Theory of Computation (TCS 3511)

Assignment

Faculty Of Computing and Informatics (FCI)

Multimedia University

Cyberjaya

23 February 2021

Member Participation

Student Id	Name	Percentage
1181302178	Lim Kee Hian	25%
1181302569	Muhammad Afham bin Md Awal-ludin	25%
1181301417	Lim Ji Jun	25%
1181302343	Lim Ee Tien	25%

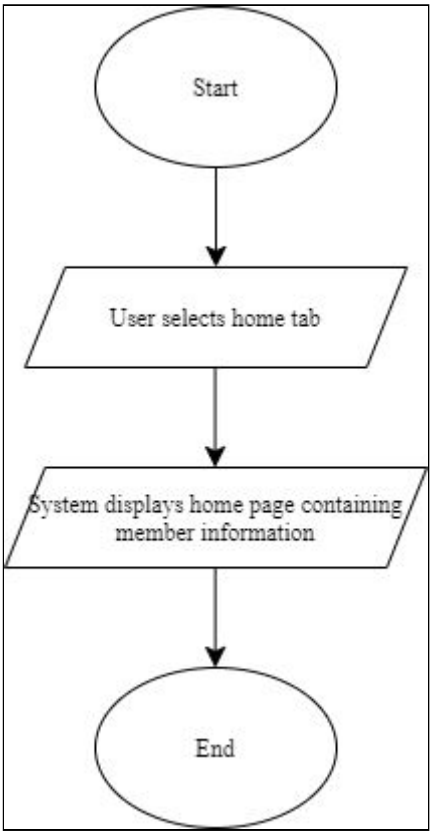
Introduction

We have made a simple application that converts regular grammar to finite state machine, which are NFA, NFA without e transition, DFA and minimized DFA. User needs to inputs regular grammar in the text area provided and the select intended conversion button like NFA, NFA without Ɛ, DFA, and minimized DFA.

User can also perform a simple test of the regular grammar that have been inputted earlier. In order to perform the test, user needs to select the test button and input any number of regular expressions. Then, the system will read and process the input and display the result at the right side of the text area. The result can be either “ok” to indicate that the test is passed successfully or “not ok” to indicate that the test has failed.

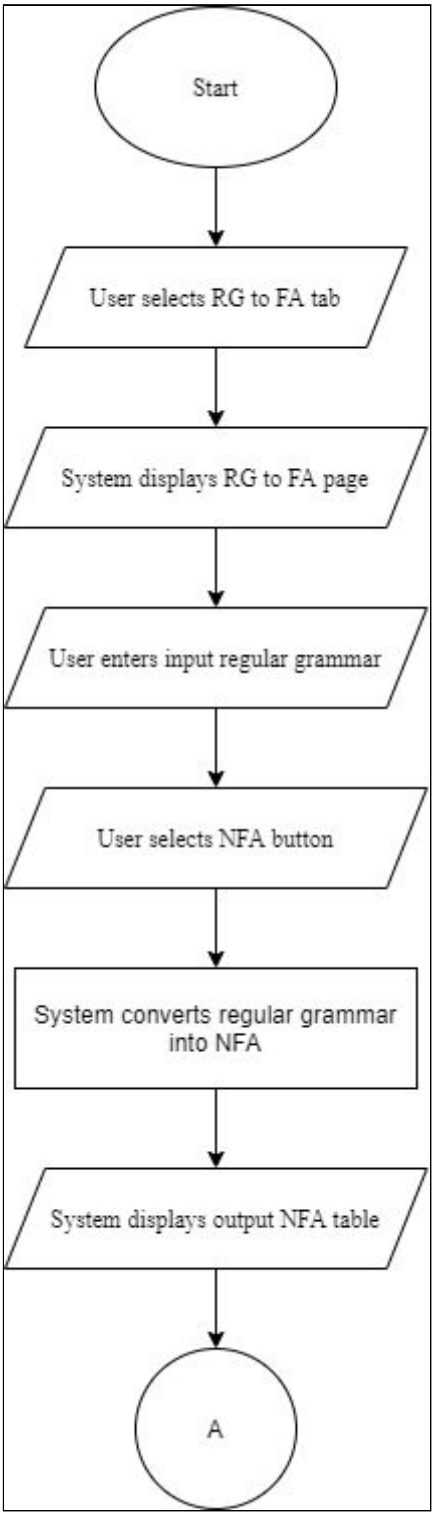
Design Flowchart

a. Home

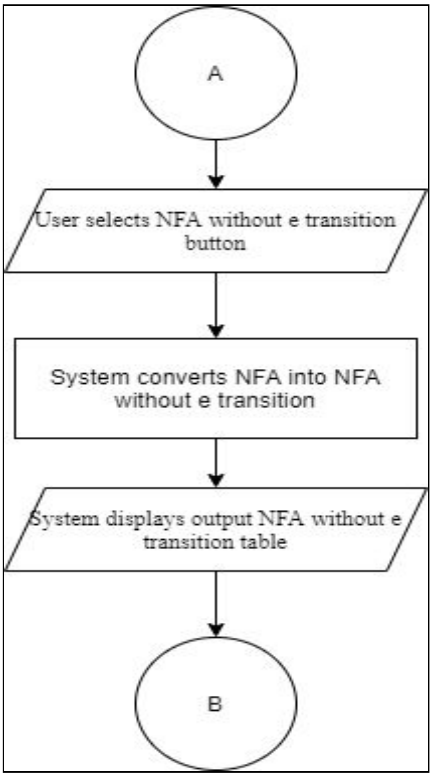


b. Main (RG To FA)

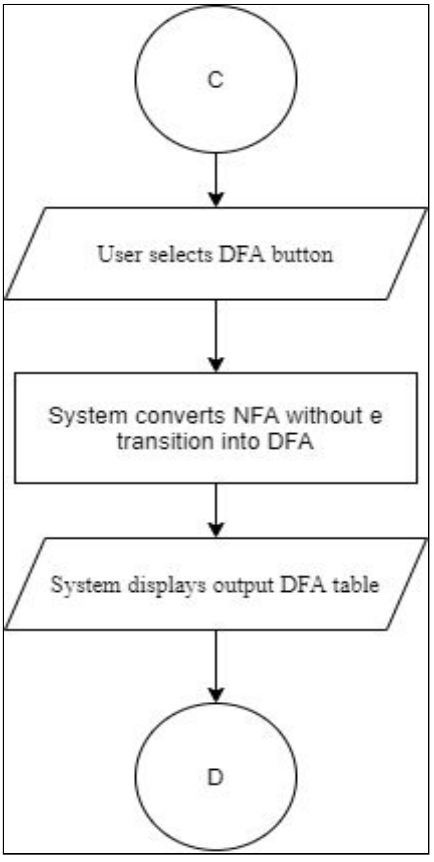
i. RG to NFA


Start
User selects RG to FA tab
System displays RG to FA page
User enters input regular grammar
User selects NFA button
System converts regular grammar into NFA
System displays output NFA table
A

