01) Enas Gaber
02) Dina Hossam
03) Monica Salama
04) Mostafa Gabriel

a. A description of the used data structures.

NFA node :

- id
- A number indicates a state ‘s final state
- A map with the key input character and value is all transitions from current state with this input

DFA node :

- id
- A number indicates a state ‘s final state
- A vector with all the NFA states of the current DFA state

- vector <vector> Transition table : to store the transitions of DFA states
- queue<vectors>: to get the minimized states.

b. Explanation of all algorithms and techniques used

<u>From lexical rules to NFA:</u>

1- Transform RE to Postfix:

- All variables are added to the output expression. Left parentheses are always pushed onto the stack. When a right parenthesis is encountered, symbols on top of the stack are popped and copied to the resulting expression until the symbol at the top of the stack is a left parenthesis. At that time, parentheses are discarded.
- If the symbol being scanned has a higher precedence than the symbol at the top of the stack,the symbol being scanned is pushed onto the stack and the scan pointer is advanced.
- If the precedence of the symbol being scanned is lower than or equal to the precedence of the symbol at the top of the stack, one element of the stack is popped to the output and the scan pointer is not advanced. So, the symbol being scanned will be compared with the new top element on the stack.
- When the end of the expression is encountered, the stack is popped to the output.

2- Evaluate Postfix :

- The postfix expression is scanned left-to-right. Operands are placed on a stack until an operator is found.
- Operands are then removed from the stack, the operation is performed on the operands and the result is pushed on to the stack.
- The process continues until the end of expression is reached.

3- NFA operations:

- Nfa from char
- Oring 2 NFA’s
- Concatenate 2 NFA’s
- Zero or more

Conversion from NFA to DFA Algorithm :

The start state of DFA is epsilon_Closure(start state in the NFA) , and the accepting states of DFA are all those sets of N's states that include at least one accepting state of N

```
initially, epsilon_Closure(start state ) is the only state in DFA states, and it is unmarked;
while ( there is an unmarked state T in DFA states ) {
        mark T;
        for ( each input symbol a ) {
                U =epsilon_Closure ( move(T, a))
                if ( U is not in Dstates )
                        add U as an unmarked state to DFA states
                Add U in the transition table with input a
        }
}
```

Compute the epsilon closure of a state T :

```
epsilon_Closure(T) {
        push all states of T onto stack;
        initialize epsilon_closure(T) to T;
        while ( stack is not empty ) {
                pop t, the top element, off stack;
                 for ( each state u with an edge from t to u labeled t )
                 if ( u is not in epsilon_closure(T) ) {
                          add u to epsilon_closure(T)
                          push u onto stack
                }
            }
         }
}
```

Minimizing the DFA:

Given a vector of states and the transition table and the input.
1 - put the states into classes(vectors) according to the final states.
2 - put the class into queue of vectors.
3 - give each class an id.
4 - while (size of queue > 0) {
take each class with input and compare each state in it,
While (input > 0) {
if (they are going to the same class)
put it in the same vector,
else
but it alone in a vector.
}
}
5 - repeat step from 2 to 4 while the size of queue changes.
6 - if the queue doesn't change then the vectors inside the queue are the final result.

Getting tokens

- Linear time algorithm that reads the source code line by line then split each line on some tokens that will try to match these tokens to one of the accepted ones

c. The resultant transition table for the minimal DFA.

- Attached in the submission "*TransitionTable.csv*"

d. The resultant stream of tokens for the example test program.

int
id
,
id
,
id
,
id
;
while
(
id
relop
num
)
{
id
assign
id
addop
num
;
}