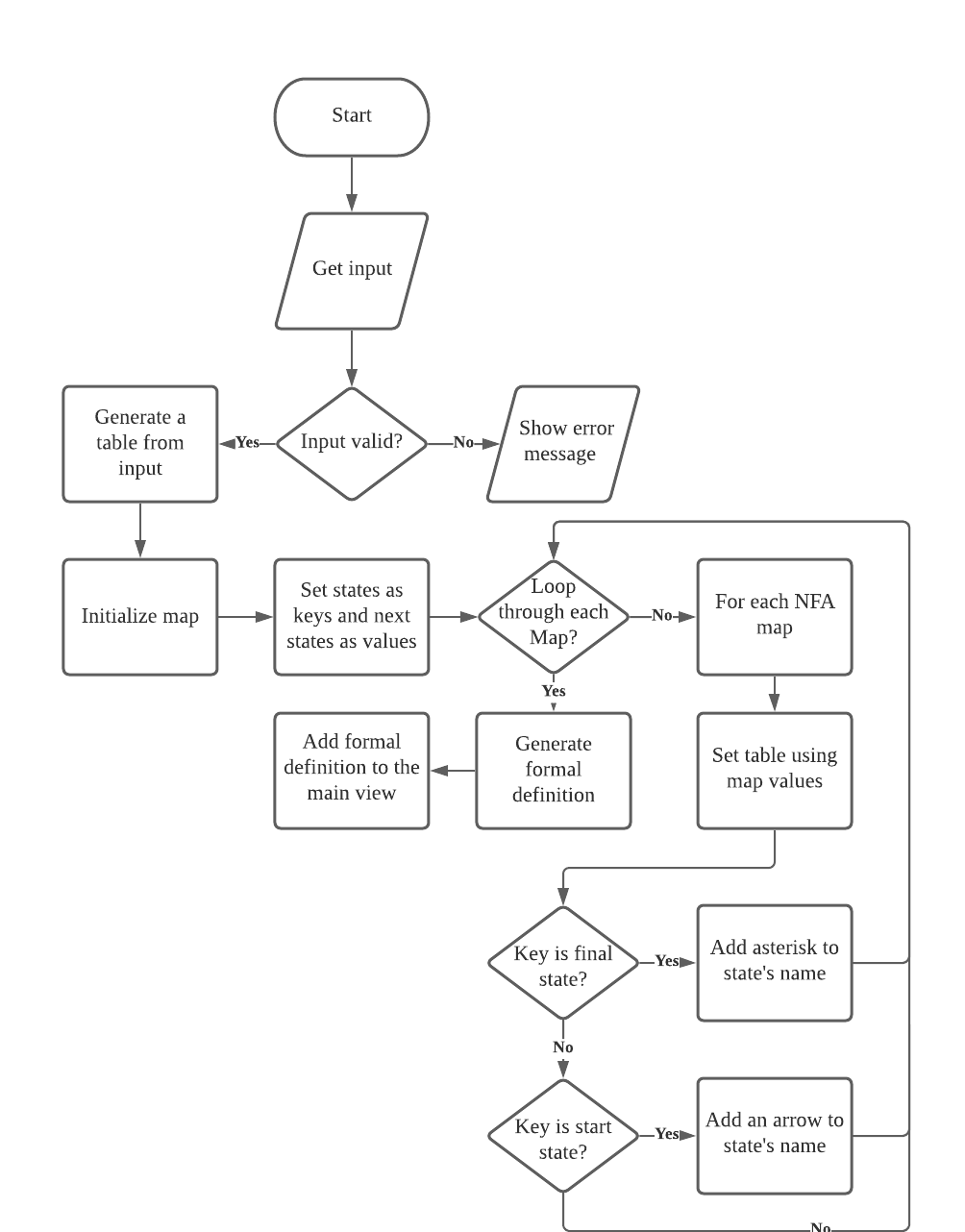


Figure 1 Flowchart for NFA with epsilon

## **Part 2 Convert Regular Grammar to NFA without Epsilon**

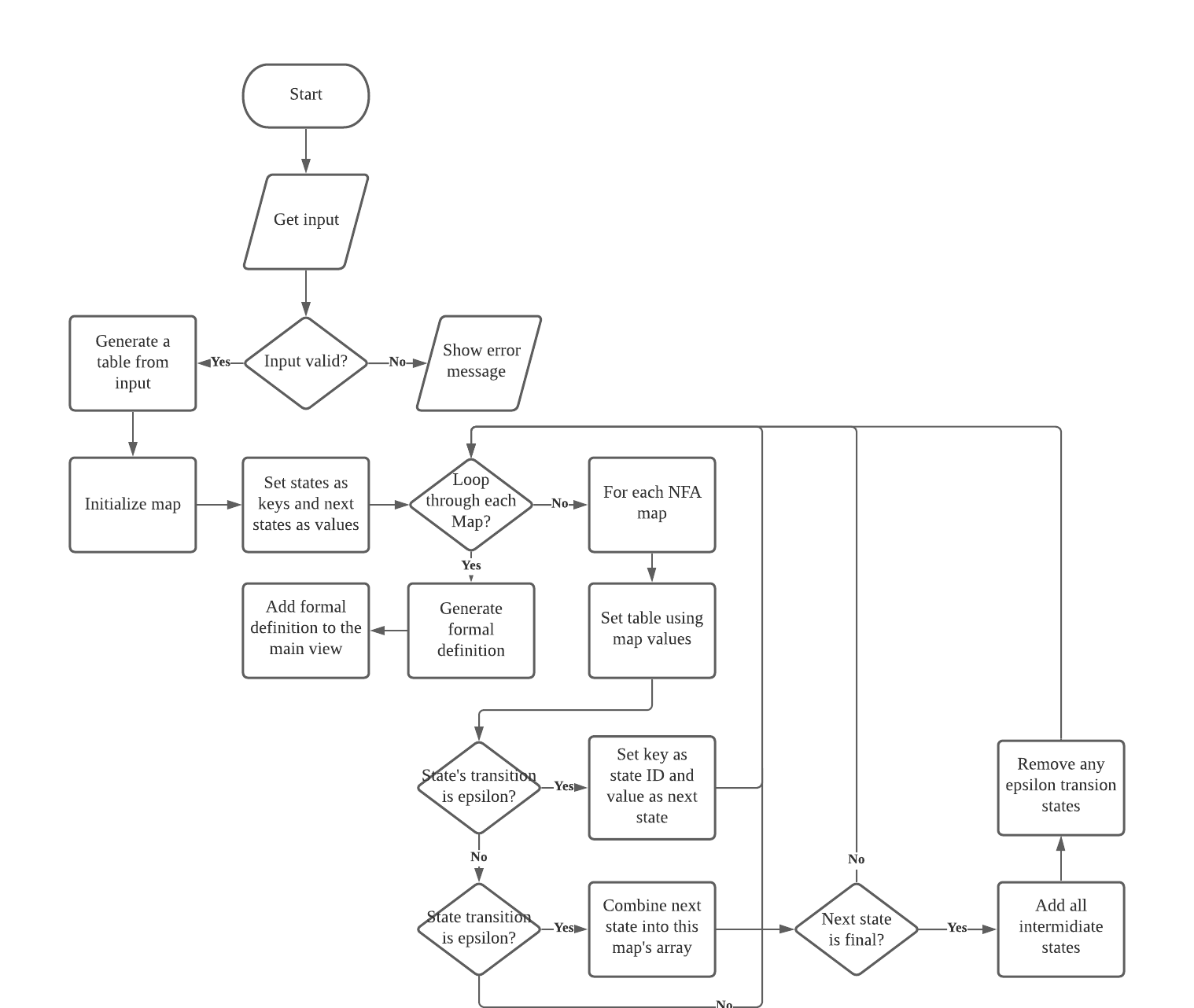


Figure 2 Flowchart for NFA without epsilon

## **Part 3 Convert Regular Grammar to DFA**

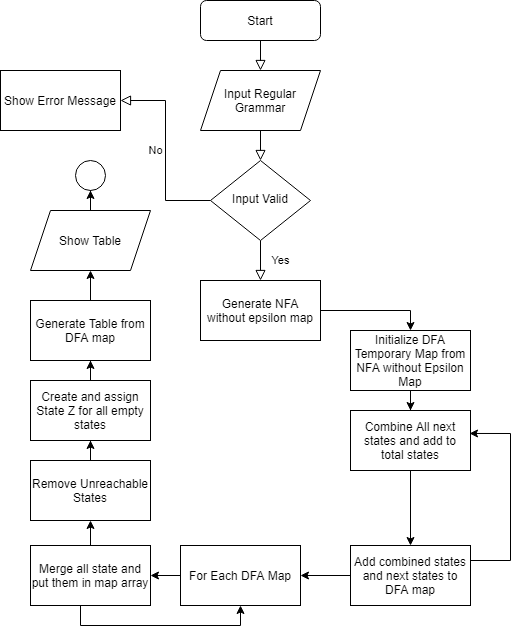


Figure 3 Flowchart for DFA

## **Part 4 Convert Regular Grammar to Minimized DFA**

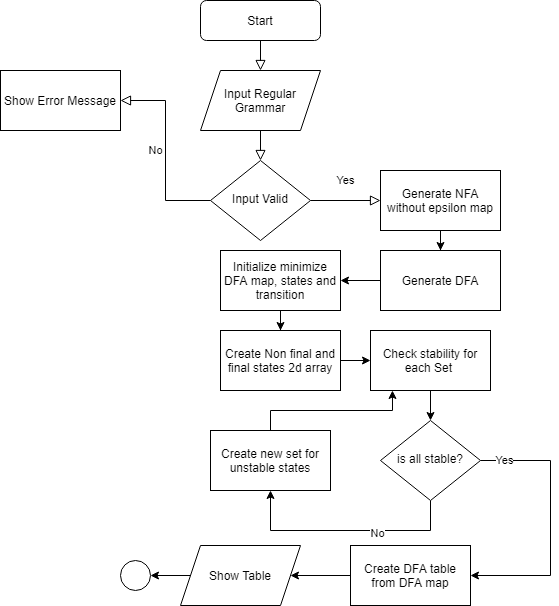


Figure 4 Flowchart for minimized DFA

## 

## Part 5 Test Strings

## 

## Figure 5 Flowchart for Testing String

# **Algorithm**

## **Part 1 Convert Regular Grammar to NFA with Epsilon**

READ Regular Grammar textarea or Load from file

Reset all variables

Split text from input field

Initialize regular grammar array by new line or space

Set each state and transition and next state to the State object list

Set total states, start states, final states and total transitions

IF format is invalid

show error message

ENDIF

IF format is valid

Generate a table layout from all the states and transitions

Initialize nfa with epsilon map

Set each states and transition as key and their value of all next states in array to the map

Assign key’s value to State object’s next State‘s value

FOR each nfa map

set each cell of the table from each map's value

IF the key of the map is final states

add an asterisk to the state's name

ENDIF

IF the key of the map is start state

add an arrow to the the states name

ENDIF

ENDFOR

Generate the formal definition layout

Add formal definition layout and table to the main view

ELSE

show error message

return

ENDIF

## 

## **Part 2 Convert Regular Grammar to NFA without Epsilon**

READ Regular Grammar textarea or Load from file

Reset all variables

Split text from input field and initialize regular grammar array by new line or space

Set each state and transition and next state to the State object list

Set total states, start states, final states and total transitions

IF format is invalid

show error message

ENDIF

IF format is valid

generate a table layout from all the states and transitions

create a new map for nfa without epsilon

IF each State's transition is not epsilon

set the map's key as the State's state id and transition and store the next state as value

