- *Lexical Analyzer reads the source program character by character to produce tokens.*
- *Normally a lexical analyzer doesn't return a list of tokens at one shot; it returns a token when the parser asks a token from it.*

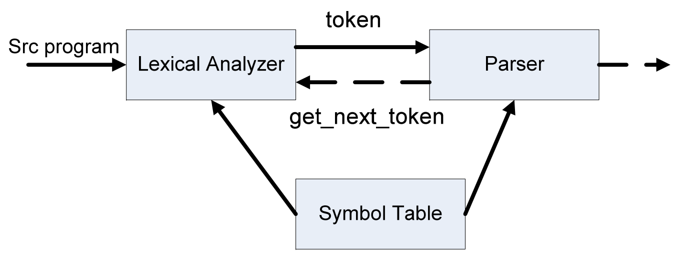


Fig : Interaction of Lexical Analyzer with Parser.

Two Phases :

⊗ Scanning [Removal of White/Blank Spaces]
⊗ Lexical Analysis [Token Identification etc.]

Issues :

⊗ Simpler design is perhaps the most important consideration. The separation of lexical analysis often allows us to simplify one or other of these phases.

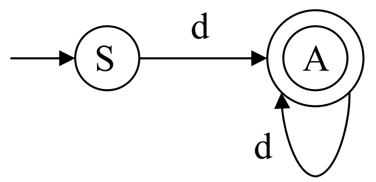
e.g. : Removal of White/Blank spaces in Lexical Analysis

⊗ Compiler efficiency is improved. A separate Lexical Analyzer allows us to construct a specialized and potentially more efficient processor for the task. A large amount of time is spent reading the source program and partitioning it into tokens. Specialized buffering techniques for reading input characters and processing tokens can significantly speed up the performance.

⊗ Compiler portability enhanced. Input alphabet peculiarities and other device-specific anomalies can be restricted to the Lexical Analyzer.

Finite Automata :

It is a mathematical model in which a system is represented. It consists of some discrete states and the system is in one of the state at any point of time. The System moves from one state another.
e.g.:



Where, d → digit

Transition diagrams are specialized flowcharts to represent Finite Automata.

○	: State
◎	: Final/Halt State
→	: Transition

Mathematical Representation

A Finite Automata can be represented as a five touple:

$$M = (Q, \Sigma, \delta, q_0, F)$$

Where, Q : Set of states
Σ : Set of Input Valid String
δ : Set of Transition Function
q_0: Start State
F : Set of Final States

Machine Equivalence :

Two FSMs M and M' are said to be equivalent; if every string recognized by M is also recognized by M'.

Isomorphic FSMs

Two FSMs M and M' are said to be isomorphic if M can be obtained by relabeling M' and vice–versa.

A finite automaton can be: Deterministic(DFA) or Non-deterministic (NFA)

Deterministic Finite Automata(DFA):

A Deterministic Finite Automata(DFA) consists of :

⊗ A finite set of states, often dented by Q.

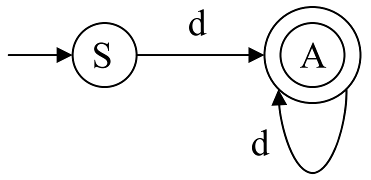
⊗ A finite set of input symbols, often denoted by Σ.

⊗ A transition function that takes a state and an input symbol as arguments and returns a state, often denoted by δ.

⊗ A start state, one of the state in Q, often denoted by q_0.

⊗ A set of final or accepting or halt states, often denoted by F. The set F is a subset of Q.

e.g. M = ({S,A}, {d}, δ, S, {A})



Simpler Notations:

⊗ Transition Diagram: A transition diagram for a DFA, M = (Q, Σ, δ, q_0, F) is a graph defined as follows :

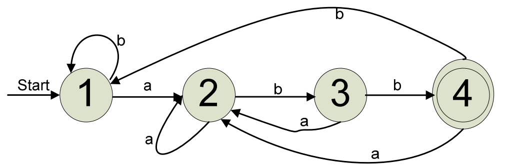
(a) For each state in 'Q', there is a node.