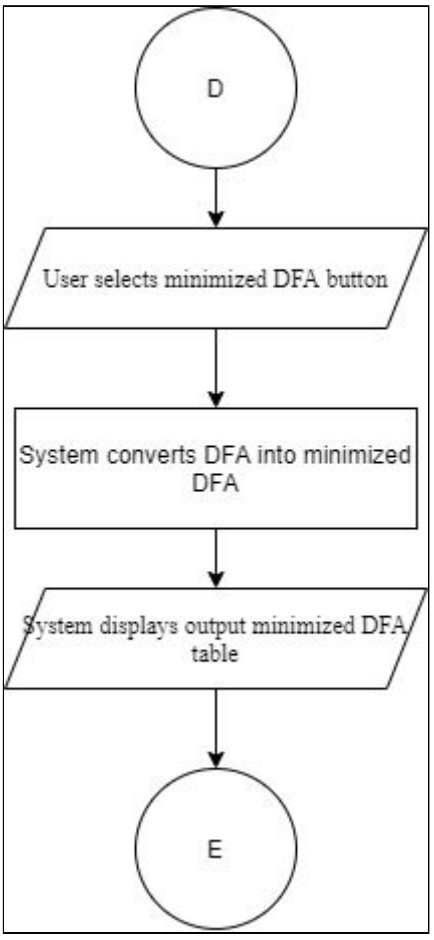
ii. NFA to NFA without e transition



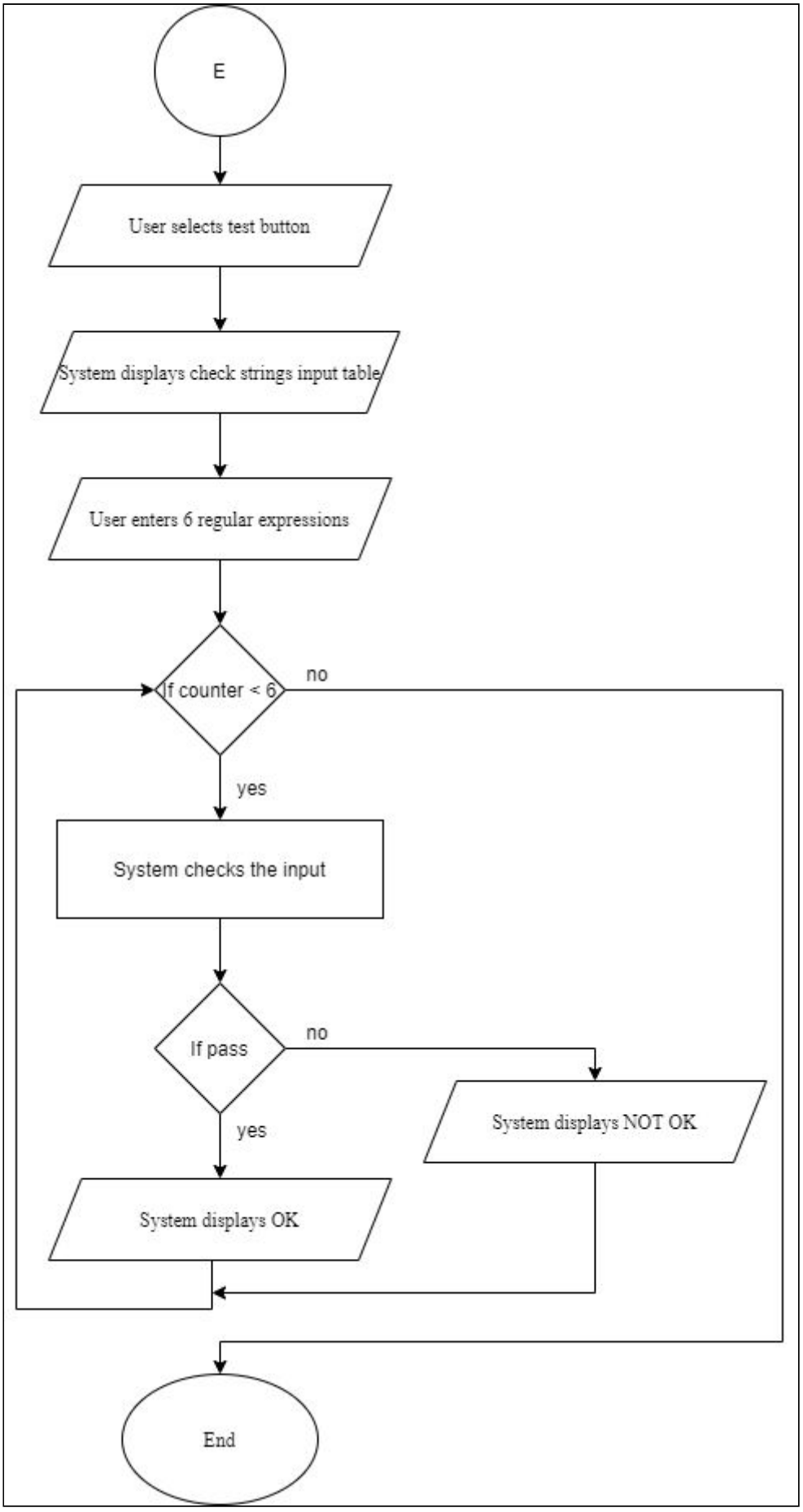
iii. NFA without e transition to DFA



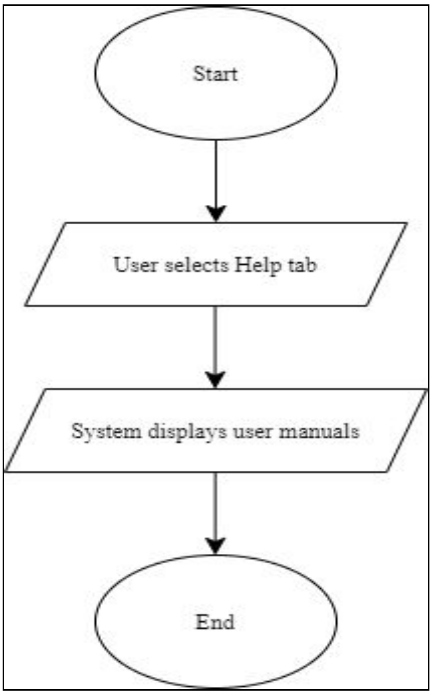
iv. DFA to minimized DFA



v. Test using regular expressions (up to 6 regular expressions)


E
User selects test button
System displays check strings input table
User enters 6 regular expressions
If counter < 6
no
yes
System checks the input
If pass
no
yes
System displays NOT OK
System displays OK
End

c. Help page


Start
User selects Help tab
System displays user manuals
End

Algorithms

Algorithms to convert to NFA

- In function nfa2(), get input from myTextArea at line 93.
- Pass the input to parseGrammar(). This function is to process the inputs of regular grammar and returns the results at line 96.
- Create an object Regular at line 130.
- From line 135 until 188, create objects Production and set or push each of them into productions[] property of the Regular object.
- Create an object nfa from line 216 until 247. The object nfa will have states as the keys. Values are represented by symbols mapped to the next or target state.
- Get all the states by calling Object.keys(nfa) at line 250 and get all values at by calling Object.values(nfa) at line 256.
- Next, from line 261 until 274, process the values to get a set of unique symbols.
- Print the symbol as the first row at line 281.
- Lastly, print all the respective states which match the symbols from line 302 until 399.

Algorithms to convert to NFA without e transition

- In function eClosure() from file nfawe.js, get the productions by calling regular.getProductions() and get all the symbols by calling regular.getNonTerminals() at line 8 and 9.
- From line 20 until 39, process the nfa to get a path with e transitions.
- Create three arrays called state, state_closure and nfa_closure_transitions.
- At line 50, call fetch_E_Closure() function to return paths that have e transitions. The result is pushed into state_closure. fetch_E_Closure() method can be found between line 295 until 338.
- After that, from line 57 until 61, called findNextStates() function to return all the next states that have e closure. The result will be assigned into next_states.
- Next, get all symbol of the next states between line 62 until 73.
- At the end, create a new Production object called pr and set its properties by calling pr.setProperties() and send state, symbol, and symbol_next_state as the arguments.
- Create a new nfawe2 object at line 116. Between line 117 until 144, push the nonTerminal into var state, symbol into var path, and nextState into rState[].

- Then, use state, path, and rState to initialize nfawe2 by assigning rState into nfawe[state][path]. Now, object nfawe2 has no e transitions.
- Get all state of nfawe2 by calling Object.keys(nfawe2) and assign into state at line 147 and get all transitions of nfawe2 by calling Object.values(nfawe2) and assign into tr1.
- Lastly, between 164 to 279, print all the symbols and the states into the right rows and columns using similar table format as in NFA conversion.

Algorithms to convert to DFA

- At line 106 in main.html, call dfa() method from dfa.js file when user click test button.
- Within dfa(), from line 3 to 5, create empty arrays dfa_states, dfa_final_states, and dfa_transitions.
- Then create an empty array called stack and initialize with start variable by calling regular.getStartVar() at line 14.
- By looping stack as long as it is larger than 0 from line 16, pop stack and store the value into object called state at line 17. Then check whether state is multi transition state. If true, then separate the state . Else, push state into new variable called states.
- Between line 35 until 42, loop the state to find the next or target states and store into next_states_union.
- Next, at line 45, get the a union from next_states_union by passing it to combineState and assign the value returned into combinedStatesUnion.
- Then, at line 48, check whether the combinedStateUnion is not null. If true, then push the combinedStateUnion into the stack. Else, push it into DFA_state.
- Next, create new arrays called state2 and term at line 104 and 105. Then assign state2 with current states while term is assigned with symbols.
- Create an object called dfa at line 119. Initialize it by assigning rState into dfa[state4][path2].
- Get all state of dfa by calling Object.keys(dfa) and assign into state5 at line 161 and get all transitions of dfa by calling Object.values(nfawe2) and assign into tr1.
- Lastly, between 166 to 273, print all the symbols and the states into the right rows and columns using similar table format as in NFA without e conversion.

Algorithms to convert to minimized DFA

- At line 107 in main.html, call Mdfa() method from mdfa.js file when user click test button.
- Within mdfa(), from line 2 to 4, create new arrays called tmptst, tmpfin, and tmpterm.
- Then, using those empty arrays, create 3 new sets called states, finalStates, and sym at line 10 until 12.
- Between line 16 and 101, loop the states.
- Inside the loop, check whether the final states include the current state. If yes, then get the next state.
- At line 107 and 108, create new arrays called state2 and term. Assign state2 with the current states and assign term with symbols.
- Create an object called mdfa at line 122. Initialize it by assigning rState into mdfa[state4][path2].
- Get all state of dfa by calling Object.keys(mdfa) and assign into state5 at line 164 and get all transitions of dfa by calling Object.values(mdfa) and assign into tr1.
- Lastly, between 169 to 278, print all the symbols and the states into the right rows and columns using similar table format as in DFA conversion.