ELSEIF a State's transition is epsilon

Look into the State objects and combine the next state in to this map's array

IF next state is final add all intermediate states

Remove any epsilon transition remaining

ENDIF

ENDIF

FOR each nfa without epsilon map

set table cell of each key to map's value

ENDFOR

Generate a formal definition layout for NFA without epsilon

Add formal definition layout and table to the main view

ELSE

show error message

return

ENDIF

## **Part 3 Convert Regular Grammar to DFA**

READ Regular Grammar textarea or Load from file

Reset all variables

Split text from input field and initialize regular grammar array by new line or space

Set each state and transition and next state to the State object list

Set total states, start states, final states and total transitions

IF format is invalid

show error message

ENDIF

IF format is valid

get nfa without epsilon map

create an empty dfa temporary map

copy nfa without epsilon map to dfa temporary map

FOR each dfa temporary map

FOR each transition

Combine all the next states and add to dfa total state list and to the map

ENDFOR

ENDFOR

FOR each dfa temporary map

Merge all the next state and put them in the array

ENDFOR

Create Z state to all empty states of DFA temporary map

Add Z to dfa total transitions

remove any unreachable state in dfa temporary map and assign them in Final DFA map

generate a table layout from all the states and transitions

Put all the states and transition from the map and their value to each cell

Add table to main view

ELSE

show error message

return

ENDIF

## **Part 4 Convert Regular Grammar to Minimized DFA**

READ Regular Grammar textarea or Load from file

Reset all variables

Split text from input field and initialize regular grammar array by new line or space

Set each state and transition and next state to the State object list

Set total states, start states, final states and total transitions

IF format is invalid

show error message

ENDIF

IF format is valid

get nfa without epsilon map

get DFA from nfa without epsilon map

Create an equivalence array and separate the total states based on final and non final the insert into the equivalence array

FOR each equivalence

check stability of each states from the DFA map

If all stable

update DFA map

BREAK

ELSE

Continue

ENDIF

ENDFOR

create table from the updated DFA map

FOR each DFA map

set table's each cell's value to DFA map's value

ENDFOR

add table to main view

ELSE

show error message

return

ENDIF

## 

## **Part 5 Check Machine**

READ Regular Grammar textarea or Load from file

Reset all variables

Split text from input field and initialize regular grammar array by new line or space

Set each state and transition and next state to the State object list

Set total states, start states, final states and total transitions

IF format is invalid

show error message

ENDIF

IF format is valid

get nfa without epsilon map

get DFA from nfa without epsilon map

get minimize DFA from DFA

create 5 test input field for machine input and 1 button

WHEN button pressed

IF test input field is not empty

Initialize current state from start state

FOR each Min DFA map

traverse through each statein minimize DFA.

