Course: Compiler Construction

Week 3: Finite Automata (FA)

Lecturer: Martha Gichuki

Learning outcomes Week 3: Finite Automata (FA)

At the end of the lecture, you will be able to:

(i) Define Regular Expressions and Regular Languages

(ii) Describe how Deterministic FA (DFA) and Non-deterministic FA (NFA) computes

(iii) Construct DFAs and NFAs using regular expressions.

Introduction:

Definition: Finite Automata

- ✓ A Finite Automaton is a recognizer used to identify the tokens occurring in an input stream.
- ✓ Simply, it is a machine with a finite number of states and a finite number of transitions between them[1]

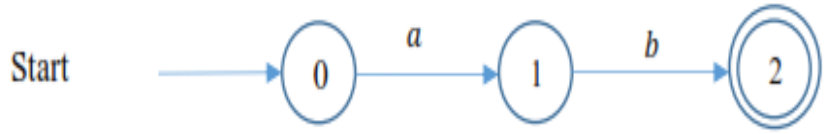
Representing Finite Automata[2]

- ✓ There is a distinctive start state where the machine starts.
- ✓ Between the states, there are transitions labelled by the alphabets of the language
- ✓ **Example;** if the machine is at state $\mathbf{S_i}$ and there is a transition labelled by alphabet $\boldsymbol{a_k} \in \Sigma$ to the state $\mathbf{S_j}$ then the machine can perform such a transition provided that the current symbol is $\boldsymbol{a_k}$.
- ✓ Starting from the **start state**, the machine finally reaches some state after exhausting all the input symbols.

[1] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 172

[2] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 172

- ✓ The **final state** reachable for all the input streams (tokens) is called the **acceptor/final state** e.g. the following is a finite automaton for the string ab^3.



- ✓ If the machine is in the state S_i and there does not exist any possible transition from S_i to the next input symbol then the input is rejected[4].

There are two types of Finite Automata[5]

A. ***Non-deterministic Finite Automata (NFA)***

- ✓ This is a finite automaton that has ability of more than one possible transition from a state on the same input symbol.
- ✓ The actual transition occurring is selected non-deterministically by the machine.
- ✓ Another non-determinism lies within the empty transitions (e-transition).
- ✓ If there is an e-transition from state S_i to state S_j then the machine can do such a transition without consuming any other input
- ✓ An input is said to be accepted/recognized by the automata if there exists at least one path from the start state of the machine to the final state whose transitions are governed by the input stream.

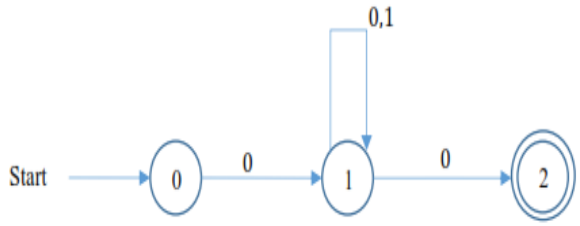
Examples:

Construct an NFA for the regular expression 0(0|1) *0

[3] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 173

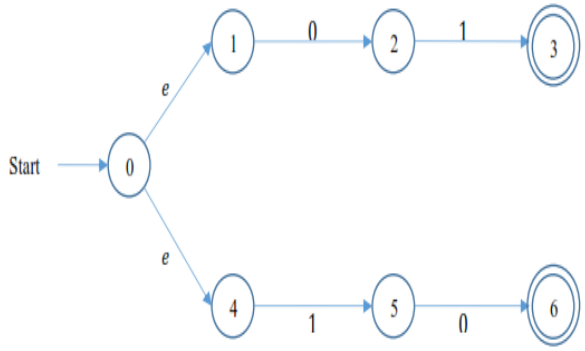
[4] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 174

[5] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 172



Note: from state 1, there are two transitions labelled by 0, one to state 1 itself (loop) and the other to state 2[6].