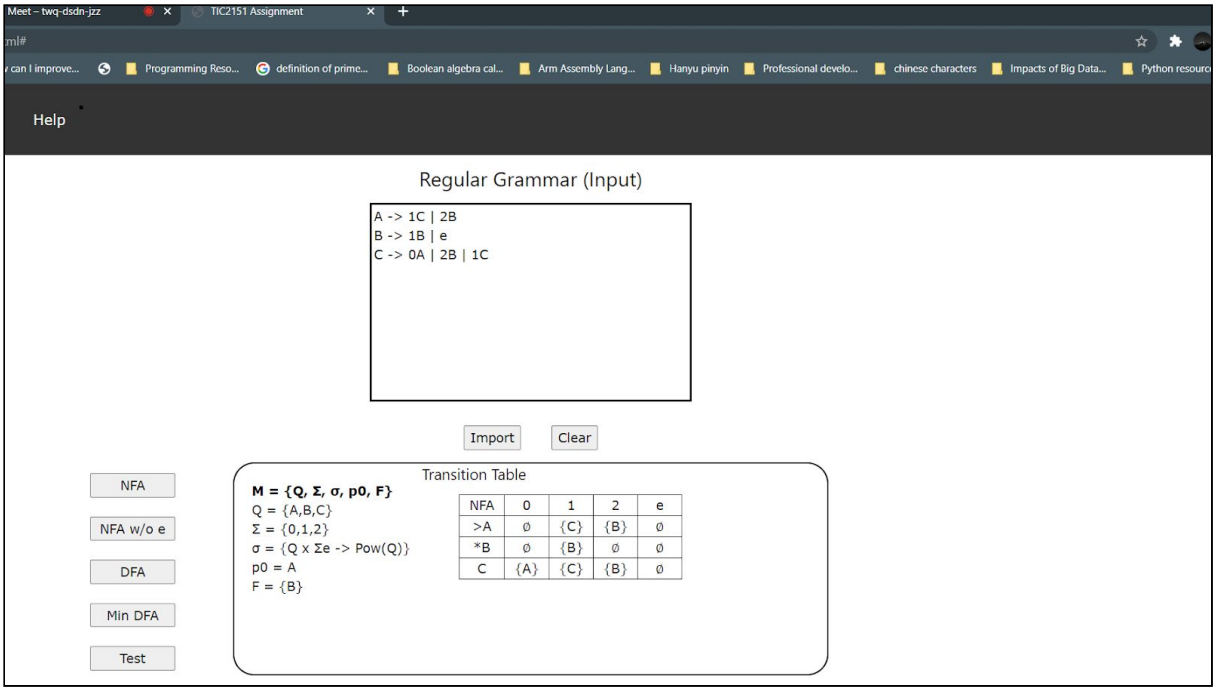
Algorithms to test the string

- At line 108 in main.html, call test() method from test.js file when user click test button.
- Within test(), create a container to receive user input of strings. Then call check() from check2.js.
- Get the input and split it by calling input.split(). Store the result into arr.
- At line 5, get all productions by calling regular.getProductions() and assign into production. Then, at line 6, get all final states by calling regular.getFinalVar() and assign into var finalStates.
- At line 11, create an empty symbolList[] and initialize it by calling production[].getSymbol() between line 13 until 16.
- Next, at line 19, get start state by calling regular.getStartVar() and assign into currentState. After that, at line 20, create an empty array results. This array will store an array of ‘ok’ and ‘not ok’ after checking each input in a loop.
- Begin by looping each input string at line 21, create another loop for each arr[index] to access every single digit. For example, access 0 for input begin with 0110101.

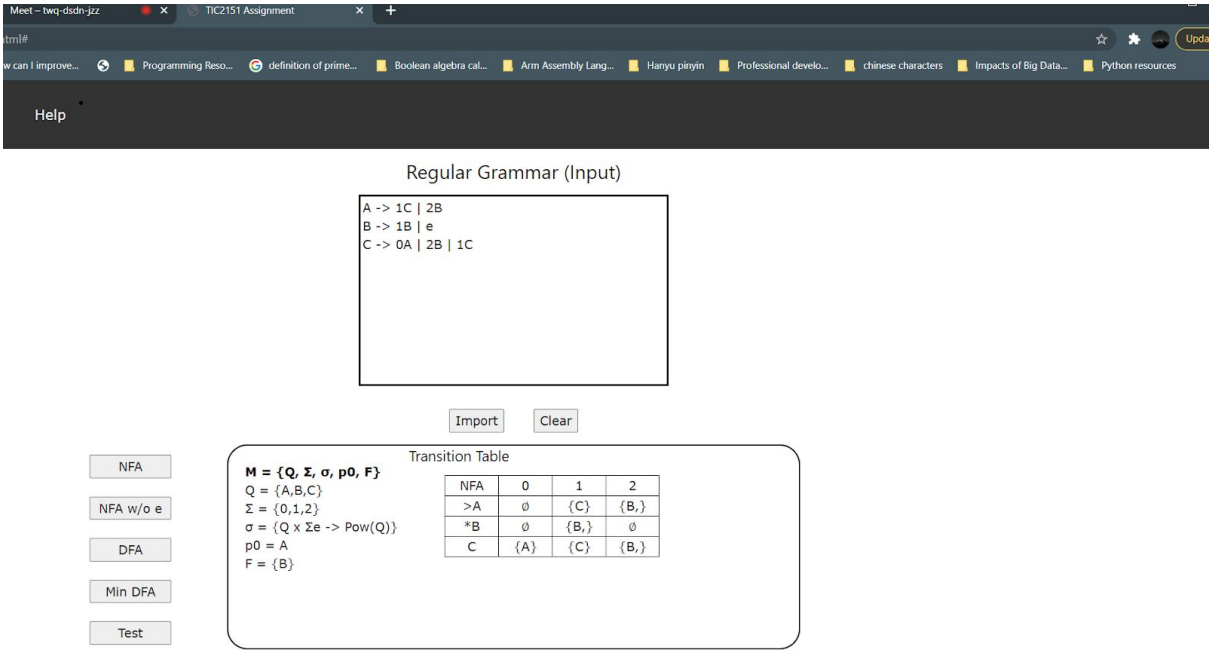
- Inside the loop, create another loop that iterates through the production at line 25.
- At line 28, check if currentState equal to production[k].getCurrentState().
- If true, then check if the current input digit equal to the production[k].getSymbol(). If true then assign production[k].getNextState() into currentState and set inputAndCurrentIsChanged = true
- Else, if the production.getSymbol() equal to ‘e’ at line 40, then check the following:
- if input reach the end and the current state is final state, then the current state is final state.
- Else if input haven’t reach the end and the current state is not final state, then go back to first production because already change state.
- Else if input haven’t reach the end and the current state is final state, then assign j to previous input index.
- Else if input reach the end and the current state is not final state, then go back to first production because already change state.
- At line 78 and 79, check whether inputAndCurrentIsChanged is to true or false. If false, then push ‘not ok’ to result. Else, push ‘ok’.

Screenshots

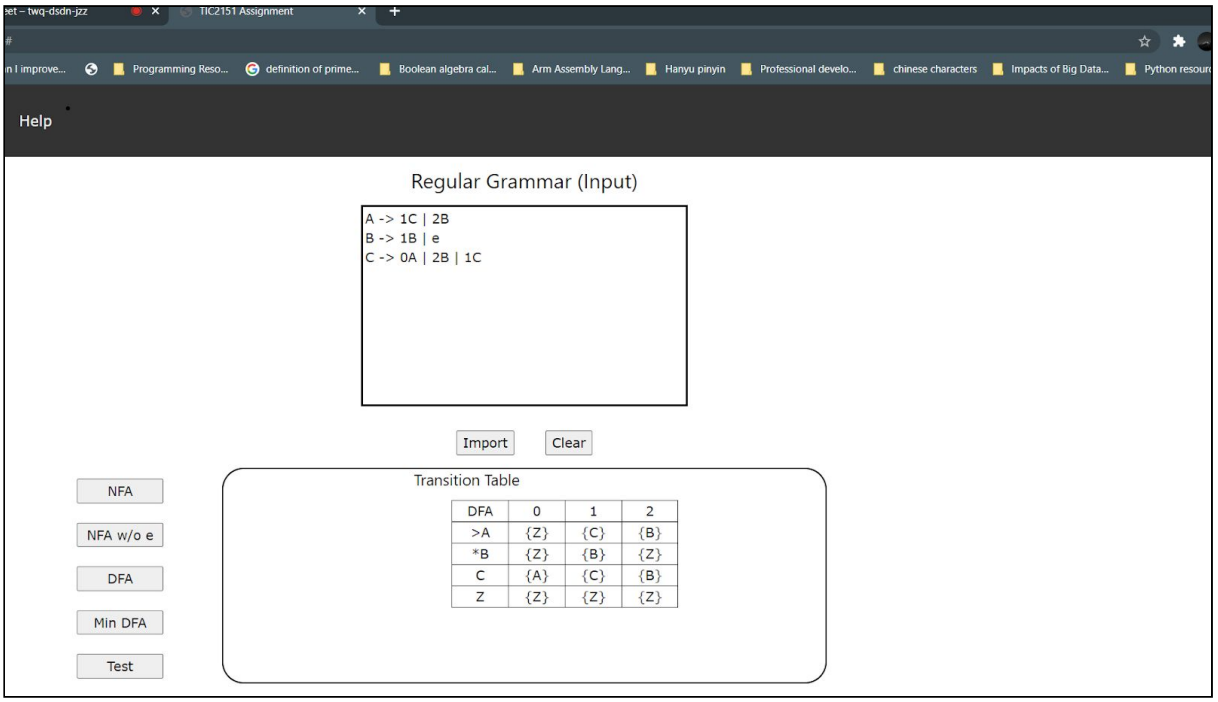
1. Example of output NFA table



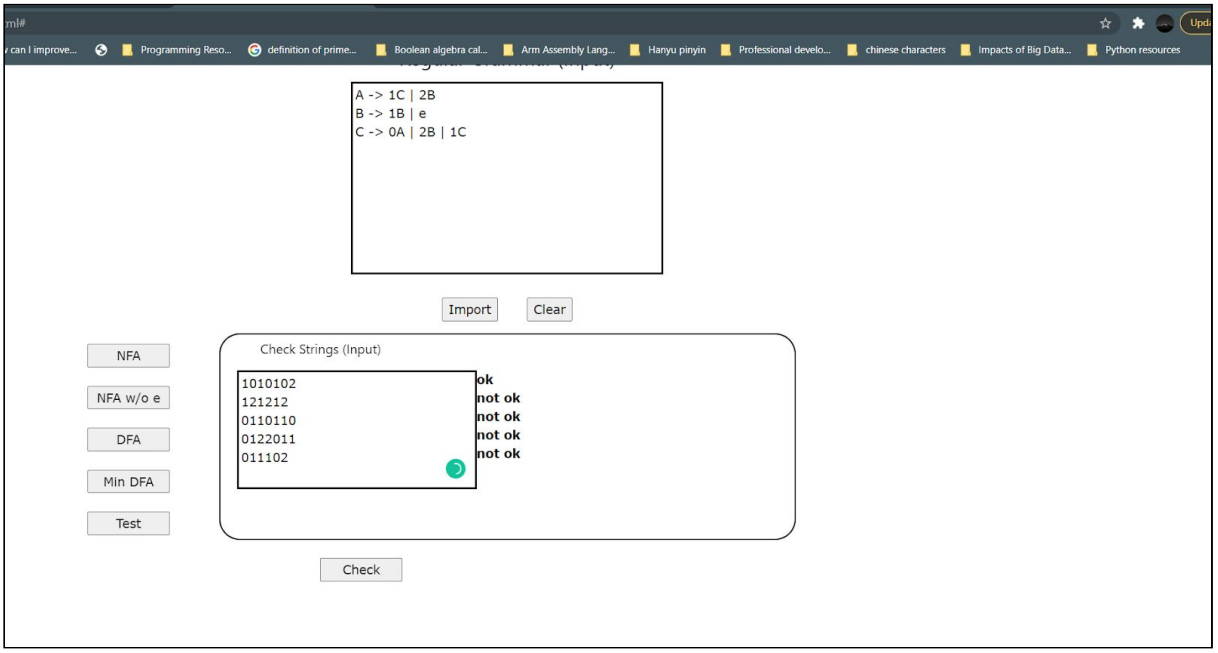
2. Example of output NFA without e transition table