current state = last traversed state

IF test input's last string is in final state

SHOW pass

ELSE

show failed

IF input is epsilon

IF current state is final state

show Pass

ELSE

show NO

ENDIF

ENDIF

ENDIF

ELSE

show error message

return

ENDIF

# 

# **Screenshots**

## Home page

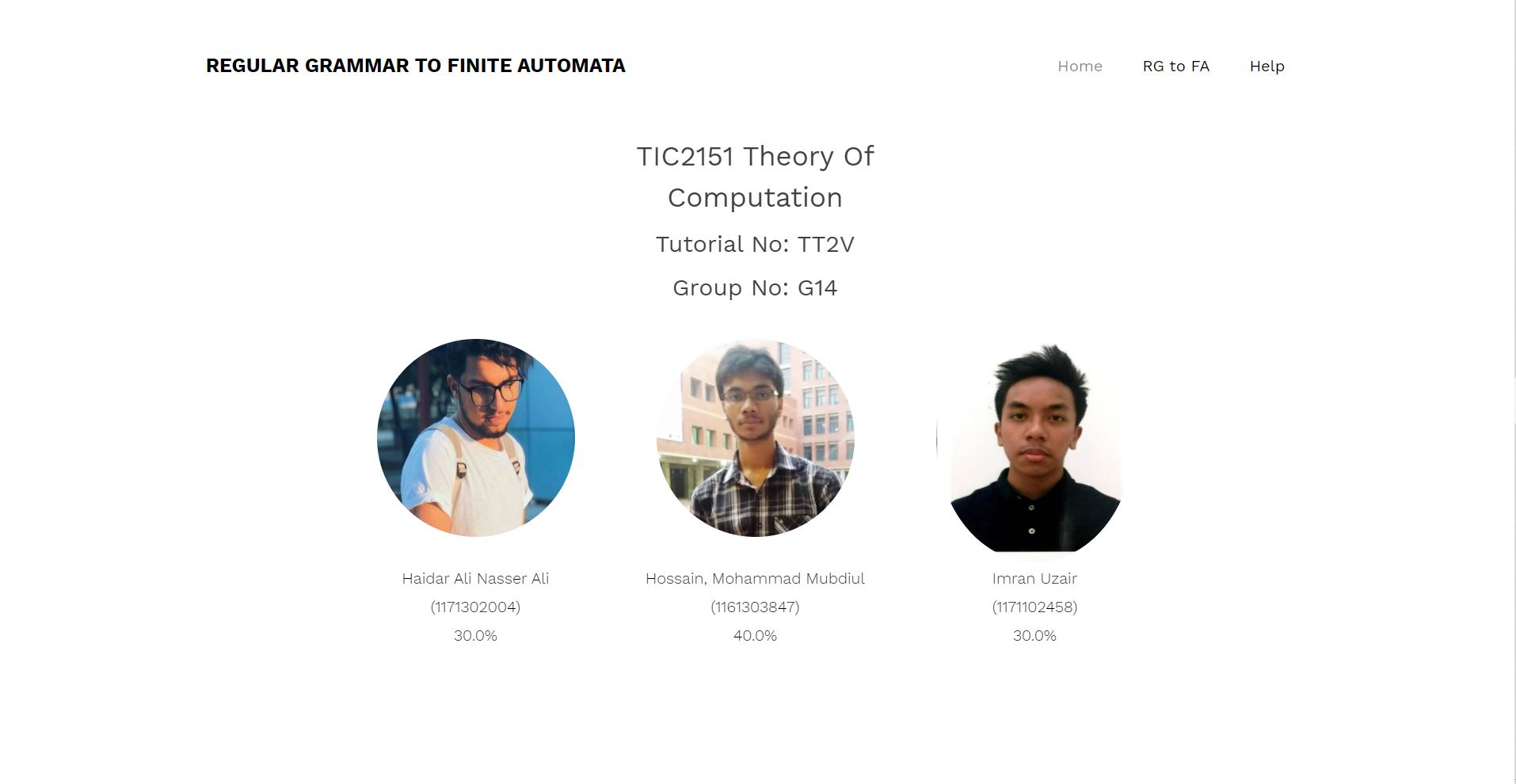


Figure 6 Home Page

# 

# 

## NFA

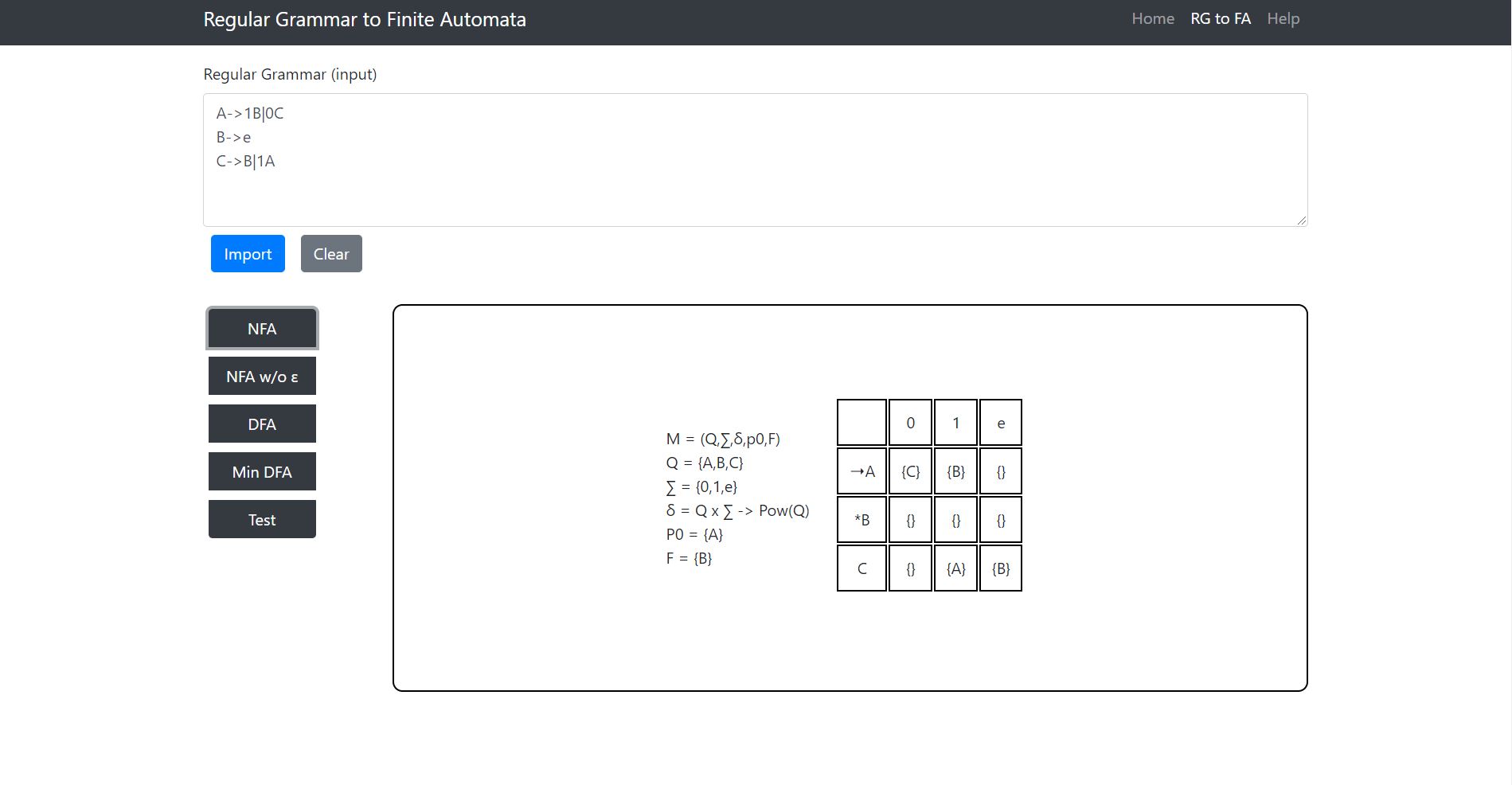


Figure 7 Regular Grammar to NFA with epsilon

## NFA w/o ɛ

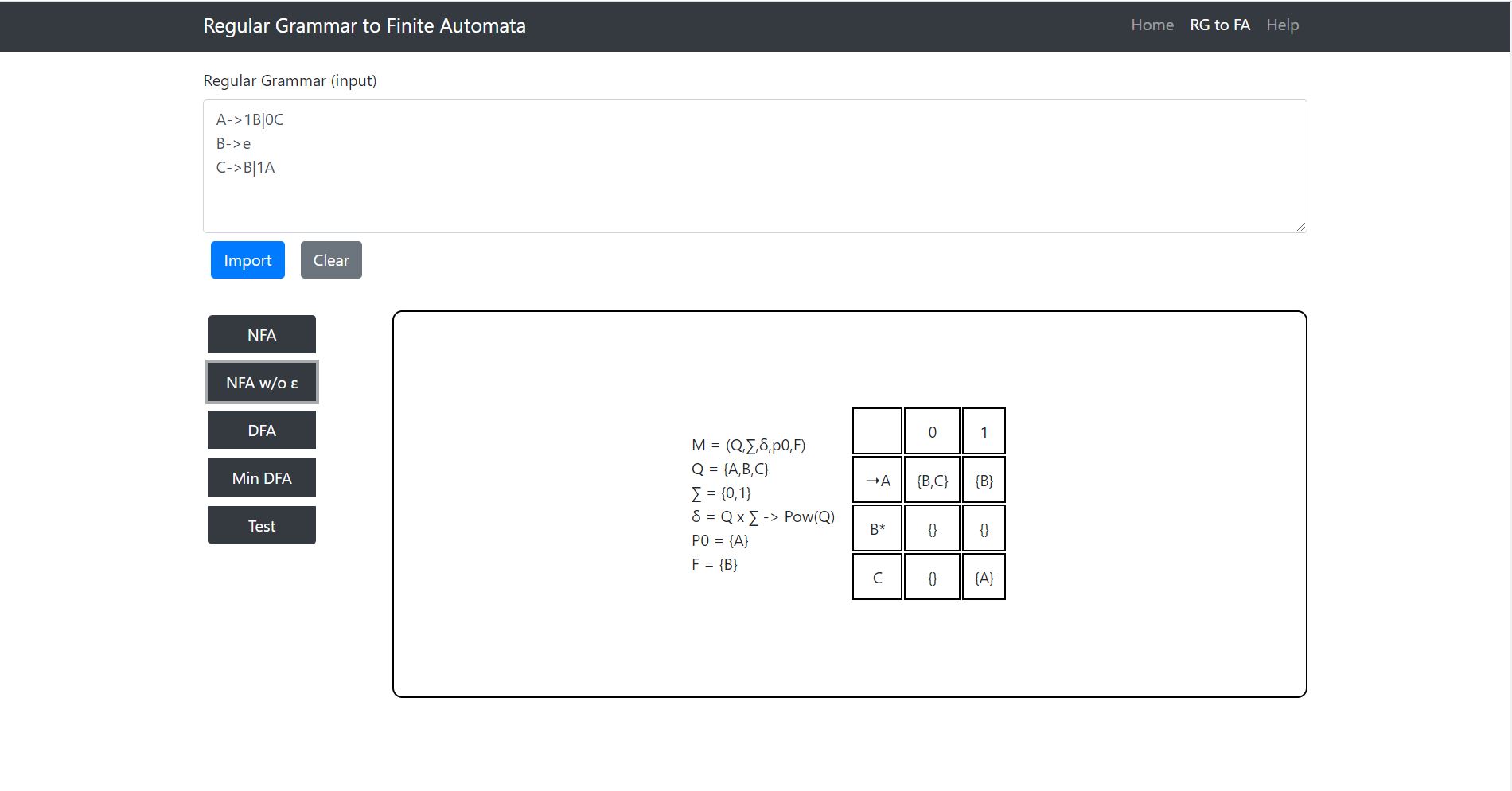


Figure 8 Regular Grammar to NFA without epsilon

## DFA

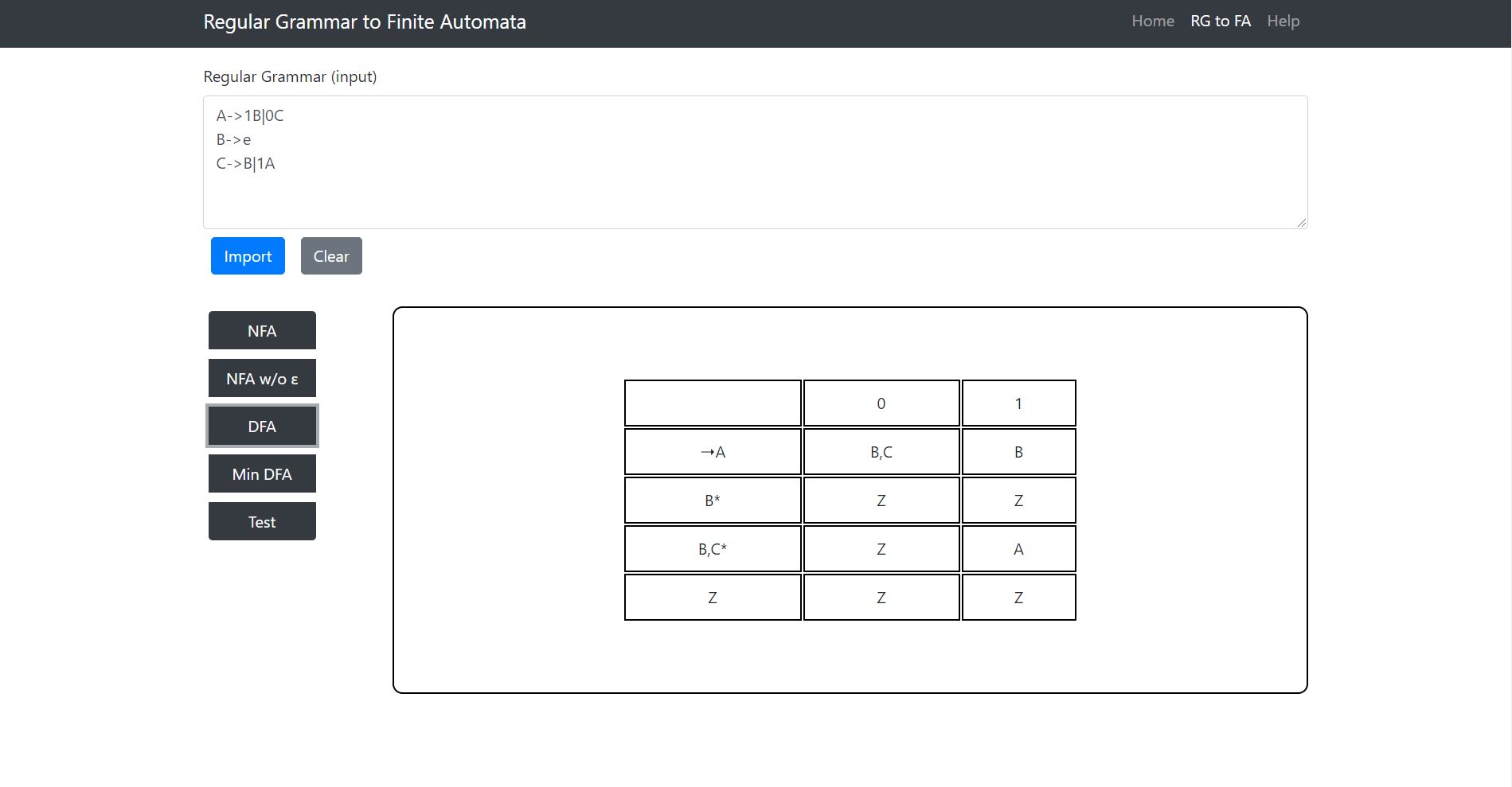


Figure 9 Regular Grammar to DFA

## Min DFA

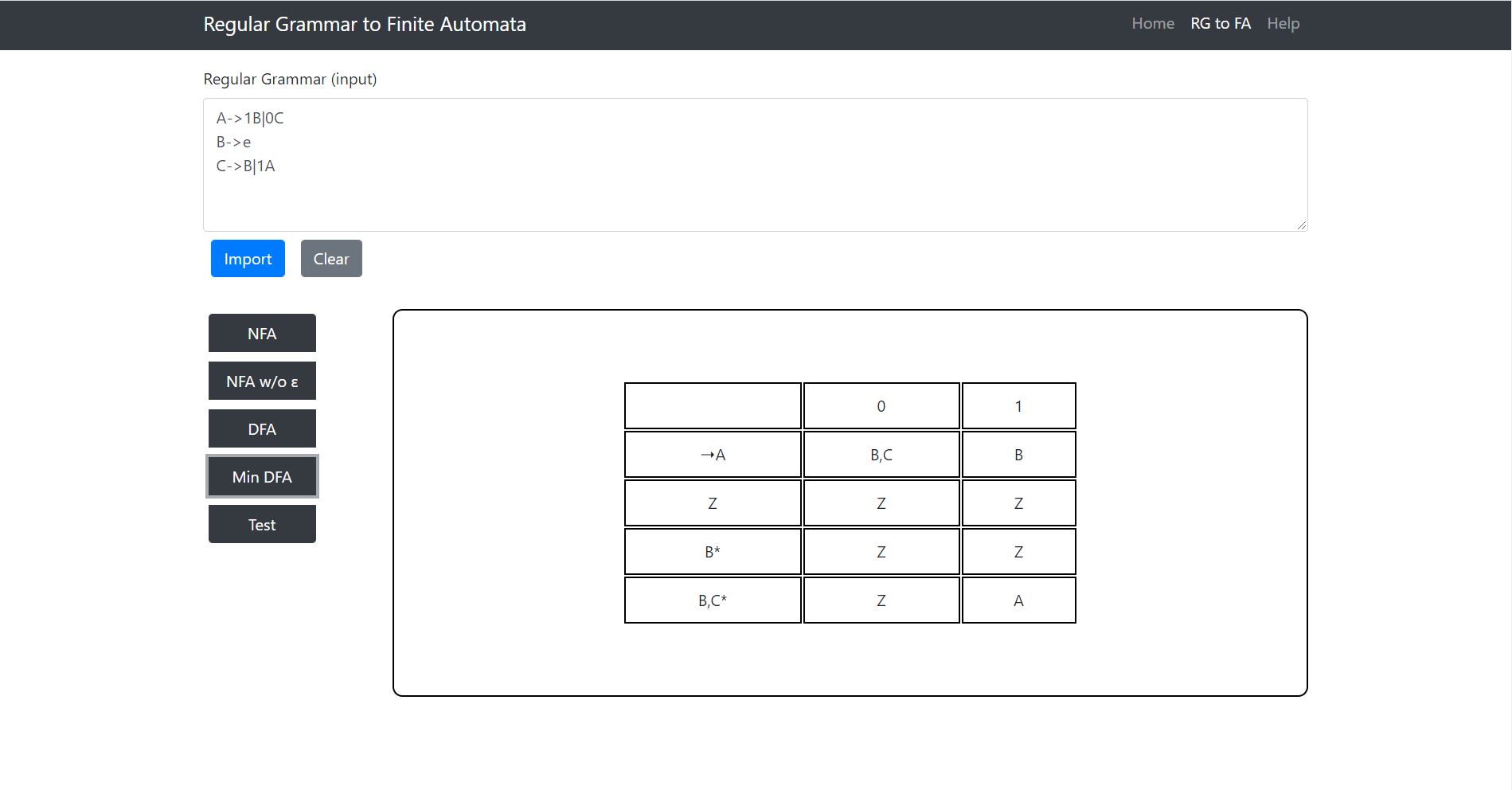


Figure 10 Regular Grammar to Minimized DFA

## Test

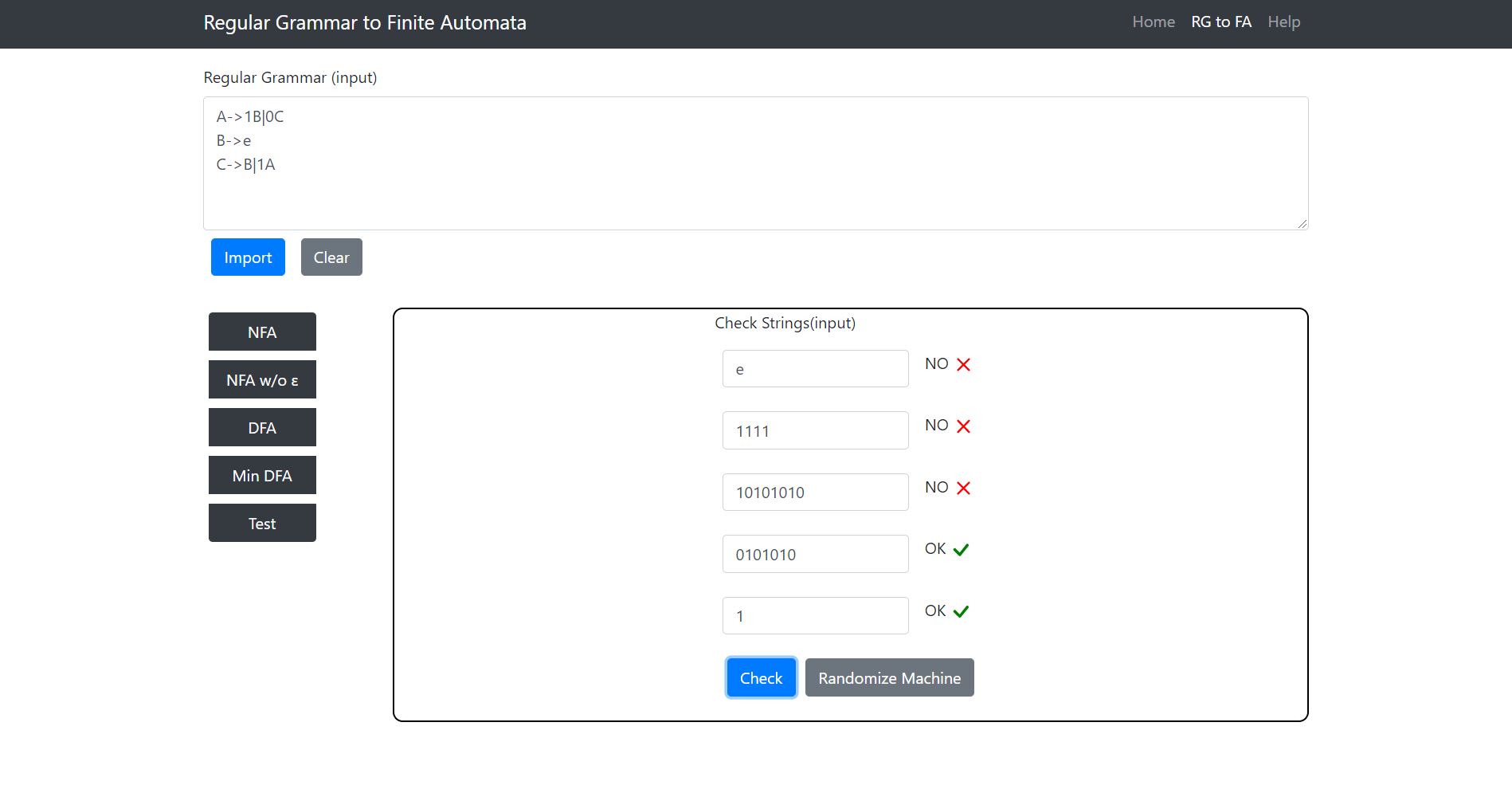


Figure 11 Testing 5 strings simultaneously

# 

# Manual

1) Use "e" to indicate "Ɛ" (epsilon).

