Ashish Simon Rattan 102116085 4CS11

A Minimized DFA for the Regular Expression (a/b)*abb

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <stack>
#include <map>
#include <cstring>
using namespace std;
#define SIZE 30

int init[20], final[20], nfa_init_size = 0, nfa_fin_size = 0;
string init_dfa[SIZE], final_dfa[SIZE];
int dfa_init_size = 0, dfa_fin_size = 0;

void print_initial_final(string type="nfa") {
   int init_count = nfa_init_size;
   int final_count = nfa_fin_size;
   if(type == "dfa"){
      init_count = dfa_init_size;
      final_count = dfa_fin_size;
   }
   cout << " initial state = ";
   for(int i = 0; i < init_count; i++)
   if(type=="nfa") cout << init[i] << " ";
   else cout << init_dfa[i] << " ";

   cout << endl;
   cout << "final state =";
   for(int i = 0; i < final_count; i++)
   if(type=="nfa") cout << final[i] << " ";
   else cout << final_dfa[i] << " ";
   cout << endl;
}

void reduce_fin(int x) {
   for(int i = x; i < nfa_fin_size - 1; i++)
      final[i] = final[i + 1];
   nfa_fin_size -= 1;
}

bool is_alphabet(char c) {
    if(c>=97 && c<=122) return true;
    else return false;
}

string formatting(string s) {
    auto l = s.length();
    string p_string;
    p_string.push_back('(');
    for(int i = 0; i < l; i++) {
        p_string.push_back(s[i]);
        if(is_alphabet(s[i]) && is_alphabet(s[i+1]))
            p_string.push_back('.');

        else if(s[i] == ')' && s[i + 1] == '(')
            p_string.push_back('.');
        else if(is_alphabet(s[i]) && s[i + 1] == '(')
            p_string.push_back('.');
        else if(s[i] == ')' && is_alphabet(s[i+1]))
            p_string.push_back('.');
        else if(s[i] == '*' && (s[i + 1] == '(' || is_alphabet(s[i+1])))
            p_string.push_back('.');
    }

    p_string.push_back(')');
    return p_string;
}

string regex_to_postfix(string s) {
    int l = s.length();
    vector<char> operands;
    stack<char> operators;
    for(int i = 0; i < l; i++) {
        char x = s[i];
        if(s[i]=='a' || s[i]=='b' ) operands.push_back(s[i]);
        switch(x) {

            case '(':
                operators.push('(');
                break;

            case ')':
            //   @_@
            while(operators.top()!='('){
                        operands.push_back(operators.top());
                        operators.pop();
            }
        break;

    case '.':
        if(operators.empty())
            operators.push('.');
        else {
            char temp = operators.top();
            if(temp == '(')
                operators.push('.');
            else if(temp == '*') {
                operands.push_back(operators.top());
                operators.pop();
                if(operators.top() == '.') {
                    operands.push_back('.');
                }
                else
                operators.push('.');
            }
            else if(temp == '.') {
                operands.push_back(operators.top());
                operators.pop();
                operators.push('.');
            }
            else if(temp == '|')
                operators.push('.');
        }
        break;

    case '|':
        if(operators.empty())
            operators.push('|');
        else {
            char temp = operators.top();
            if(temp == '(')
                operators.push('|');
            else if(temp == '*') {
            operands.push_back(operators.top());
            operators.pop();
            operators.push('|');
            }
            else if(temp == '.') {
                operands.push_back(operators.top());
                operators.pop();
                operators.push('|');
```

```
                    }
                }
                break;

            case '*':
                if(operators.empty())
                    operators.push('*');
                else {
                    char temp = operators.top();
                    if(temp == '(' || temp == '.' || temp == '|')
                        operators.push('*');
                    else {
                        operands.push_back(operators.top());
                        operators.pop();
                        operators.push('*');
                    }
                }
                break;
        }
    }
    string p;
    for(int i = 0; i < operands.size(); i++)
        p += operands[i];
    return p;
}

int thompsons_construction(string s, int table_NFA[][5]) {
    int l = s.length();
    int states = 1;
    int m, n, j, count;
    for(int i = 0; i < l; i++) {
        char x = s[i];
        switch(x) {
            case 'a':
                table_NFA[states][0] = states;
                init[nfa_init_size] = states;
                nfa_init_size += 1;
                states += 1;
                table_NFA[states - 1][1] = states;
                final[nfa_fin_size] = states;
                nfa_fin_size += 1;
                table_NFA[states][0] = states;
                states += 1;
                break;
```