(b) For each state *'q'* in Q and each input symbol *'a'* in Σ, Let *δ(q,a) = p*. Then, the transition diagram has an arc from node *'q'* to *'p'*, labeled *'a'*. If there are several input symbols that cause transitions from *'q'* to *'p'*, then the transition diagram can have one arc, labeled by the list of these symbols.

(c) There is an arrow into the start state *'q0'* labeled *'Start'*. This arrow does not originate at any node.

(d) Nodes corresponding to accepting states (those in F) are marked by a double circle. States not in F have a single circle.

e.g.: DFA accepting (a|b)*abb.



⊗ Transition Table : A transition table is a conventional, tabular representation of a function *'δ'* that takes two arguments and returns a value. The rows of the table correspond to the states, and the columns correspond to the inputs. The entry for the row corresponding to state *'q'* and the column corresponding to input *'a'* is the state *δ(q,a)*.

e.g. : Transition table to the function *'δ'* of above Transition Diagram.

	a	b
→1	**2**	1
2	**2**	3
3	**2**	4
*4	2	1

Non-Deterministic Finite Automata(NFA) :

A Non-deterministic Finite Automata(NFA) is represented essentially like DFA and consists of :

⊗ A finite set of states, often dented by Q.

⊗ A finite set of input symbols, often denoted by Σ.

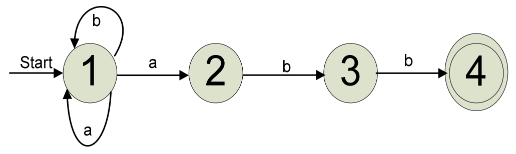
⊗ A transition function that takes a state and an input symbol as arguments and returns a set of state, often denoted by δ.

⊗ A start state, one of the state in Q, often denoted by q_0.

⊗ A set of final or accepting or halt states, often denoted by F. The set F is a subset of Q.

e.g. NFA accepting (a|b)*abb.

M = ({1,2,3,4}, {a,b}, δ, 1, {3})

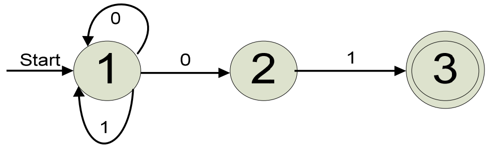


Transition Diagram & Table :

	a	b
→1	**{1,2}**	{1}
2	ϕ	{3}
3	ϕ	{4}
*4	ϕ	ϕ

Conversion of NFA to DFA : (Sub-Set Construction)

e.g. 1. An NFA accepting all strings that end in *01*.



Complete Subsets:

	0	1
ϕ	ϕ	ϕ
→1	**{1,2}**	{1}
2	ϕ	{3}
*3	ϕ	ϕ
{1,2}	**{1,2}**	{1,3}
*{1,3}	**{1,2}**	{1}
*{2,3}	ϕ	{3}
*{1,2,3}	{1,2}	{1,3}

Formulation:

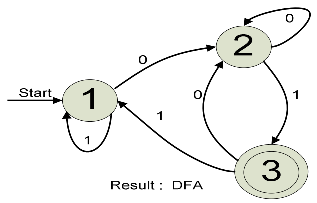
Inputs = 2

States = 3

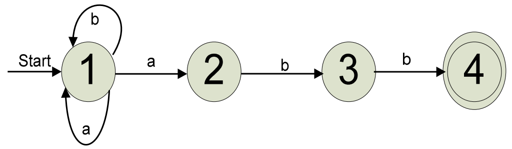
Number of Sub Sets : $2^3 = 8$

Preferred Transition Table :

	0	1
→1	**{1,2}**	{1}
2	ϕ	{3}
*3	ϕ	ϕ
{1,2}	**{1,2}**	{1,3}
*{1,3}	{1,2}	{1}



e.g. 2 : NFA accepting (a|b)*abb.



