NEW YORK UNIVERSITY

CSCI-GA.2130-001

Compiler Construction

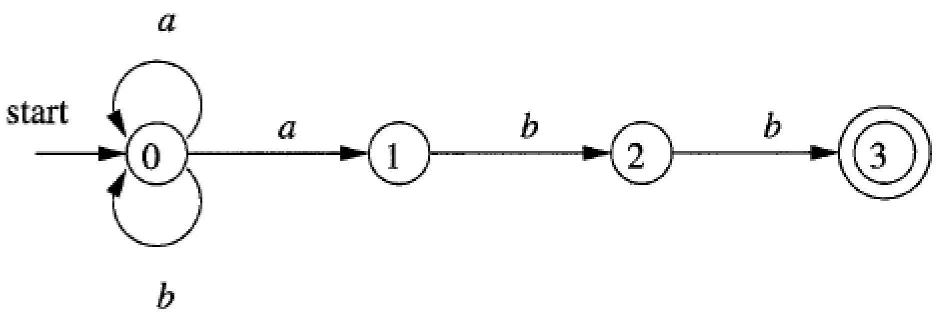
Lecture 5:
Lexical Analysis II

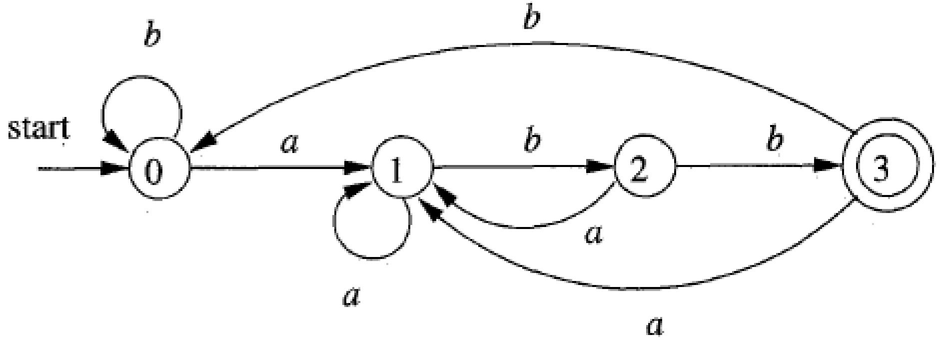
Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu

The Magic Behind It All: Finite Automata

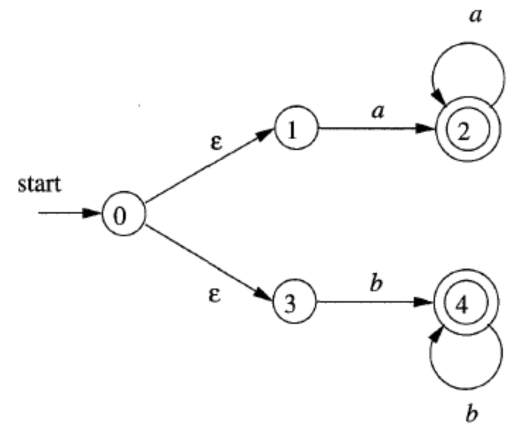
- Recognizers: "yes" or "no" about each input string
- Two Flavors:
  - Non-deterministic Finite Automata (NFA)
  - Deterministic Finite Automata (DFA)
- Main parts
  - States
    - Start
    - Accepting or final
  - transitions

Which is Which?



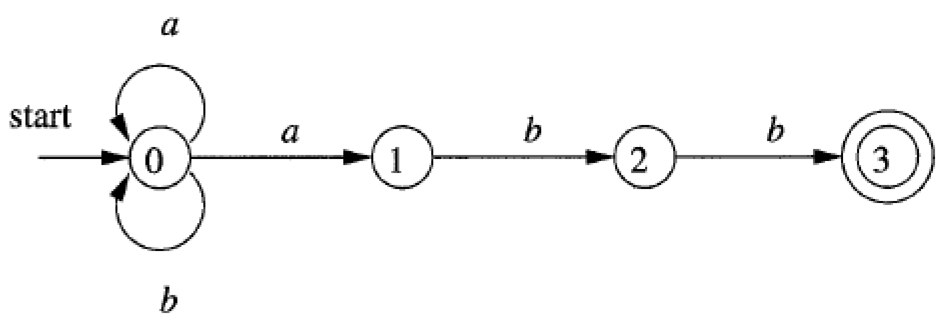


NFA



- Finite set of states S
- Input alphabet Σ
- transition function that gives for each state and for each $\Sigma \cup \{\varepsilon\}$ a set of next states
- A starting state S_0
- A set of accepting or final state(s)