3. Example of output DFA



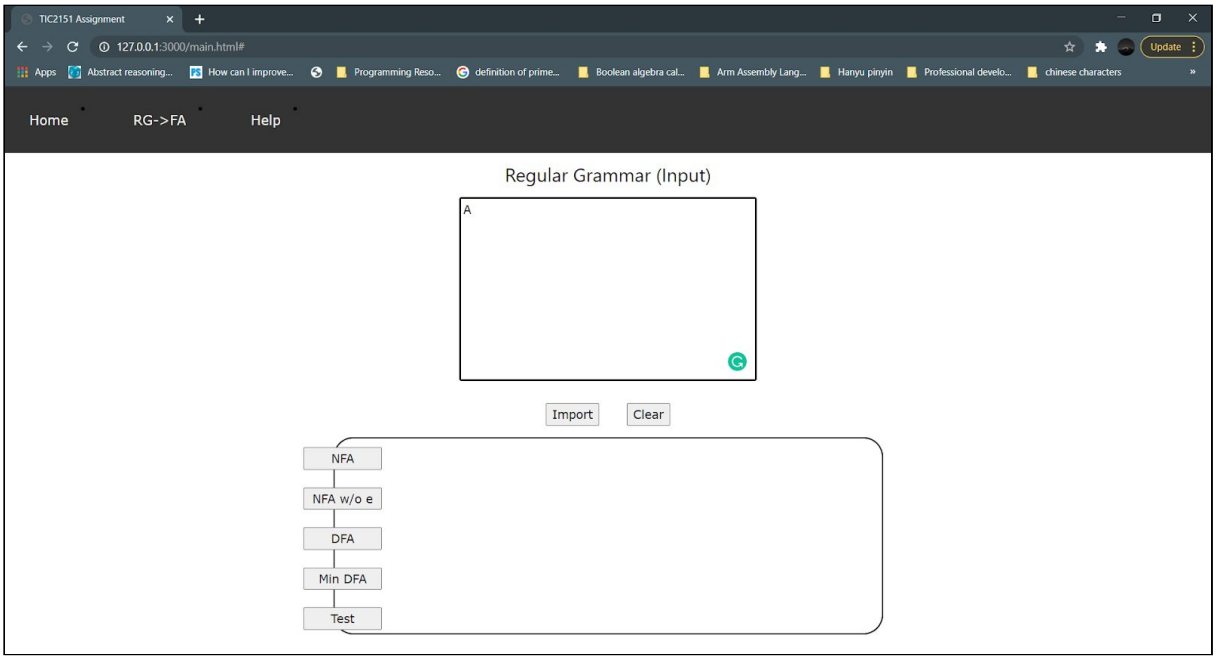
4. Example of output test strings



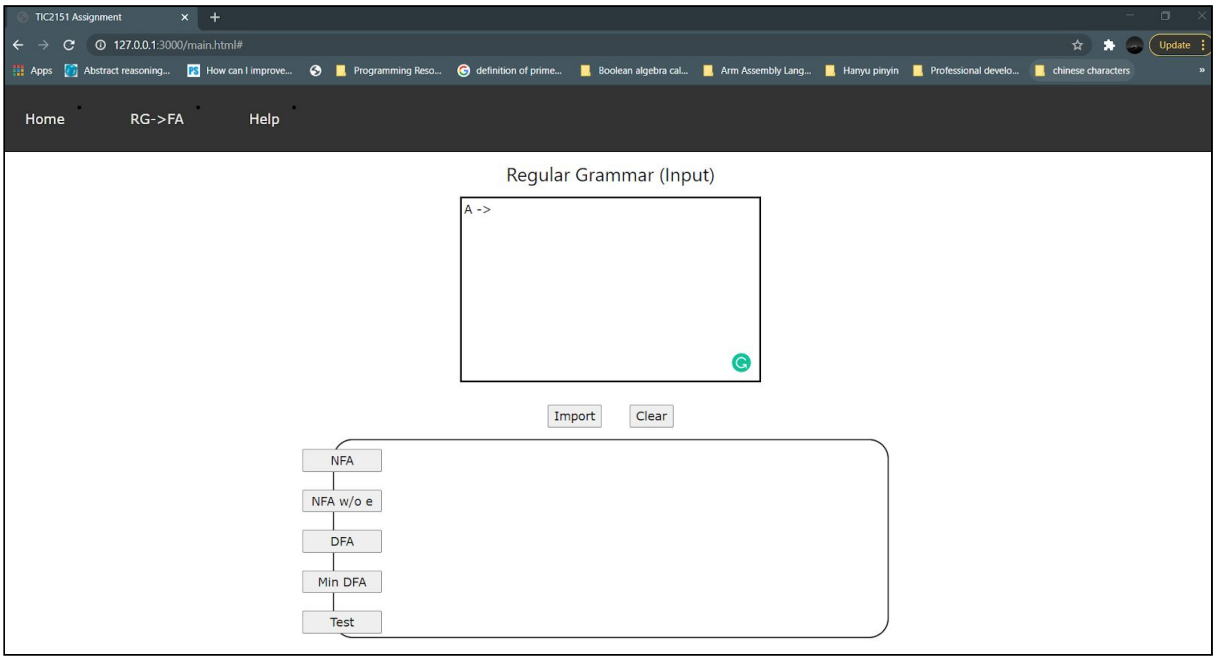
Manual with Examples

Following are step by step instructions on how to run the program.

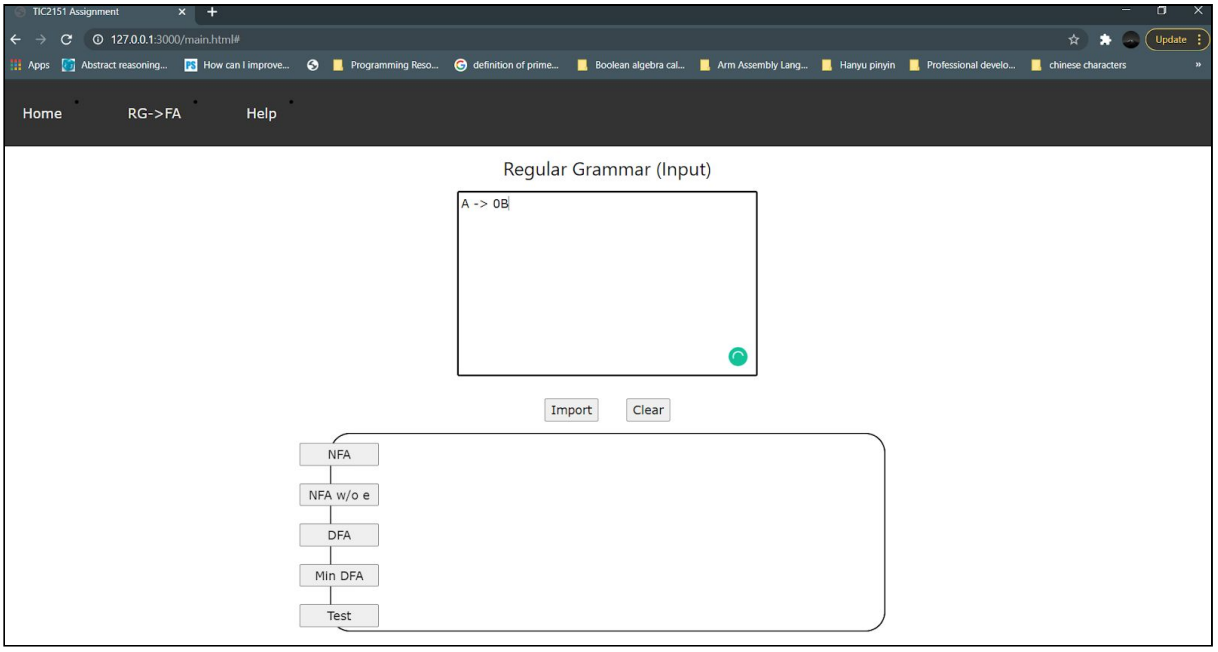
1. Select RG->FA tab from the top navigation and RG to FA page will appear.
2. There are few lines of regular grammar inputs that are automatically written in the text area below the Regular Grammar (Input). You can choose either to use the given inputs or type down your input in the text area.
3. When entering your own inputs, there are several rules that need to be followed. First, you need to make sure that all non terminals or states need to be in uppercase letters from **A** to **Z**. Below is an example of how to write a state A.



4. The program will automatically read the state A above as the start or initial state.
5. Second, to represent state A transit to another state, you must write characters '->' as shown in figure below.