Transition Table:

	a	b
→1	**{1,2}**	{1}
2	ϕ	{3}
3	ϕ	{4}
*4	ϕ	ϕ
{1,2}	**{1,2}**	{1,3}
{1,3}	**{1,2}**	{1,4}
{1,4}	{1,2}	{1}

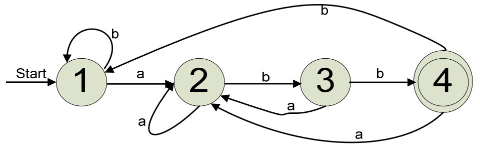
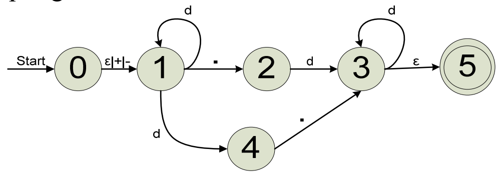


Fig.: Final DFA

Finite Automata with Epsilon-Transition :

It is the extension of finite automata; with a transition on ε, the empty string, an NFA is allowed to make a transition spontaneously, without receiving an input symbol.

e.g. : An ε-NFA accepting decimal numbers.



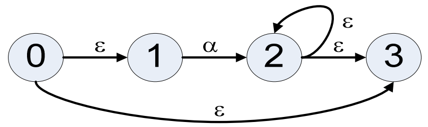
Where d = 0|1|.....|9|

Algorithm for finding ε-Closure(s)

→ δ is added to ε-closure(s)

→ If f is in ε-closure(s), and there is an edge labeled ε from f to u, then u is also added to ε-closure(s). This step is repeated till not state is left which can be added to ε-closure(s).

Example : Find ε-closure(0) for the following NFA.



Solution : ε-closure(0) = {0, 1, 2}

Subset Construction Algorithm :

1. A = ε-closure(0), i.e. start state.
2. For each input symbol $a \in \Sigma$ do.

Let T be set of states to which there is a transition a from states in A.

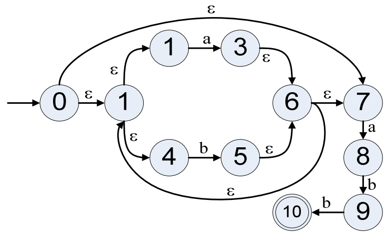
(Look for the states in A on which there is a transition on input symbol, say a, b. Then T_a = *{set of states to which transition made on a}*. T_b = *{set of states to which transition is made on b}*)

$B = \varepsilon\text{-closure}(T_a)$
$C = \varepsilon\text{-closure}(T_b)$

Mark A as its transition state has been found.

3. Repeat the step 2 with newly found states i.e. B and C and this process will be repeated till all the newly found states are marked and there is at least one state which contains final state of NFA.

Example : Find the DFA for following ε-NFA.



$\Sigma = \{a, b\}$

Now applying the algorithm,

$A = \varepsilon\text{-closure}(0) = \{0, 1, 2, 4, 7\}$

Here, There is transition on input symbol a, b on $2, 4, 7$ to $3, 5, 8$.

Thus, $T_a = \{3, 8\}$

$T_b = \{5\}$

Hence, $B = \varepsilon\text{-closure}(T_a)$

$C = \varepsilon\text{-closure}(T_b)$

Thus, we have found transition for A i.e. to B on a, and to C on b.

Now, Repeat the process for B,

$B = \varepsilon\text{-closure}(T_a) = \varepsilon\text{-closure}\{3,8\} = \varepsilon\text{-closure}(3) \cup \varepsilon\text{-closure}(8)$

$= \{1, 2, 3, 4, 6, 7\} \cup \{8\} = \{1, 2, 3, 4, 6, 7, 8\}$

$C = \varepsilon\text{-}closure(T_b) = \varepsilon\text{-}closure\{5\} = \{1, 2, 4, 5, 6, 7\}$

Let *D'* be the set of marked states. Thus add *A* to *D'* i.e. *D' = {A}*

For B,

$B = \{1, 2, 3, 4, 6, 7, 8\}$

$T_a = \{3, 8\}$

$T_b = \{5, 9\}$

$\varepsilon\text{-}closure(T_a) = \varepsilon\text{-}closure\{3, 8\} = B$

$\varepsilon\text{-}closure(T_b) = \varepsilon\text{-}closure\{5, 9\} = \{1, 2, 4, 5, 6, 7, 9\} = D$

Thus, we have found transition for *B* to *B* on *a* and *B* to *D* on *b*.

Hence, *D' = {A, B, …}* i.e. *B* is marked.