Construct NFA for the regular expression 01|10[7]



B. Deterministic Finite Automata (DFA)

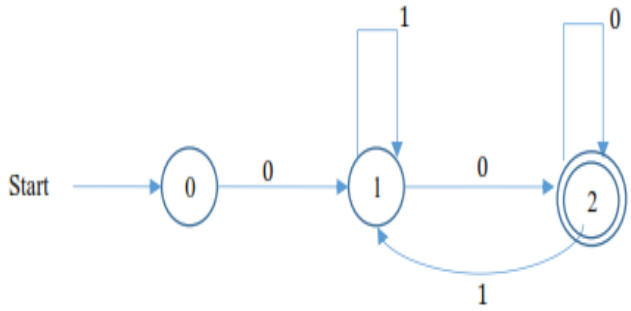
✓ This is a finite automaton that cannot have more than one transition emanating from the same state labelled by the same input symbol. Also, there cannot be any empty transitions[8]

Examples

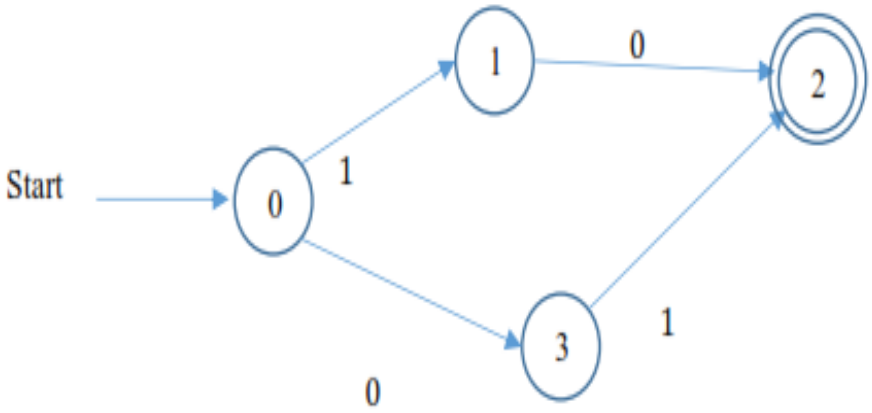
Construct a DFA for the following regular expressions[9]:

a) 0(0|1) *0

[6] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 172
[7] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 172
[8] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 174
[9] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 176



b) 01|10



Note: Every non-deterministic alternative of an NFA gets converted into an explicit path from start to final state in a DFA. This results in an explosion in the number of states in a DFA[10]

The following is an algorithm to test whether an input stream is accepted by a DFA or not:

Algorithm: DFA-test

Input: a DFA with a start state S_0.

An input stream

Output: "yes" is accepted, "no" otherwise[11].

[10] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 176

[11] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 175

Begin

S=S_0

While not end-of-input do

let c = next input symbol

if there is a transition on c from S to S_1 then

S=S_1

end while

If S is a final state and c=end-of-input then

return "yes"

else

return "no[12]*"*

Note: The existence of non-determinism makes the task of testing if an input stream is acceptable difficult. This is because it requires *trying all possible alternatives* that may occur in non-deterministic transition of states[13]

Converting Regular Expressions to NFA

- ✓ **Three steps are involved: -**

(i) Break down the regular expression into the simplest sub-expressions,

(ii) Construct the corresponding NFAs,

(iii) Combine these small NFAs guided by the operations of the regular expression[14].

- ✓ This construction is known as Thompson's construction rules; which are as follows: -

Thompson's construction rules…

i) For e, the NFA consists of two states (start and final state)[15] . The transition is labelled by (e).

[12] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 176

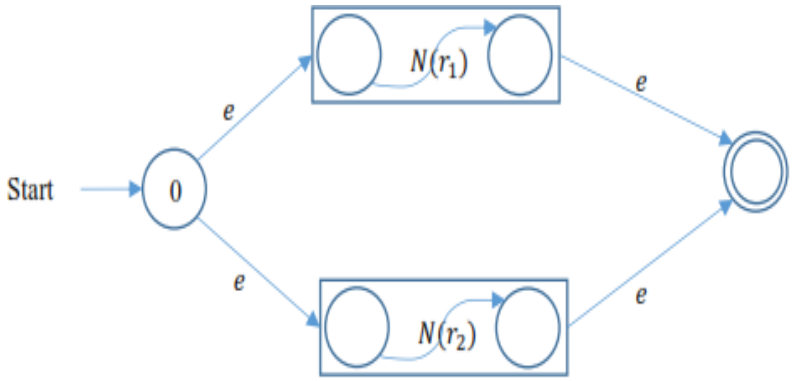
[13] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 177