Another Presentation of NFA: Transition Tables



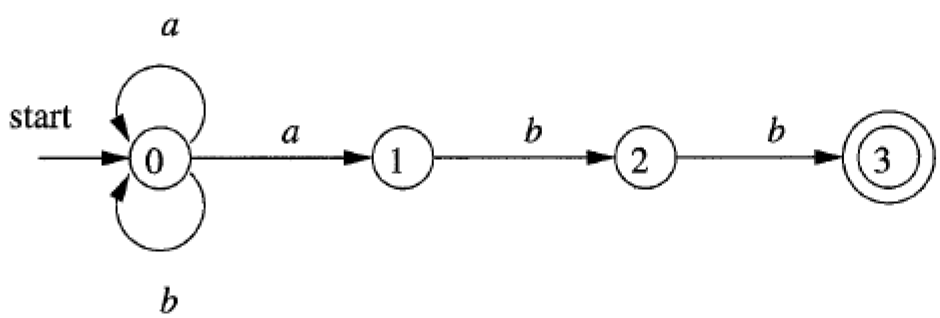
STATE	a	b	ϵ
0	$\{0, 1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$

+ We can easily find the transition
- Lot of space

Acceptance of Input String

Input string x is accepted if and only if:

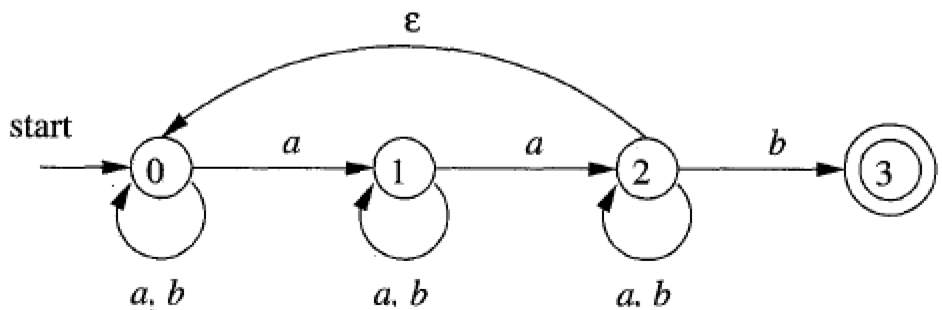
There is some path in the transition graph from start to one of the accepting states



Which of the following are accepted: abb, *aaa, aabb, aaabb, bbb* ?

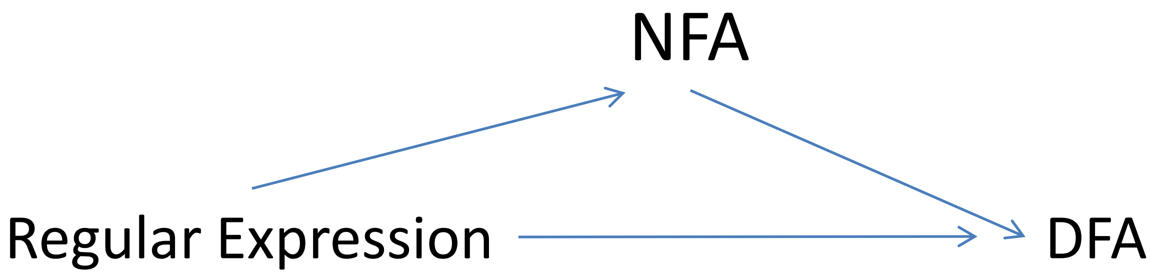
Example

- For the following NFA indicate all paths labeled *aabb*

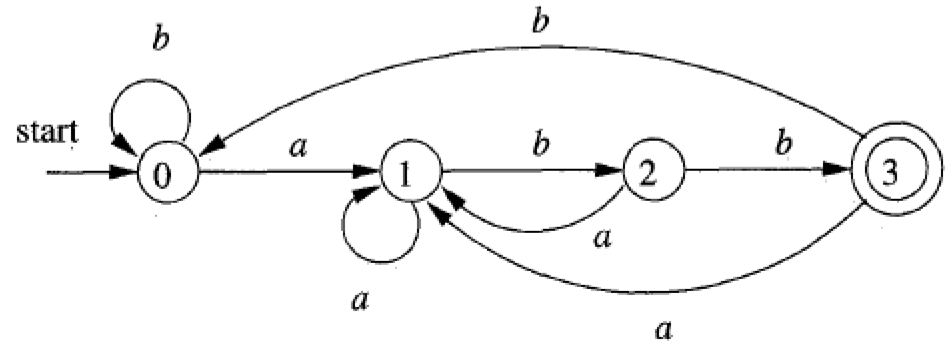


DFA

- Special case of NFA
- No moves on ε
- For each state S, and input symbol a, there is exactly one edge out of s labeled a



```
s = s0;
c = nextChar();
while ( c != eof ) {
        s = move(s, c);
        c = nextChar();
}
if ( s is in F ) return "yes";
else return "no";
```