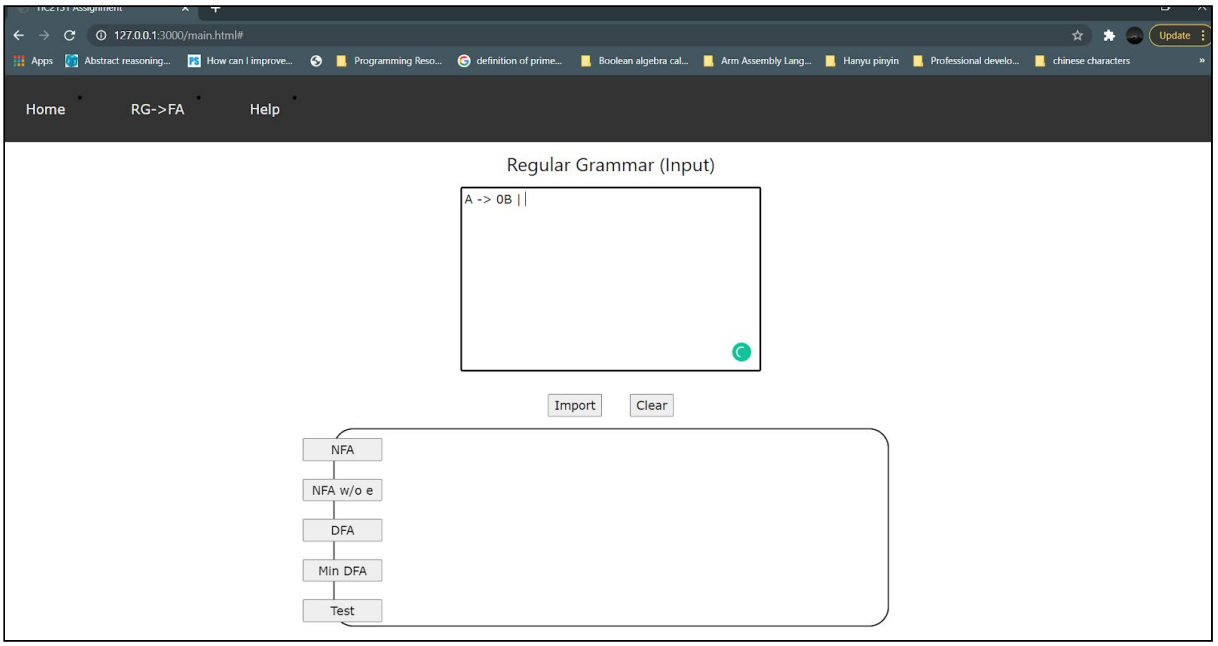


6. Other characters are strictly restricted.
7. Third, the program can only accept terminals or symbols **0, 1,** and **2**. Let's say that state A consumes a symbol 0 before reaching state B. Then you need to write down as A -> 0B as shown in figure below.

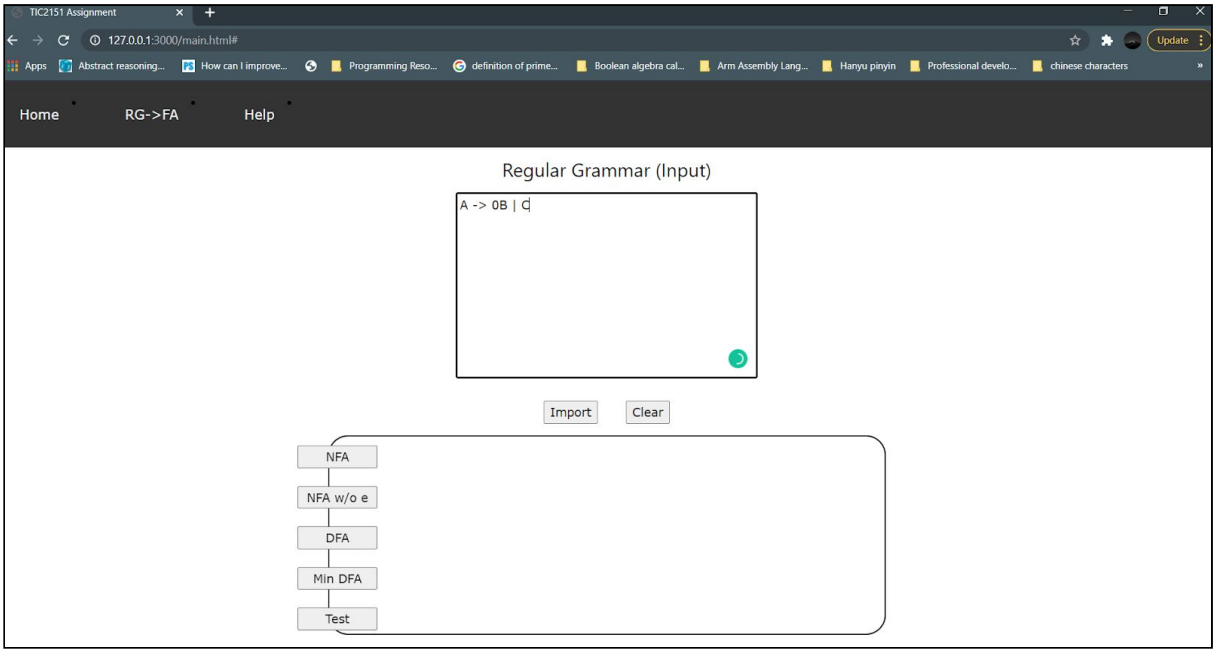


8. Fourth, you may also want to write down multiple transitions, meaning that state A might go to another state or to itself when consuming a symbol. Therefore, you must

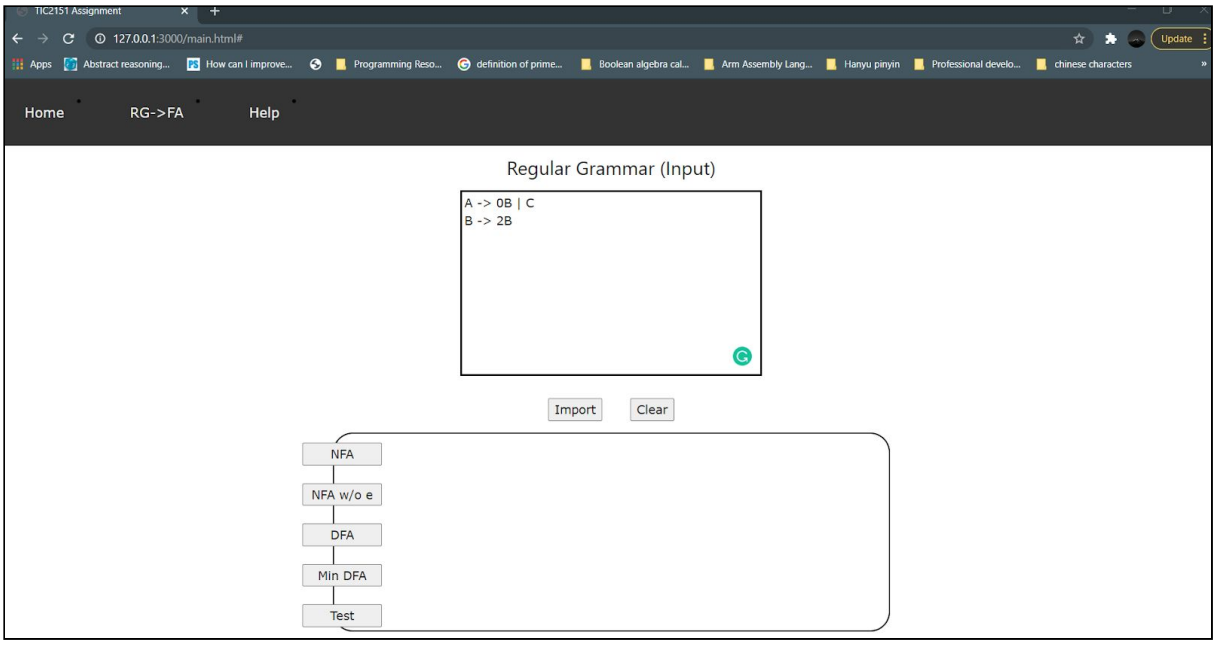
type character '|' after state B. Figure below shows how to write the correct multiple transitions.



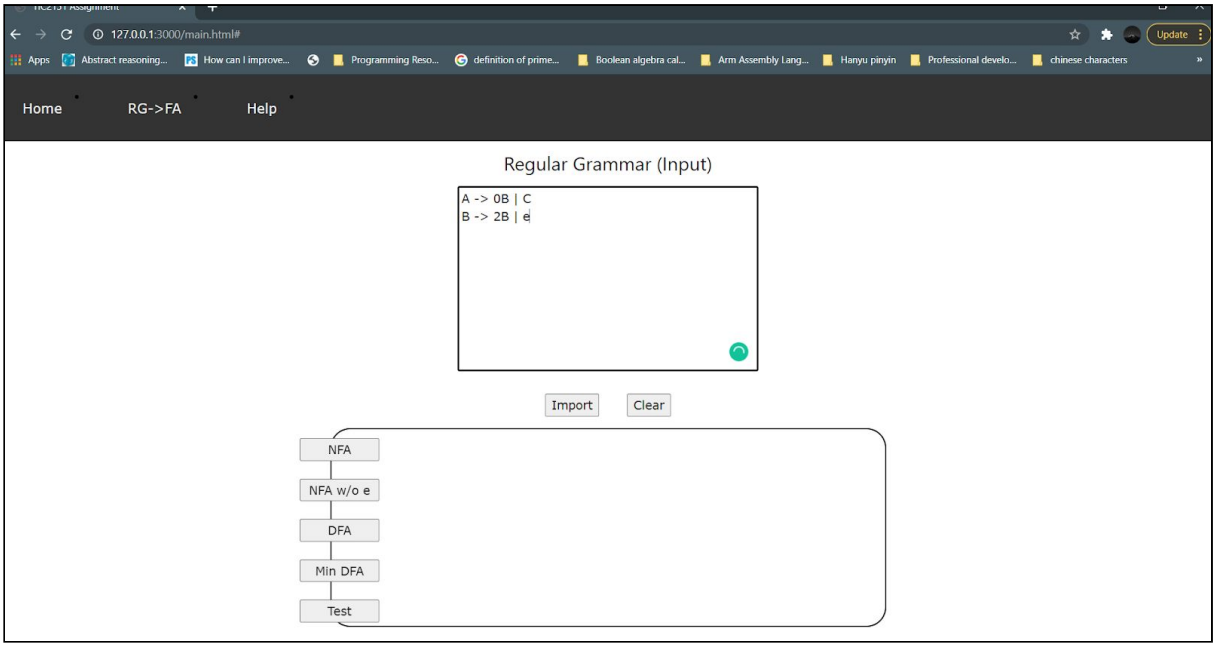
9. Note that the program is only able to accept a single symbol when moving from one state to another state. For example, **0A**, or **2B** and not **012C**.
10. Besides, if you want to make state A go through e transitions to another state (state C), you can just write down C without any symbol preceding as shown in figure below.