```
case 'b':
   table_NFA[states][0] = states;
   init[nfa_init_size] = states;
   nfa_init_size += 1;
   states += 1;
   table_NFA[states - 1][2] = states;
   final[nfa_fin_size] = states;
   nfa_fin_size += 1;
   table_NFA[states][0] = states;
   states += 1;
   break;

case '.':
   m = final[nfa_fin_size - 2];
   n = init[nfa_init_size - 1];
   table_NFA[m][3] = n;
   reduce_fin(nfa_fin_size - 2);
   nfa_init_size -= 1;
   break;

case '|':
   for(j = nfa_init_size - 1, count = 0; count < 2; count++) {
      m = init[j - count];
      table_NFA[states][3 + count] = m;
   }
   nfa_init_size = nfa_init_size - 2;
   init[nfa_init_size] = states;
   nfa_init_size += 1;
   table_NFA[states][0] = states;
   states += 1;
   for(j = nfa_fin_size - 1, count = 0; count < 2; count++) {
      m = final[j - count];
      table_NFA[m][3] = states;
   }
   nfa_fin_size = nfa_fin_size - 2;
   final[nfa_fin_size] = states;
   nfa_fin_size += 1;
   table_NFA[states][0] = states;
   states += 1;
   break;

case '*':
   m = init[nfa_init_size- 1];
   table_NFA[states][3] = m;
   table_NFA[states][0] = states;
```

```
            init[nfa_init_size- 1] = states;
            states += 1;
            n = final[nfa_fin_size- 1];
            table_NFA[n][3] = m;
            table_NFA[n][4] = states;
            table_NFA[states - 1][4] = states;
            final[nfa_fin_size- 1] = states;
            table_NFA[states][0] = states;
            states += 1;
            break;
        }
    }
    return states;
}

void print_NFA_table(int table_NFA[][5], int states) {
    cout << endl;
    cout << setw(43) << "NFA Transition Table" << endl << endl;
    cout << setw(10) << "States" << setw(10) << "a" << setw(10) << "b" << setw(10)
<< "e" << setw(10) << "e" << endl;
    for(int i = 0; i < 50; i++)
        cout << "-";
    cout << endl;
    for(int i = 1; i < states; i++) {
        for(int j = 0; j < 5; j++) {
            if(table_NFA[i][j] == -1)
                cout << setw(10) << "  ";
            else
                cout << setw(10) << table_NFA[i][j];
        }
        cout << endl;
    }
    cout << endl;
    print_initial_final();
}

void print_DFA_table(string table_DFA[][3], int state) {
    cout << endl << endl;
    cout << setw(43) << "DFA Transition Table" << endl << endl;
    cout << setw(10) << "States" << setw(10) << "a" << setw(10) << "b" << endl;
    for(int i = 0; i < 60; i++)
        cout << "-";
    cout << endl;
    for(int i = 0; i < state; i++){
        for(int j = 0; j < 3; j++)
```