2) Use enter or space to separate each Regular Grammar.

3) "->" is used to indicate "arrow".

4) The First RG is always the first state

5) After the arrow, each transition and their state is divided by "|". e.g., S->0A|0B

6) To enter an epsilon transition, just type the next state. e.g., S->A|B.

7) Press "e" for final state. e.g., S->1A|e.

8) To import a file, press the import button. Only text files and regular grammar files are supported.

9) Clear button will clear all the fields and views

10)The states named are merged for DFA and Minimized DFA due to clarification for new states' origin..

# Limitations

Due to time constraints the newly merged states were not renamed in DFA and Minimized DFA table. This doesn’t affect the output result.

# 

# 

# 

# 

# 

# 

# 

# Important Codes

Function to convert RG to NFA

|  |
| --- |
| function convertRGtoNFA() { // this function is called when NFA button is pressed.  convertRG();   if (!invalidFormat) {  let table = generateNFATable(); //generate an nfa table   var nfaMap = getNfaMap();   for (let [k, v] of nfaMap) { //k is state id+ transition, e.g. A\_b v is array of next states  var splitStateID = k.split("\_");   var stateIndex = totalStates.indexOf(splitStateID[0]);  var transitionIndex;  let isThisFinal = false;   if (splitStateID[1] != "epsln") {  if (splitStateID[1] == 'e') {  isThisFinal = true;  }  else {  transitionIndex = totalTransitions.indexOf(splitStateID[1]);  }  }  else {  transitionIndex = totalTransitions.indexOf('e');  }  if (!isThisFinal) {  let elementsState = "";  for (i = 0; i < v.length; i++) {  if (i == 0) {  elementsState += "{";  }  elementsState += v[i];   if (i != v.length - 1) {  elementsState += ",";  }  }  elementsState += "}";  table.rows[stateIndex + 1].cells[transitionIndex + 1].innerHTML = elementsState;  }  else {  table.rows[stateIndex + 1].cells[0].innerHTML = "\*" + splitStateID[0];  }  if (startState == splitStateID[0]) {  table.rows[stateIndex + 1].cells[0].innerHTML = "➝" + splitStateID[0];  if (isThisFinal) {  table.rows[stateIndex + 1].cells[0].innerHTML = "➝" + splitStateID[0] + "\*";  }  }  }   //todo: show all the starts states inside leftSideDiv  leftSideDiv.innerHTML = "M = (Q,∑,δ,p0,F)" + "<br>" + "Q = {" + totalStates.join() + "}" + "<br>" + "∑ = {" + totalTransitions.join() + "}" + "<br>" + "δ = Q x ∑ -> Pow(Q)" + "<br>" + "P0 = {" + startState + "}" + "<br>" + "F = {" + finalStates + "}";   //leftSideDiv.appendChild(table); // adding values to left side  rightSideDiv.appendChild(table); // adding table to right side   mainViewContainer.appendChild(leftSideDiv);  mainViewContainer.appendChild(rightSideDiv);   mainView.appendChild(mainViewContainer);   }  else {  mainView.innerHTML = "Please enter correct Regular expression!";  }  } |