[14] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 177

[15] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 184

ii) For any alphabet symbol 'a' in the alphabet set Σ, the NFA also consists of two states with transition labelled by a[16]

iii) For the regular expression $r_1|r_2$, if $N(r_1)$ is NFA for r_1 and $N(r_2)$ is NFA of r_2 then NFA of $N(r_1|r_2)$ is constructed as follows[17]:

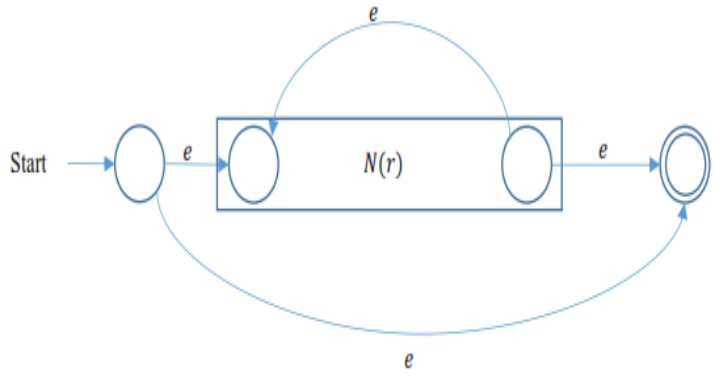


- ✓ This means that the e-transitions are introduced from the new start state to the start states of $N(r_1)$ and $N(r_2)$ and similarly from the final state of $N(r_1)$ and $N(r_2)$ to the newly created final state[18].

iv) For the regular expression r_1r_2, the NFA $N(r_1r_2)$ is constructed by merging the NFAs $N(r_1)$ and $N(r_2)$ i.e. the final state of $N(r_1)$ is merged with the start state of $N(r_2)$ as shown[19]: -

[16] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 184
[17] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 185
[18] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 185
[19] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 160

v) For the regular expression r*, N(r*) is constructed as follows[20]: -



This means that e-transitions from the new start state to the new final state correspond to zero occurrences of r, whereas, from the final state to the initial state of N(r) corresponds to the repeated occurrence of r[21]

vi) If **N(r)** is NFA for a regular expression, it is also NFA for the parenthesized expression **(r)**[22].

Example: Use Thompson construction rules to come up with the NFA of the following regular expression a**(a|b) * *ab***