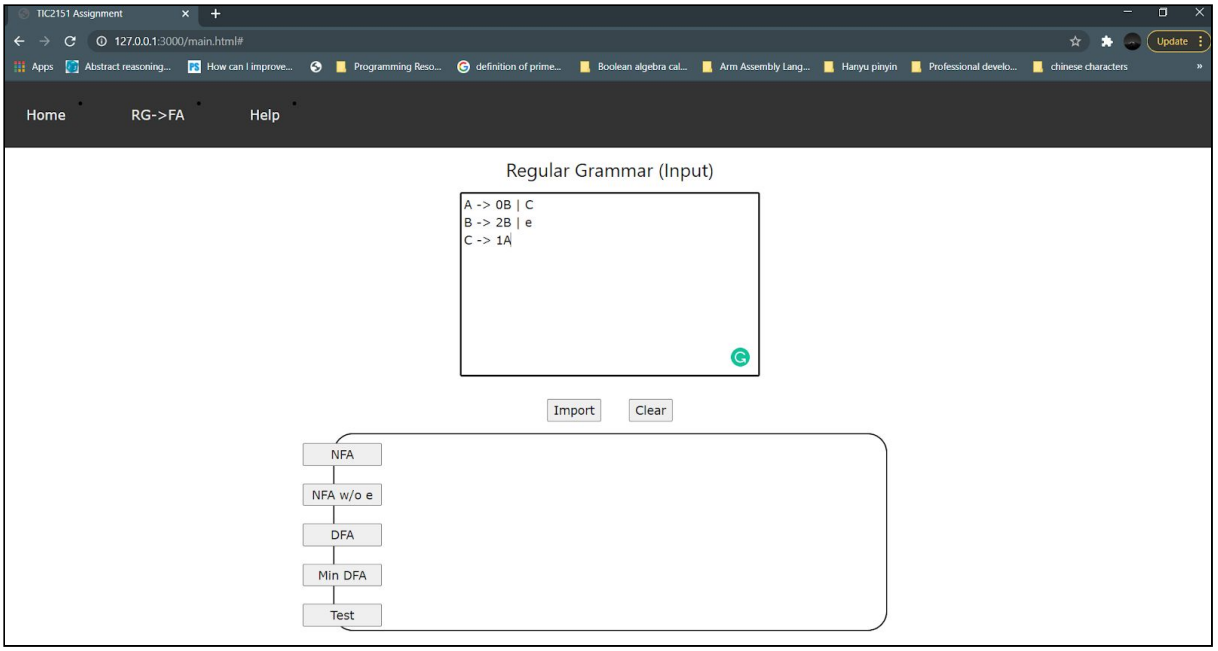
11. Then, please make sure that state B also has its own transitions by following step 3 until 11 at the next line. Figure below shows how to write down state B goes to itself when consuming symbol 2.



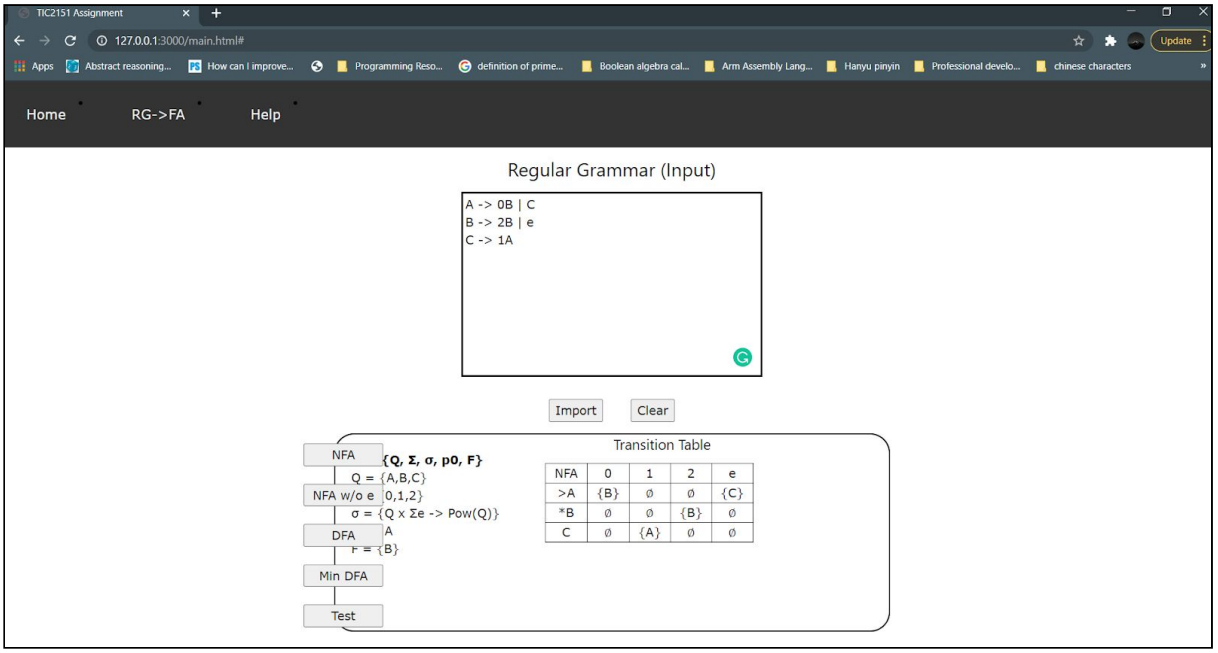
12. To represent a state is a final state, you can write that state to transit to **e**. Let's say that state B is the final state. Therefore, you need to make state B goes to e after the character '|' as shown in figure below.

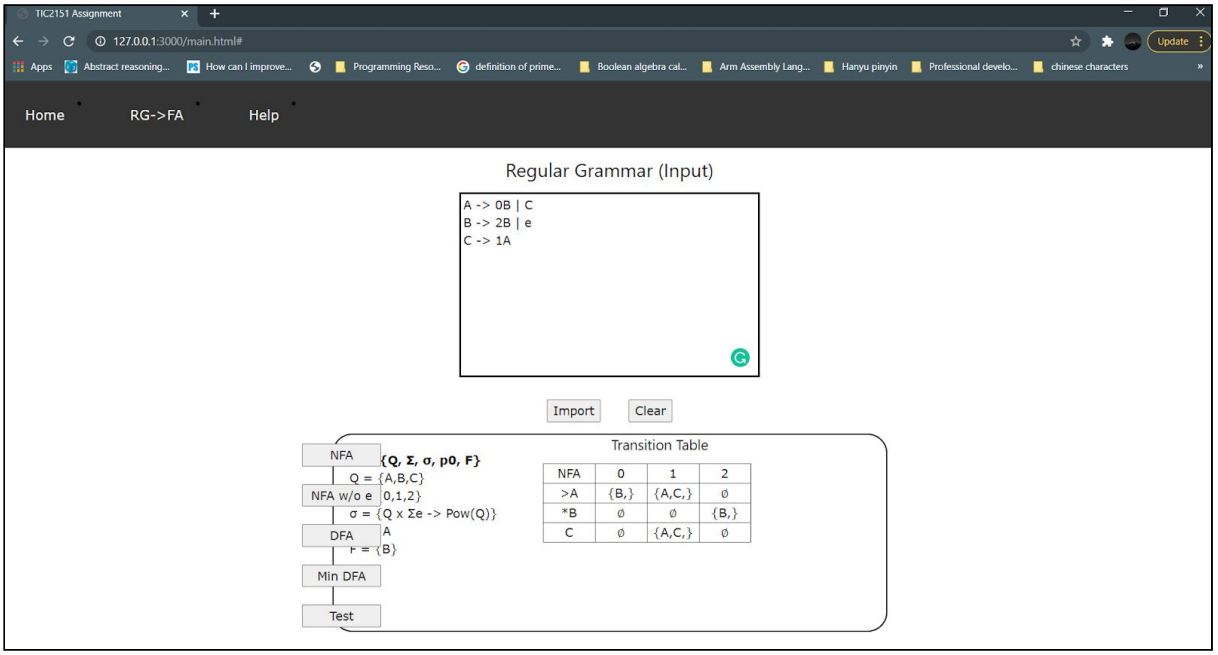


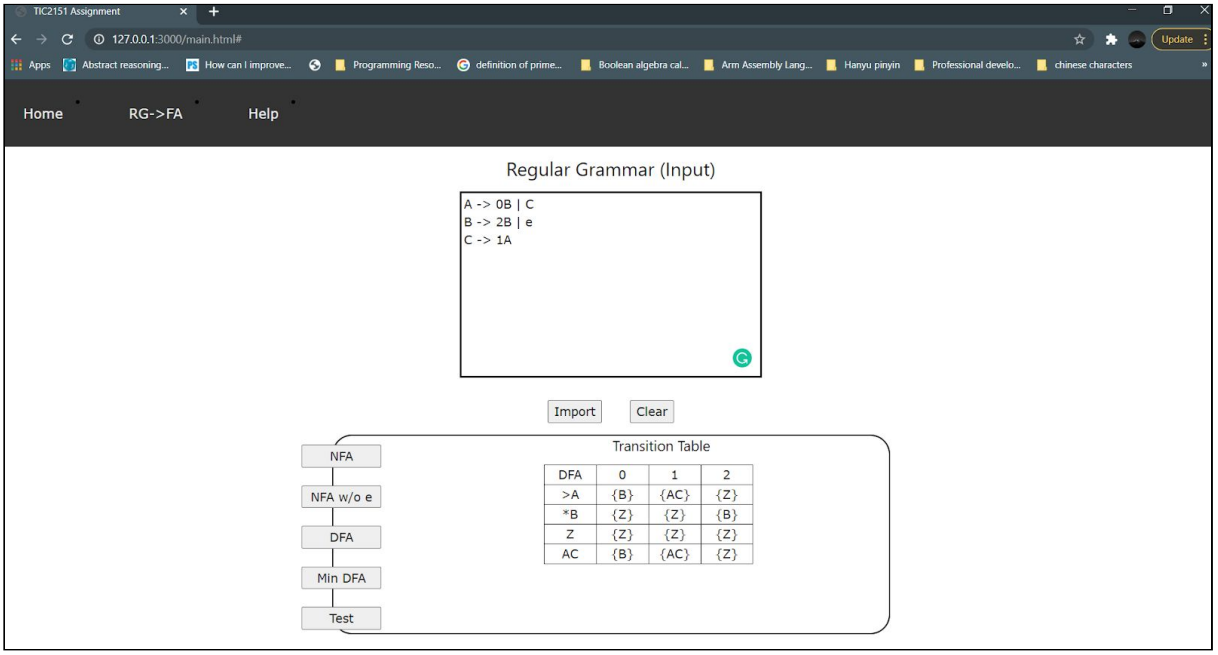
13. Based on the figure above, the program will automatically identify that state B is a final state.
14. Then, write down transitions of state C to make sure that each state has its own transitions as shown in figure below.