For C,

$C = \{1, 2, 4, 5, 6, 7\}$

$T_a = \{3, 8\}$ *(from states 2, 7 on a)*

$T_b = \{5\}$ *(from state 4 on b)*

$\varepsilon\text{-}closure(T_a) = \varepsilon\text{-}closure\{3, 8\} = B$

$\varepsilon\text{-}closure(T_b) = \varepsilon\text{-}closure\{5\} = C$

Thus, transition from *C* to *B* on *a* and *C* to *C* on *b* have found. *C* is marked. Hence, *D' = {A, B, C, …}*

For D,

$D = \{1, 2, 4, 5, 6, 7, 9\}$

$T_a = \{3, 8\}$

$T_b = \{5, 10\}$

$\varepsilon\text{-}closure(T_a) = \varepsilon\text{-}closure\{3, 8\} = B$

$\varepsilon\text{-}closure(T_b) = \varepsilon\text{-}closure\{5, 10\} = \{1, 2, 4, 5, 6, 7, 10\} = E$

Thus, we have found transitions for *D* to *B* on *a* and *D* to *E* on *b*.

Hence, *D* is included in marked state, i.e. *D' = {A, B, C, D, …}*

For E,

$E = \{1, 2, 4, 5, 6, 7, 10\}$

$T_a = \{3, 8\}$

$T_b = \{5\}$

$\varepsilon\text{-}closure(T_a) = \varepsilon\text{-}closure\{3, 8\} = B$

$\varepsilon\text{-}closure(T_b) = \varepsilon\text{-}closure\{5\} = C$

Thus, we have found transitions for *E* to *B* on *a* and *E* to *C* on *b*.

Hence, *E* is included in marked state, i.e. *D' = {A, B, C, D, E}*

Thus, there is now new state and all the states are marked and also E is final state, as it contains the final state 10 of NFA.

Thus, final DFA is obtained by combining all the states of DFA.

Transition Table is given as,

δ	*a*	*b*
→A	B	**C**
B	B	**D**
C	B	**C**
D	B	**E**
***E**	**B**	**C**

$\Sigma = \{a, b\}$

$Q = \{A, B, C, D, E\}$

$q_0 = A$

$F = \{E\}$

$\delta \leftarrow$ *Shown above.*

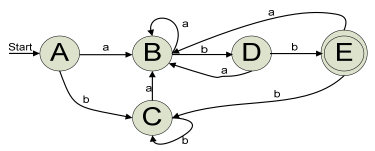


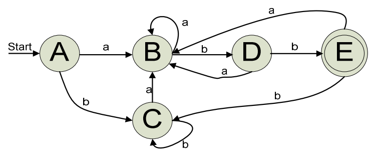
Fig. DFA obtained from above NFA.

<u>Minimization of states :</u>

1. Separate the stares into group of halt and non-halt states.
2. Take any two states in a two sup-groups and test whether they are same with respect to the transition function. If so, merge them into one, otherwise separate them into different sub-groups.

Then, draw the final Transition Diagram.

e.g. : DFA accepting (a|b)*abb



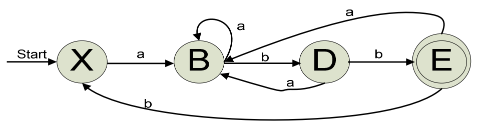
Minimization:

Transition Table

δ	a	b
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C

1. Separation : {A,B,C,D}, {E}
2. {A,C},{B,D},{E}
3. {X}, {B},{D}

Final DFA :



Removal of Inaccessible States :

1. Mark the start state.
2. Starting from a marked state, mark all the stares for which there exists a transition.
3. Repeat 2 until no more new marked states are added.
4. Remove all the unmarked state as they are inaccessible states.

Removal of Empty-Cycles :

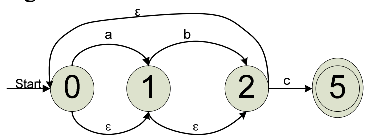
An empty cycles is detected if the system can reach the same state following a set of empty moves only.

Algorithm: Detection & Removal of empty cycles.

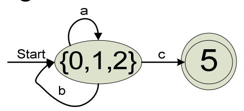
(a) Mark all the states.
(b) Construct a tree with any marked states as root node.
(c) A child node to a parent node is that node for which there exists an empty transition from the parent node.