“Yes” or “No”?

abba

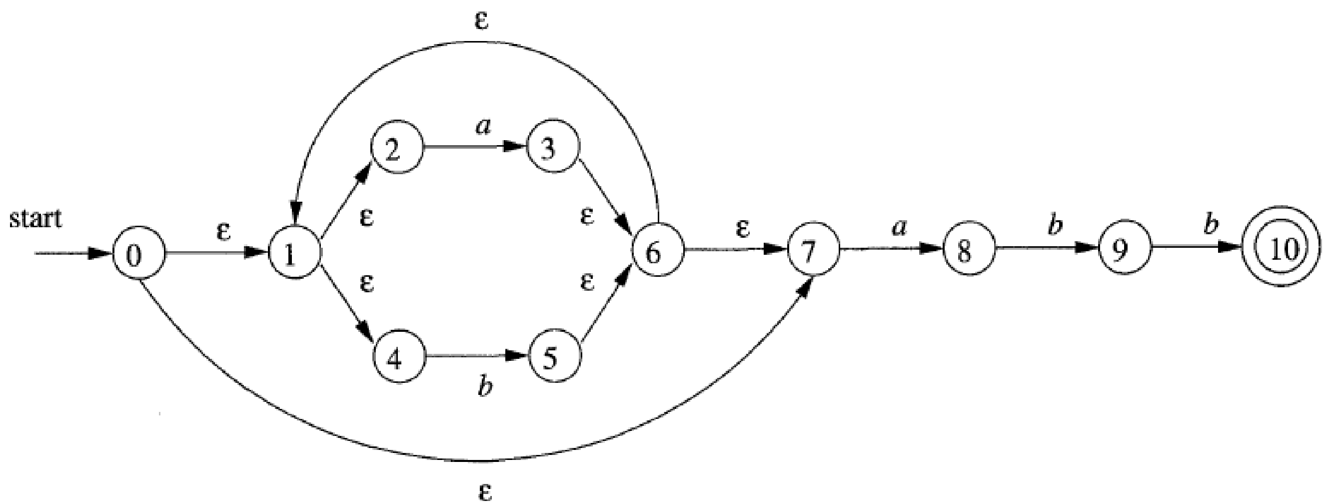
babb

aababb

abbb

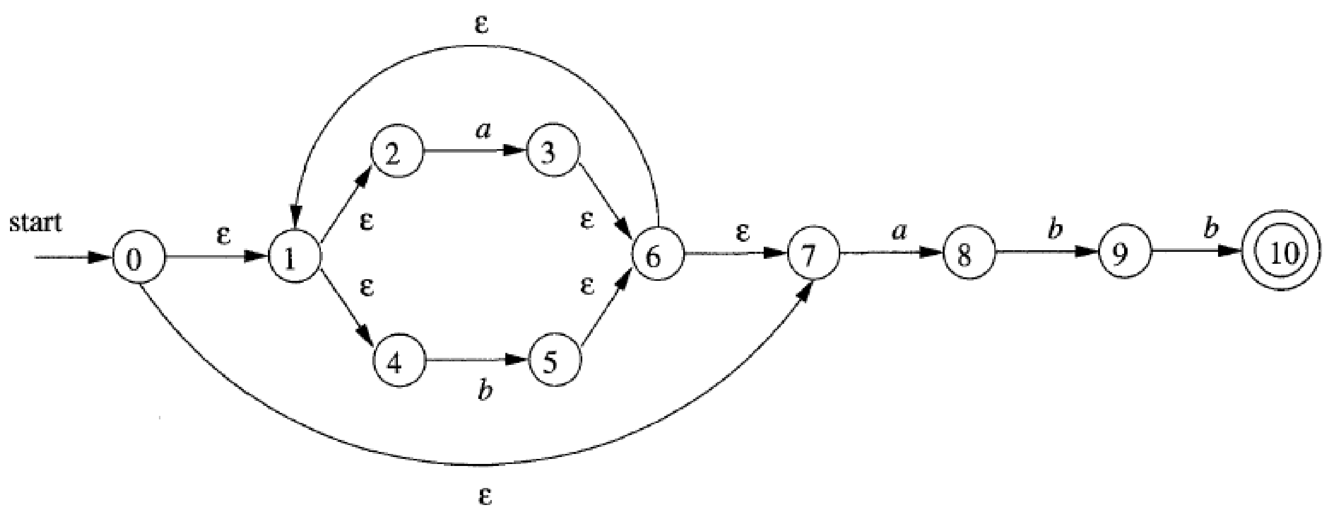
Some Definitions

OPERATION	DESCRIPTION
ϵ-*closure*(s)	Set of NFA states reachable from NFA state s on ϵ-transitions alone.
ϵ-*closure*(T)	Set of NFA states reachable from some NFA state s in set T on ϵ-transitions alone; $= \cup_{s \text{ in } T}\ \epsilon\text{-}closure(s)$.
$move(T, a)$	Set of NFA states to which there is a transition on input symbol a from some state s in T.



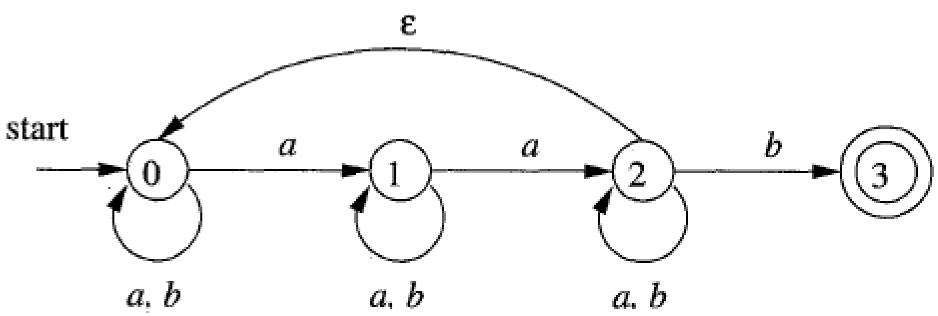