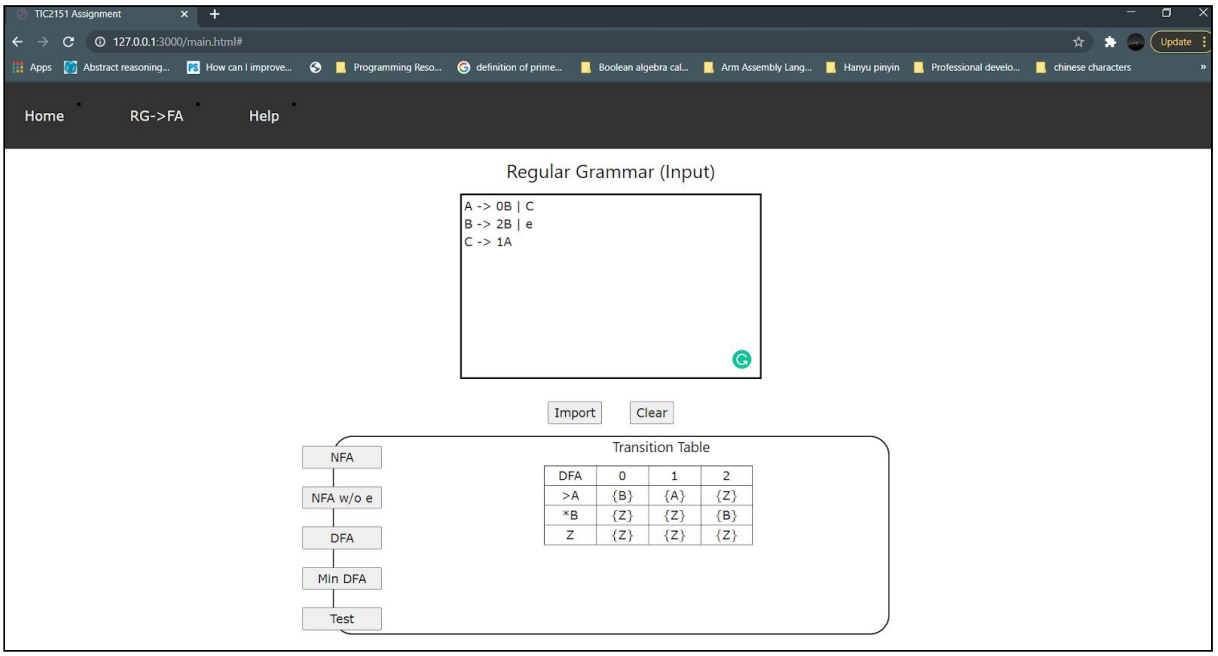


15. To see the NFA table, click the NFA button and and output of NFA table with information at the left side will appear. Same goes to the other buttons which are NFA w/o e transition, DFA, and Min DFA buttons. Figure below shows all the tables output.

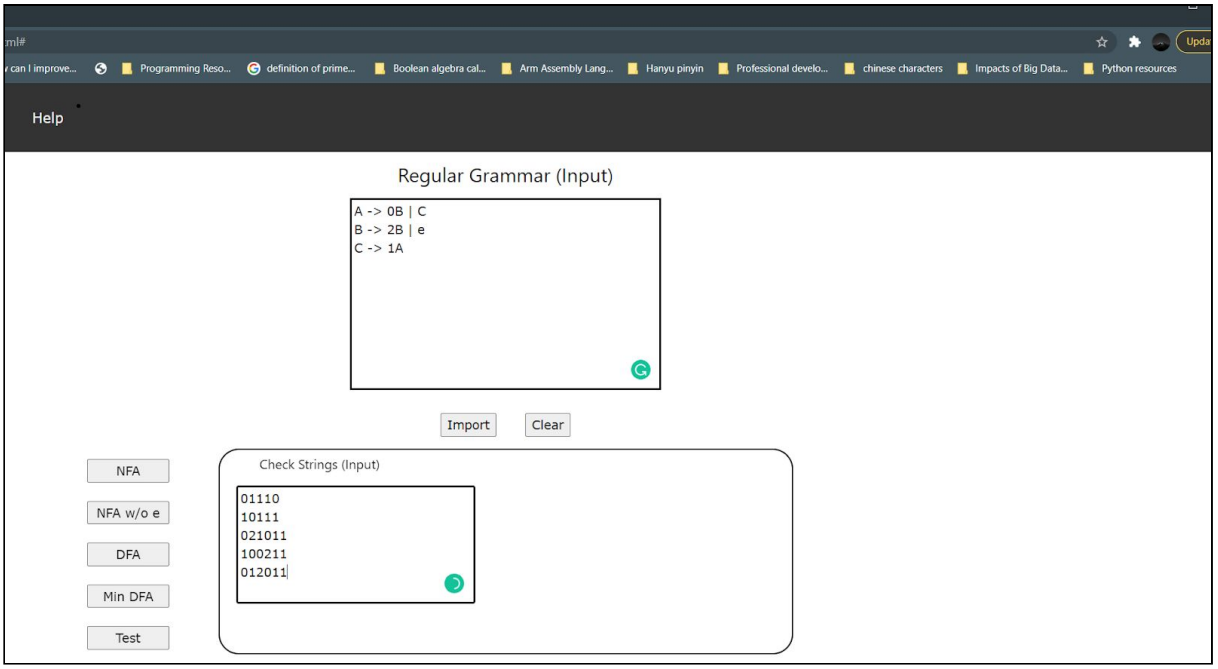



TIC2151 Assignment
127.0.0.1:3000/main.html#
Update
Apps
Abstract reasoning...
How can I improve...
Programming Reso...
definition of prime...
Boolean algebra cal...
Arm Assembly Lang...
Hanyu pinyin
Professional develo...
chinese characters
Home
RG->FA
Help
Regular Grammar (Input)
A -> 0B | C
B -> 2B | e
C -> 1A
Import
Clear
NFA
NFA w/o e
DFA
Min DFA
Test
{Q, Σ, σ, p0, F}
Q = {A,B,C}
0,1,2}
σ = {Q x Σe -> Pow(Q)}
A
F = {B}
Transition Table
NFA 0 1 2
>A {B,} {A,C,} ∅
*B ∅ ∅ {B,}
C ∅ {A,C,} ∅

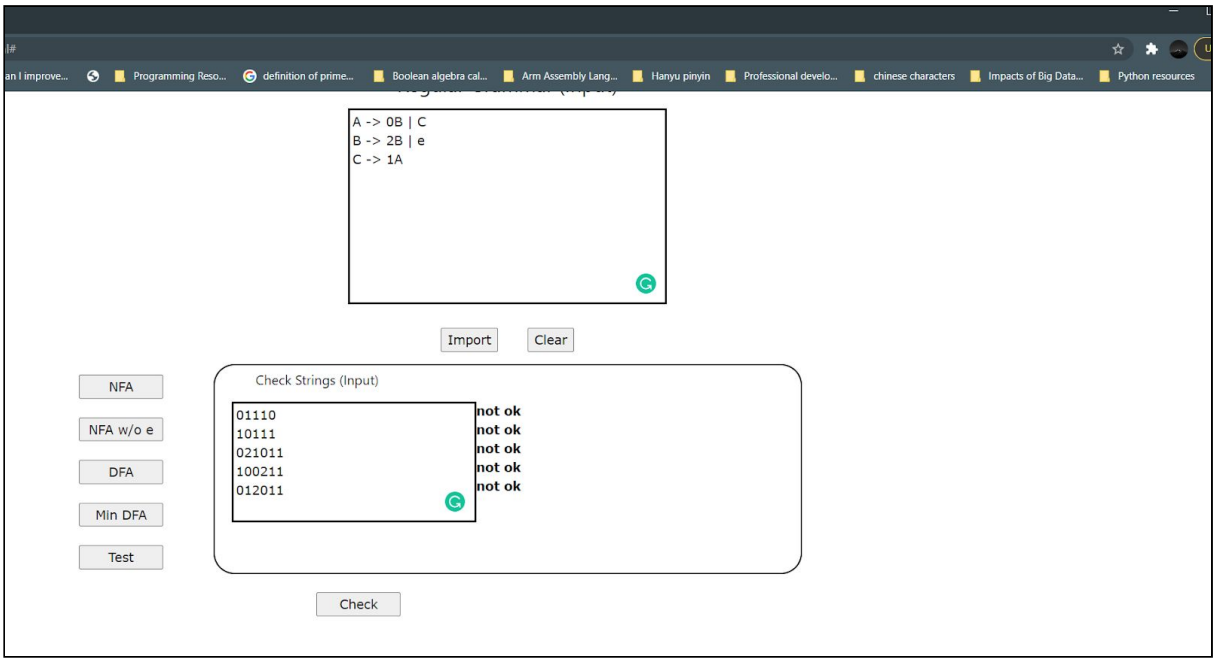

TIC2151 Assignment
127.0.0.1:3000/main.html#
Update
Apps
Abstract reasoning...
How can I improve...
Programming Reso...
definition of prime...
Boolean algebra cal...
Arm Assembly Lang...
Hanyu pinyin
Professional develo...
chinese characters
Home
RG->FA
Help
Regular Grammar (Input)
A -> 0B | C
B -> 2B | e
C -> 1A
Import
Clear
NFA
NFA w/o e
DFA
Min DFA
Test
Transition Table
DFA 0 1 2
>A {B} {AC} {Z}
*B {Z} {Z} {B}
Z {Z} {Z} {Z}
AC {B} {AC} {Z}



16. It is important to click from NFA to Min DFA in sequential order. If not then the program wouldn't be able to generate the tables.
17. To test the regular grammar, click the test button. A box to write the strings will appear. Then, you can fill up to 6 combinations of strings of 0, 1 and 2 and you must enter them line by line as shown in figure below.