(d) If the root node reappears, an empty cycle is detected and the path form the root up to the leaf node denotes the empty cycle.
(e) - Merge all the nodes in the path with the root node.
- Unmark the root state(node).
(f) Repeat steps (a) to (e) until all the states are unmarked.

e.g. :



Applying the Algorithm:



Removal of Empty-Transition :

For all the transition function $\delta(x,\varepsilon) = y$, the empty transition can be removed by introducing new transition function,

$$\delta(x,*) = z \text{ for all } \delta(y,*) = z$$

While removing empty transitions, on superimposing the halt or stop state over any other state make that state also a halt state.

i.e. For all $\delta(x,\varepsilon) = y$, make x as halt state if y is a halt state.

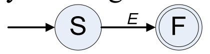
e.g. : Transition table of ε-NFA; accepting decimal numbers.

Algorithm for Regular Expression to FA:

Let E is a Regular Expression.

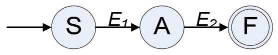
1. Create a start state and a final state.

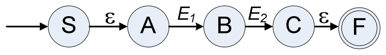
2. Try to recognize E.



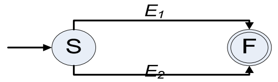
3. If $E = E_1.E_2$,

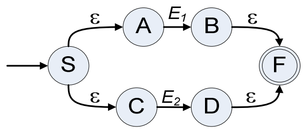
- Create a new state (Let *A*)
- Establish transition from *S* to *A* on E_1 and from *A* to *F* on E_2.

i.e. : 

or, 

4. If $E = E_1|E_2$,
   - Remove the transition on *E* to establish two transitions between *S* and *F* on E_1 and E_2.

i.e. : 

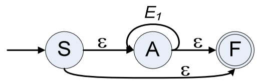
or, 

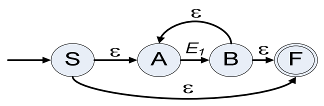
5. If $E = E_1*$,
   - Create a new state,
   - Establish

$\delta(S,\varepsilon) = A$

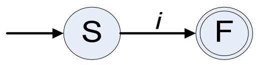
$\delta(A,E_1) = A$

$\delta(A,\varepsilon) = F$

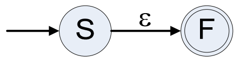
i.e. 

or, 

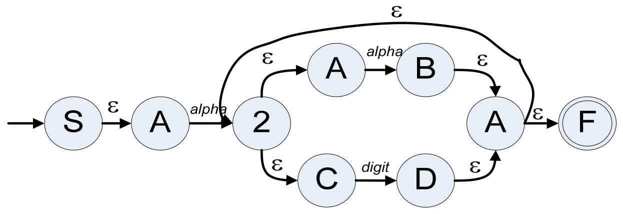
6. If *E* is an input symbol *(i)*, then Replace *E* by *i*.

i.e. 

7. If E = ε then,



Example : *id = alpha.(alpha|digit)**



Construction of Lexical Analysis

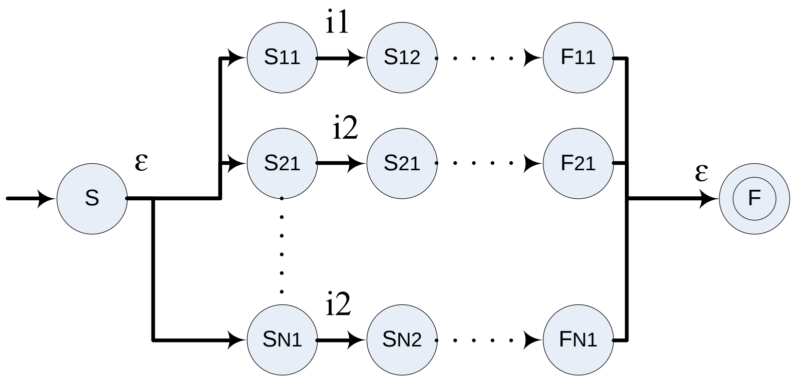


Fig. NDFSM for N-machine

Algorithm

1. Construct an NDFSM for N-Tokens.
2. Create a new Start state and a Final state.
3. Connect newly created Start and Final states to individual NDFSM's Start and Final States with empty moves.
4. Convert resultant NDFSM to DFSM and minimize its states if necessary.
5. Write a program for resultant DFSM which is your final program for lexical analyzer.