```
            cout << setw(10) << table_DFA[i][j];
        cout << endl;
    }
    cout << endl;

    print_initial_final("dfa");
}

vector<int> e_closure(int table_NFA[][5], int x) {
    stack<int> s;
    map<int, int> m;
    vector<int> v;
    int y;
    s.push(x);
    m[x] = 1;
    while(!s.empty()) {
        y = s.top();
        s.pop();
        if(table_NFA[y][3] == -1)
            continue;
        else {
            s.push(table_NFA[y][3]);
            m[table_NFA[y][3]] = 1;
            if (table_NFA[y][4] == -1)
                continue;
            else {
                s.push(table_NFA[y][4]);
                m[table_NFA[y][4]] = -1;
            }
        }
    }
    map<int, int>::iterator itr;
    itr = m.begin();
    while (itr != m.end()) {
        v.push_back(itr->first);
        itr++;
    }
    return v;
}

string state_name(int i) {
    char s = 'q';
    string p;
    p += s;
    if(i == 0) {
        p += '0';
        return p;
    }
    int a[100];
    int j = 0;
    while(i > 0) {
        int x = i % 10;
        a[j] = x;
        j += 1;
        i = i / 10;
    }
    for(int i = j - 1; i >= 0; i--) {
        int x = a[i];
        p += (x + '0');
    }
    return p;
}

void init_CHECK(vector<int> v, string s) {
    for(int i = 0; i < v.size(); i++) {
        if(v[i] == init[0]) {
            init_dfa[dfa_init_size] = s;
            dfa_init_size += 1;
        }
    }
}

void final_CHECK(vector<int> v, string s) {
    for(int i = 0; i < v.size(); i++) {
        if(v[i] == final[0]) {
            final_dfa[dfa_fin_size] = s;
            dfa_fin_size += 1;
        }
    }
}

bool check_a_b(string word) {
    auto len = word.length();
    int i = 0;
    for(i = 0; i < len; i++) {
        if(word[i] == 'a' || word[i] == 'b')
            continue;
        else
            return false;
    }
    if(i == len)
        return true;
    return false;
}