18. Then click check to see the output of the test. The result can be either 'ok' or 'not ok' and it will appear at the right side of the box.



19. Instead of writing down your own input, you can also export a file containing the input by clicking on the import button.
20. Last but not least, you can clear the input by clicking on the clear button to start fill in new input.

Limitations

1. Only can read number 0, 1, and 2 as terminals or symbols.
2. No intermediate states. Meaning that a state cannot have multiple symbols consumption before reaching the target state.
3. Only one final state can be accepted.

Important codes

1. **Code from rg2.js to convert into NFA.**

```
if (state[j] == start && final.indexOf(state[j]) > -1){
  tabl += "<td>" + ">*" + state[j] + "</td>";
  for (var i = 0; i<term2.length; i++) {  // each state
    // console.log(i)
    // console.log("f&s term", term2[i])
    // console.log("f&s symb", symb[i])
    // console.log("f&s next", next[i])
    if (term2[i] == symb[i]) {
      if (next[i]=='')
        tabl += "<td>Ø</td>";
      else
        tabl += "<td>{" + next[i] + "}</td>";
    }
    else if (term2[i] !== symb[i]){
      tabl += "<td>Ø</td>"
      // console.log(symb[i+1])
      symb.splice(i+1,0,symb[i])
      next.splice(i+1,0,next[i])
    }
    else if (symb[i]=='e'){
      tabl += "<td>Ø</td>"
    }
  }
}
else if (state[j] == start){
  tabl += "<td>" + ">" + state[j] + "</td>";
  for (var i = 0; i<term2.length; i++) {
    // console.log(i)
    // console.log("s term", term2[i])
    // console.log("s symb", symb[i])
    // console.log("s next", next[i])
    if (term2[i] == symb[i]) {
      if (next[i]=='')
        tabl += "<td>Ø</td>";
      else
        tabl += "<td>{" + next[i] + "}</td>";
    }
    else if (term2[i] !== symb[i]){
      tabl += "<td>Ø</td>"
      // console.log(symb[i+1])
      symb.splice(i+1,0,symb[i])
      next.splice(i+1,0,next[i])
    }
    else if (symb[i]=='e'){
```

2. **Code from nfawe.js to convert into NFA with e transition.**

```
var tabl = "<table><tr>";
    tabl += "<h5>" + "Transition Table" + "</h5>";
    tabl += "<td>" + "NFA" + "</td>";

for (var i = 0; i<term2.length; i++){
  tabl += "<td>" + term2[i] + "</td>";
}

for (var j = 0; j<tr1.length; j++){ //each line
// console.log(leng.length)
tabl += "<tr>";

var symb = Object.keys(tr1[j]);
var next = Object.values(tr1[j]);


console.log("OIIIIIIIIIIIIIIIIIIII", symb)
/*
console.log("OIIIIIIIIIIIIIIIIIIII", symb2)
console.log("OIIIIIIIIIIIIIIIIIIII", Object.keys(tr[tr.length-1]))
*/

if (state[j] == start && final.indexOf(state[j]) > -1){
  tabl += "<td>" + ">*" + state[j] + "</td>";
  for (var i = 0; i<term2.length; i++) {  // each state
    // console.log(i)
    // console.log("f&s term", term2[i])
    // console.log("f&s symb", symb[i])
    // console.log("f&s next", next[i])
    if (term2[i] == symb[i]) {
      if (next[i]=='')
        tabl += "<td>Ø</td>";
      else
        tabl += "<td>{" + next[i] + "}</td>";
    }
    else if (term2[i] !== symb[i]){
      tabl += "<td>Ø</td>"
      // console.log(symb[i+1])
      symb.splice(i+1,0,symb[i])
      next.splice(i+1,0,next[i])
    }
    else if (symb[i]=='e'){
      tabl += "<td>Ø</td>"
```

3. **Code from dfa.js to convert into DFA.**

```
var tabl = "<table><tr>";
tabl += "<h5>" + "Transition Table" + "</h5>";
tabl += "<td>" + "DFA" + "</td>";

var start = regular.getStartVar();
var final = regular.getFinalVar();

var state5 = Object.keys(dfa)
var tr1 = Object.values(dfa)

console.log(Object.keys(tr1[0]))

for (var i = 0; i<term2.length; i++){
  tabl += "<td>" + term2[i] + "</td>";
}

for (var j = 0; j<tr1.length; j++){
  tabl += "<tr>"

  var symb = Object.keys(tr1[j]);
  var next = Object.values(tr1[j]);

  if (state5[j] == start && state5[j].includes(final) == true){
    tabl += "<td>" + ">*" + state5[j].replace(/[\])}[{(,]/g, '') + "</td>";
    for (var i = 0; i<term2.length; i++) {  // each state
      // console.log(i)
      // console.log("f&s term", term2[i])
      // console.log("f&s symb", symb[i])
      // console.log("f&s next", next[i])
      if (term2[i] == symb[i]) {
        if (next[i]=='')
          tabl += "<td>Ø</td>";
        else
          tabl += "<td>{" + next[i][0].replace(/[\])}[{(,]/g, '') + "}</td>";
      }
      else if (term2[i] !== symb[i]){
        tabl += "<td>Ø</td>"
        // console.log(symb[i+1])
        symb.splice(i+1,0,symb[i])
        next.splice(i+1,0,next[i])
      }
      else if (symb[i]=='e'){
```

4. **Code from mdfa.js to convert into minimized DFA.**

```
var tabl = "<table><tr>";
tabl += "<h5>" + "Transition Table" + "</h5>";
tabl += "<td>" + "DFA" + "</td>";

var start = regular.getStartVar();
var final = regular.getFinalVar();

var state5 = Object.keys(mdfa)
var tr1 = Object.values(mdfa)

console.log(Object.keys(tr1[0]))

for (var i = 0; i<term2.length; i++){
        tabl += "<td>" + term2[i] + "</td>";
}

for (var j = 0; j<tr1.length; j++){
    tabl += "<tr>"

    var symb = Object.keys(tr1[j]);
        var next = Object.values(tr1[j]);

    if (state5[j] == start && state5[j].includes(final) == true){
        tabl += "<td>" + ">*" + state5[j].replace(/[\])}[{(,]/g, '') + "</td>";
        for (var i = 0; i<term2.length; i++) {  // each state
            // console.log(i)
            // console.log("f&s term", term2[i])
            // console.log("f&s symb", symb[i])
            // console.log("f&s next", next[i])
            if (term2[i] == symb[i]) {
                if (next[i]=='')
                    tabl += "<td>Ø</td>";
                else
                    tabl += "<td>{" + next[i][0].replace(/[\])}[{(,]/g, '') + "}</td>";
            }
            else if (term2[i] !== symb[i]){
                tabl += "<td>Ø</td>"
                // console.log(symb[i+1])
                symb.splice(i+1,0,symb[i])
                next.splice(i+1,0,next[i])
            }
            else if (symb[i]=='e'){
                tabl += "<td>Ø</td>"
            }
```

5. **Code from check.js to check the strings.**

```
var currentState = regular.getStartVar();  //current state is assigned with the regular start state
var results = [];  //array of results, ['ok'] @ ['not ok']
for (var i = 0; i < arr.length; i++) {  //first is loops through each input
  for (var j = 0; j < arr[i].length; j++) {  //loops single number like 0101, access 0 first..then 0
    currentState = regular.getStartVar();  //
    console.log(arr[i].length);
    for (var k = 0; k < production.length; k++) {
      console.log(k +' number of symbol -> '+ production[k].getSymbol(), production[k].getNextState())
      console.log(currentState, production[k].getCurrentState());
      if(currentState == production[k].getCurrentState()) {
        var inputAndCurrentIsChanged = false;
        console.log('00000000000')
        if(production[k].getSymbol() == arr[i][j]) { // if 0011, then arr[i][j] represents each single number
            console.log(currentState  + "->" + arr[i][j] + production[k].getNextState());
            currentState = production[k].getNextState();
            inputAndCurrentIsChanged = true;
            k=-1;
            j += 1;
            console.log(j)
            console.log("11111111111111")
        }
        else if (production[k].getSymbol() == 'e') { // the current state is final state and the input is at last index
          console.log('e');
          if ((j >= arr[i].length) && (finalStates.indexOf(currentState) > -1)){ // if input reach the end and the current state
            console.log(currentState + " is end state");
            inputAndCurrentIsChanged = true;
            console.log("22222222222222")
          }
          else if ((j < arr[i].length) && (finalStates.indexOf(currentState) == -1)) { // if input havent reach the end and the
            console.log(currentState  + "->" + arr[i][j] + production[k].getNextState());
            currentState = production[k].getNextState();
            inputAndCurrentIsChanged = true;
            k=-1; // go back to first production because already change state
            j += 1;
            console.log(j)
            console.log("333333333333333")
          }
          else if ((j < arr[i].length) && (finalStates.indexOf(currentState) > -1)) { // if input havent reach the end and the c
            inputAndCurrentIsChanged = false;
            j = arr[i].length - 1;
            console.log("444444444444444")
            break;
          }
```