Function to make the NFA without the ɛ

|  |
| --- |
| function nfaWOEP() { // we convert the NFA with epsilon to NFA without epsilon here.  convertRG();  if (!invalidFormat) {  let nfaWOepMap = nfaWOeMap(); // returning NFA without epsilon map.   var table = generateNFAWETable();   for (let [k, v] of nfaWOepMap) { //k is state id+ transition, e.g. A\_b v is array of next states  var splitStateID = k.split("\_");   var stateIndex = totalStates.indexOf(splitStateID[0]);  var transitionIndex = totalTransitions.indexOf(splitStateID[1]);   if (splitStateID[1] != "e") {  let elementsState = "";  for (i = 0; i < v.length; i++) {  if (i == 0) {  elementsState += "{";  }  elementsState += v[i];   if (i != v.length - 1) {  elementsState += ",";  }  }  elementsState += "}";  table.rows[stateIndex + 1].cells[transitionIndex + 1].innerHTML = elementsState;   }  }  for (let i = 0; i < totalStates.length; i++) {  if (finalStates.includes(totalStates[i])) {  table.rows[i + 1].cells[0].innerHTML += "\*";  }  if (startState == totalStates[i]) {  table.rows[i + 1].cells[0].innerHTML = "➝" + table.rows[i + 1].cells[0].innerHTML;  }  }  //todo: show all the starts states inside leftSideDiv  leftSideDiv.innerHTML = "M = (Q,∑,δ,p0,F)" + "<br>" + "Q = {" + totalStates.join() + "}" + "<br>" + "∑ = {" + totalTransitions.join() + "}" + "<br>" + "δ = Q x ∑ -> Pow(Q)" + "<br>" + "P0 = {" + startState + "}" + "<br>" + "F = {" + finalStates.join() + "}";  //leftSideDiv.appendChild(table); // adding values to left side  rightSideDiv.appendChild(table); // adding table to right side   mainViewContainer.appendChild(leftSideDiv);  mainViewContainer.appendChild(rightSideDiv);   mainView.appendChild(mainViewContainer);  }  else {  mainView.innerHTML = "Please enter correct Regular expression!";  } } |

Function from NFA to DFA

|  |
| --- |
| function nfaToDFA() // {  convertRG();   if (!invalidFormat) {  let finalDFA = dfaStates();  console.log(dfaFinalStates);  var table = generateDFATable(DFAStates, totalTransitions);   for (let [k, v] of finalDFA) {   let splitK = k.split("\_");   let indexOfState = DFAStates.indexOf(splitK[0]);   let indexOfTransition = totalTransitions.indexOf(splitK[1]);   let elementsState = v.join();   table.rows[indexOfState + 1].cells[indexOfTransition + 1].innerHTML = elementsState;   }  for (let i = 0; i < DFAStates.length; i++) {  if (dfaFinalStates.includes(DFAStates[i])) {  table.rows[i + 1].cells[0].innerHTML += "\*";  }  if (startState == DFAStates[i]) {  table.rows[i + 1].cells[0].innerHTML = "➝" + table.rows[i + 1].cells[0].innerHTML;  }  }    mainView.appendChild(table);  }  else {  mainView.innerHTML = "Please enter correct Regular expression!";  }   //console.log();  } |

Function to minimize the DFA

|  |
| --- |
| function minimizeDFA() { // when MinDFA button is pressed this function is called.   convertRG(); // we start from very beginning to convert the Regular expression to NFA->NFA without e->DFA.   if (!invalidFormat) {   let minimizedDFA = minimizeDFAAction(); // then we calculate minimization of the dfa   var table = generatedMinimizedDFATable()   for (let [k, v] of minimizedDFA) {   let splitK = k.split("\_");   let indexOfState = MinimizedDFAStates.indexOf(splitK[0]);   let indexOfTransition = totalTransitions.indexOf(splitK[1]);   table.rows[indexOfState + 1].cells[indexOfTransition + 1].innerHTML = v;   }  for (let i = 0; i < MinimizedDFAStates.length; i++) {  if (MinimizedDfaFinalStates.includes(MinimizedDFAStates[i])) {  table.rows[i + 1].cells[0].innerHTML += "\*";  }  if (MinimizedDfaStartState == MinimizedDFAStates[i]) {  table.rows[i + 1].cells[0].innerHTML = "➝" + table.rows[i + 1].cells[0].innerHTML;  }  }   mainView.appendChild(table);  }  else {  mainView.innerHTML = "Please enter correct Regular expression!";  } } |

Function to test the minimized DFA

|  |
| --- |
| function testMinimizedDFA() { // checkin input strings will be done here.    let minimizedDFA = minimizeDFAAction();   for (let i = 0; i < 5; i++) {  let inputCheckString = document.getElementById("checkStrings" + i).value;   let cleanString = inputCheckString.replace(/\s|\n/g, "");   let showResultDiv = document.getElementById('inputStringsImageDiv' + i);  showResultDiv.innerHTML = '';   if (cleanString.length != 0) {   let isAccepted = getTestResult(minimizedDFA, cleanString);  console.log(isAccepted);    let imageSource = document.createElement('img');  let textStatus = document.createElement('p');  imageSource.setAttribute('id','checkedStatus'+i);  imageSource.setAttribute('alt', "Checked");  imageSource.style.width = "30px";  imageSource.style.height = "30px";  //imageSource.style.float = "right";  if (isAccepted) {  imageSource.src = "images/check.svg";  textStatus.innerHTML = "OK";  }  else {  imageSource.src = "images/x.svg";  textStatus.innerHTML = "NO";  }  textStatus.appendChild(imageSource);  showResultDiv.style.display = "block";   showResultDiv.appendChild(textStatus);  //showResultDiv.appendChild(imageSource);  }  }  console.log(minimizedDFA); } |