int NFA_to_DFA(int table_NFA[][5], int states, string table_DFA[][3]) {
    bool flag[states];
    memset(flag, true, sizeof(flag));
    int state = 0, j = 0;
    map<vector<int>, string> map_e_to_state;
    vector<int> v, v1, v2, v3, v4;
    v = e_closure(table_NFA, init[0]);
    flag[init[nfa_init_size]] = false;
    map_e_to_state[v] = state_name(j++);
    init_CHECK(v, map_e_to_state[v]);
    final_CHECK(v, map_e_to_state[v]);
    stack<vector<int> > st;
    st.push(v);
    int count = 0;
    while(true) {
        while(!st.empty()) {
            vector<int> v;
            v = st.top();
            st.pop();
            count += 1;
            table_DFA[state][0] = map_e_to_state[v];
            for(int i = 0; i < v.size(); i++) {
                flag[v[i]] = false;
                int temp = table_NFA[v[i]][1];
                int temp1 = table_NFA[v[i]][2];
                if (temp >= 0)
                    v1.push_back(temp);
                if (temp1 >= 0)
                    v3.push_back(temp1);
            }
            map<int, int> map_temp, map_temp1;
            map<int, int>::iterator it;
            for(int i = 0; i < v1.size(); i++) {
                v2 = e_closure(table_NFA, v1[i]);
                for(int j = 0; j < v2.size(); j++)
                    map_temp[v2[j]] = 1;
                v2.clear();
            }
            for(int i = 0; i < v3.size(); i++) {
                v4 = e_closure(table_NFA, v3[i]);
                for(int j = 0; j < v4.size(); j++)
                    map_temp1[v4[j]] = 1;
                v4.clear();
            }
            for(it = map_temp.begin(); it != map_temp.end(); it++) {
                v2.push_back(it->first);
                flag[it->first] = false;
            }
            for(it = map_temp1.begin(); it != map_temp1.end(); it++) {
                v4.push_back(it->first);
                flag[it->first] = false;
            }
            if(v2.empty())
                table_DFA[state][1] = "--";
            else {
                string t = map_e_to_state[v2];
                char flagg = t[0];
                if(flagg == 'q')
                    table_DFA[state][1] = map_e_to_state[v2];
                else {
                    table_DFA[state][1] = state_name(j++);
                    map_e_to_state[v2] = table_DFA[state][1];
                    init_CHECK(v2, map_e_to_state[v2]);
                    final_CHECK(v2, map_e_to_state[v2]);
                    st.push(v2);
                }
            }
            if(v4.empty())
                table_DFA[state][2] = "--";
            else {
                string t = map_e_to_state[v4];
                char flagg = t[0];
                if(flagg == 'q')
                    table_DFA[state][2] = map_e_to_state[v4];
                else {
                    table_DFA[state][2] = state_name(j++);
                    map_e_to_state[v4] = table_DFA[state][2];
                    init_CHECK(v4, map_e_to_state[v4]);
                    final_CHECK(v4, map_e_to_state[v4]);
                    st.push(v4);
                }
            }
            v1.clear();
            v2.clear();
            v3.clear();
```

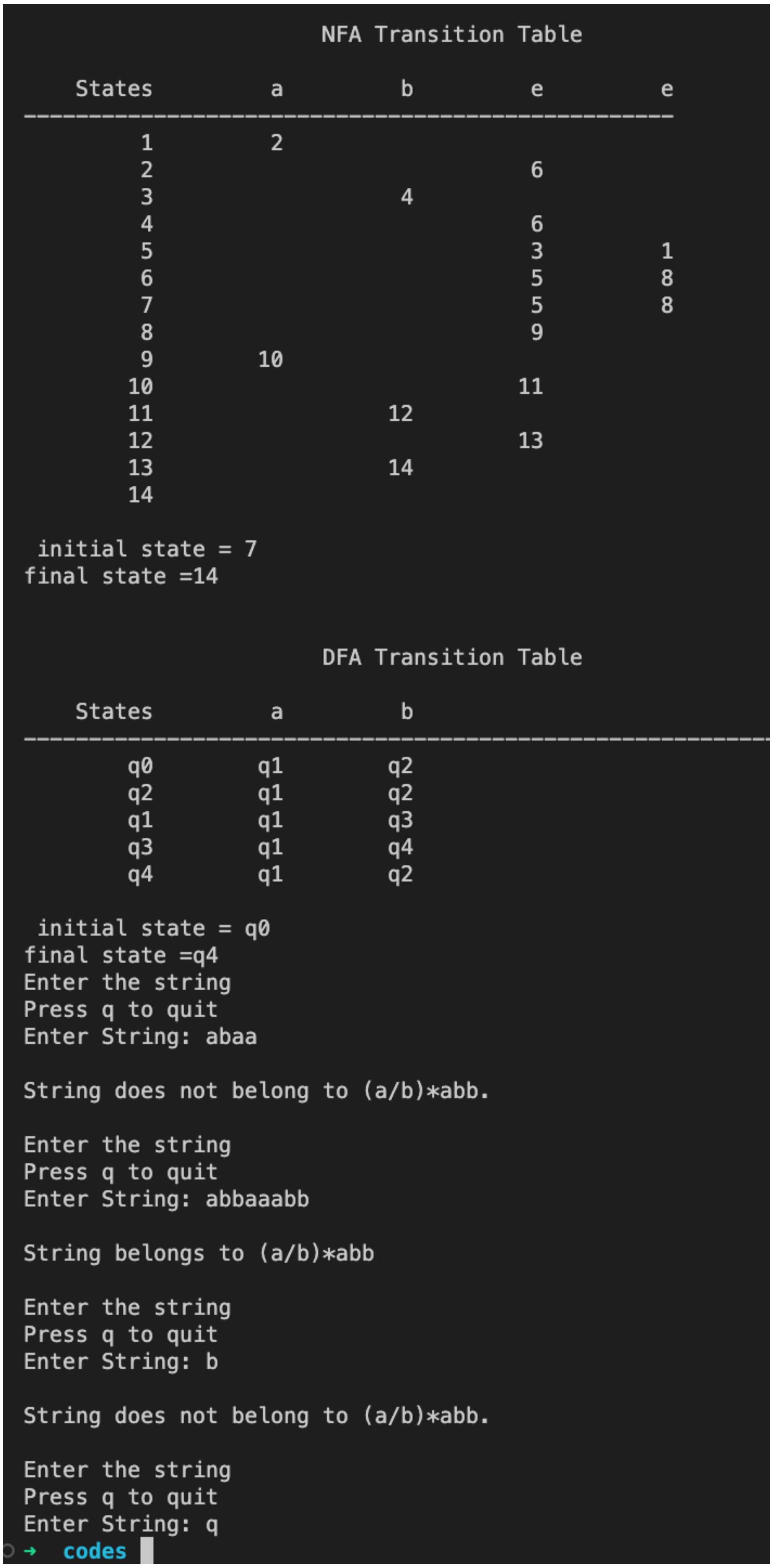
```
            v4.clear();
            state += 1;
        }
        int k = 1;
        for(k = 1; k < states; k++) {
            if(flag[k]) {
                v = e_closure(table_NFA, k);
                map_e_to_state[v] = state_name(j++);
                init_CHECK(v, map_e_to_state[v]);
                final_CHECK(v, map_e_to_state[v]);
                cout << endl << map_e_to_state[v] << " represents :- ";
                for(int i = 0; i < v.size(); i++)
                    cout << v[i] << " ";
                cout << endl;
                st.push(v);
                break;
            }
        }
        if(k == states)
            break;
    }
    print_DFA_table(table_DFA, state);
    return state;
}