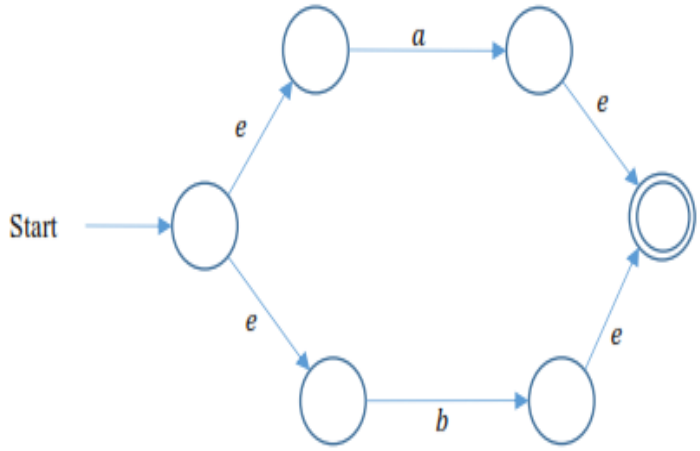
The sub-expressions are[23]: -

1. **a**

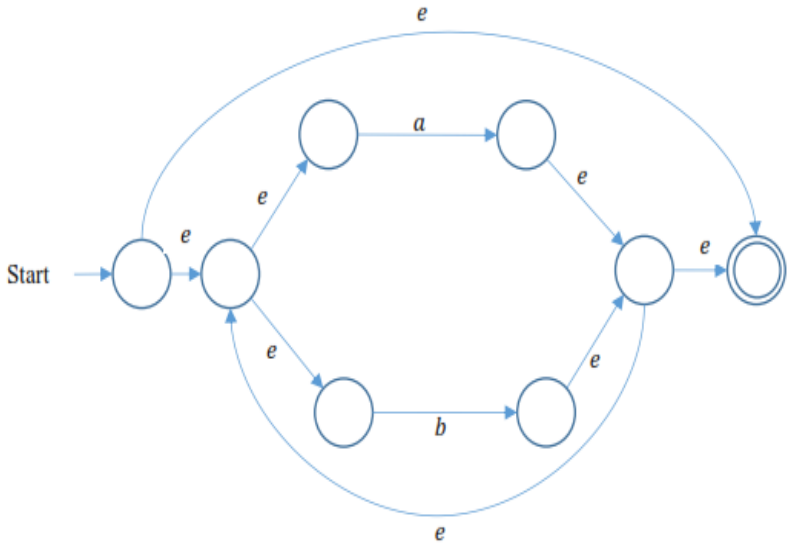
2. **b**

[20] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 186
[21] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 186
[22] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 159
[23] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 160

3. **a|b**[24]



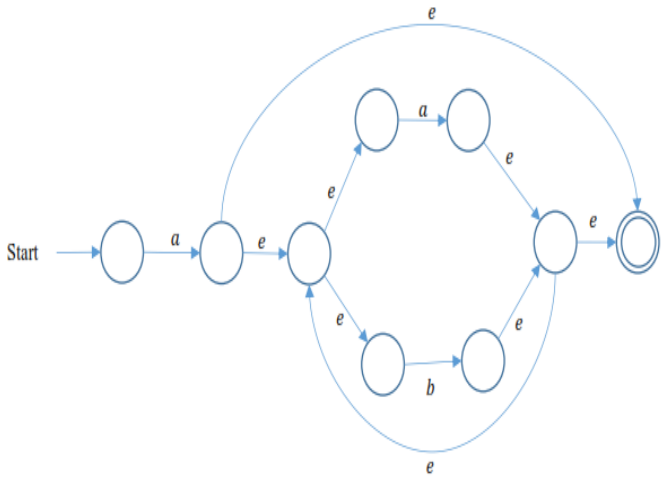
4. **(a|b)***



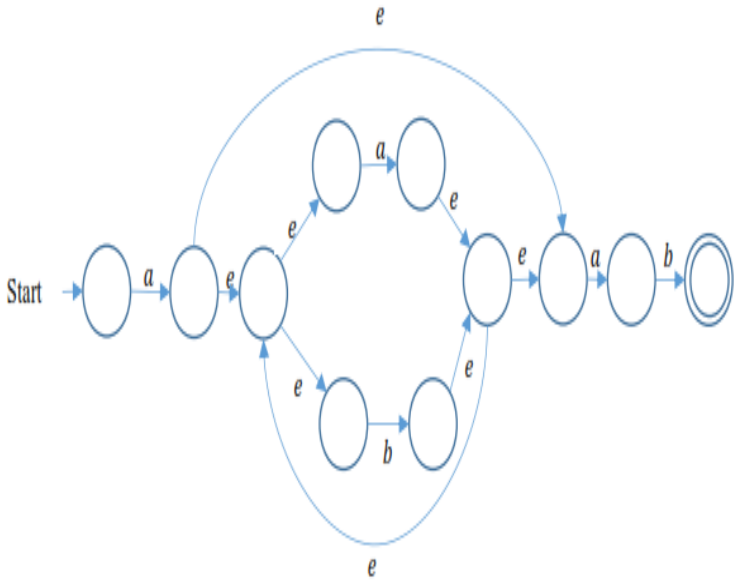
5. **a(a|b)*** [25]

[24] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 184

[25] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 188



6. a(a|b)*ab[26]



[26] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 188

Content Covered in Week 3: Finite Automata

At the end of the lecture, we were able to:

(i) Define regular expressions and regular Languages

(ii) Describe how DFA and NFA computes

(iii) Construct DFAs and NFAs using regular expressions.

Course Text Books

1. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007))

2. Compiler Construction: Principles and practices; Kenneth C. Louden; Cengage Learning; 1st edition, ISBN-10: 0534939724 (1997)

3. Basics of Compiler Design: Torben Mogensen; DIKU University of Copenhagen Universitetsparken 1 DK-2100 Copenhagen DENMARK (2007)

4. Engineering a Compiler: 2nd edition; Keith D. Cooper and Linda Torczon; Morgan Kaufmann Publishers ISBN: 978-0-12-088478-0 (2003)

5. Compiler Design: Santanu Chattopadhyay; PHI Learning Publishers, ISBN 812032725X, 9788120327252 (2005)