void run_code(string table_DFA[][3], string word, int state) {
    auto len = word.length();
    string temp = init_dfa[0];
    bool check = check_a_b(word);
    if(!check)
        temp = "  ";
    int i = 0;
    for(i = 0; i < len; i++) {
        if(temp == "  ") {
            cout << endl << "String does not belong to (a/b)*abb" << endl << endl <<
endl;
            break;
        }
        else {
            int j = 0;
            for(j = 0; j < state; j++)
                if(temp == table_DFA[j][0])
                    break;
            if(word[i] == 'a')
                temp = table_DFA[j][1];
            else if(word[i] == 'b')
                temp = table_DFA[j][2];
        }
    }
    if(i == len) {
        int j = 0;
        for(j = 0; j < dfa_fin_size; j++) {
            if(temp == final_dfa[j]) {
                cout << endl << "String belongs to (a/b)*abb" << endl << endl;
                break;
            }
        }
        if(j == dfa_fin_size)
            cout << endl << "String does not belong to (a/b)*abb." << endl << endl;
    }
}

int main() {
    int table_NFA[1000][5];
    for(int i = 0; i < 1000; i++)
    for(int j = 0; j < 5; j++)
    table_NFA[i][j] = -1;
    int states = 0;
    string regex = "(a|b)*abb";
    regex = formatting(regex);
    regex = regex_to_postfix(regex);
    states = thompsons_construction(regex, table_NFA);
    print_NFA_table(table_NFA, states);
    string table_DFA[1000][3];
    int State_DFA = NFA_to_DFA(table_NFA, states, table_DFA);

    while(true) {
        string word;
        cout << "Enter the string" << endl;
        cout << "Press q to quit" << endl;
        cout << "Enter String: ";
        cin >> word;
        if(word == "q")
            break;
        run_code(table_DFA, word, State_DFA);
    }
    return 0;
}
```

OUTPUT:



```
                        NFA Transition Table

    States         a         b         e         e
--------------------------------------------------
         1         2
         2                             6
         3                   4
         4                             6
         5                             3         1
         6                             5         8
         7                             5         8
         8                             9
         9        10
        10                            11
        11                  12
        12                            13
        13                  14
        14

 initial state = 7
final state =14


                        DFA Transition Table

    States         a         b
------------------------------------------------------------
        q0        q1        q2
        q2        q1        q2
        q1        q1        q3
        q3        q1        q4
        q4        q1        q2

 initial state = q0
final state =q4
Enter the string
Press q to quit
Enter String: abaa

String does not belong to (a/b)*abb.

Enter the string
Press q to quit
Enter String: abbaaabb

String belongs to (a/b)*abb

Enter the string
Press q to quit
Enter String: b

String does not belong to (a/b)*abb.

Enter the string
Press q to quit
Enter String: q
○ ➜  codes
```