Simulating NFA

```
S = ε-closure(s0);
c = nextChar();
while ( c != eof ) {
        S = ε-closure(move(S, c));
        c = nextChar();
}
if ( S ∩ F != ∅ ) return "yes";
else return "no";
```



Example

Simulate the following NFA on *aabb*




```
S = ε-closure(s0);
c = nextChar();
while ( c != eof ) {
        S = ε-closure(move(S, c));
        c = nextChar();
}
if ( S ∩ F != ∅ ) return "yes";
else return "no";
```

What is the transition table of the above NFA?

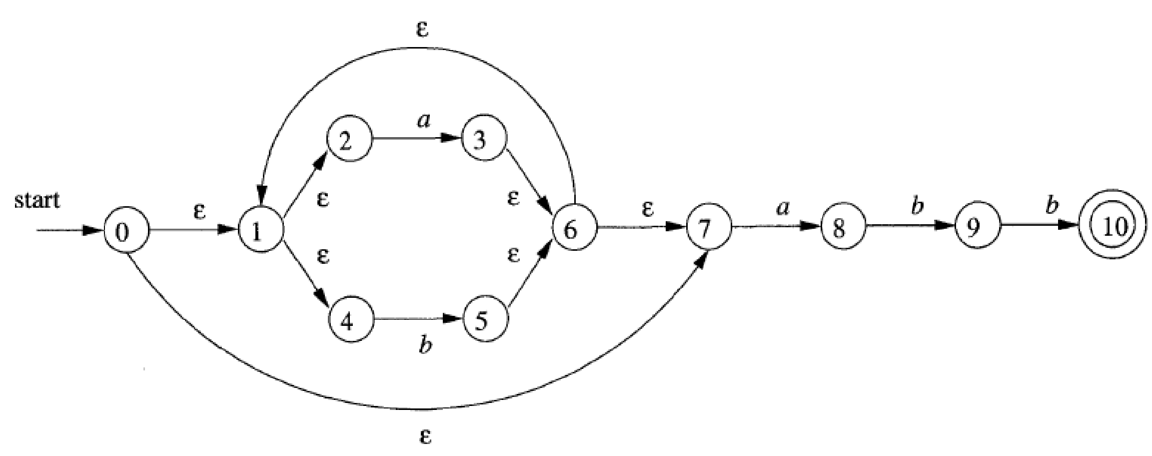
NFA -> DFA

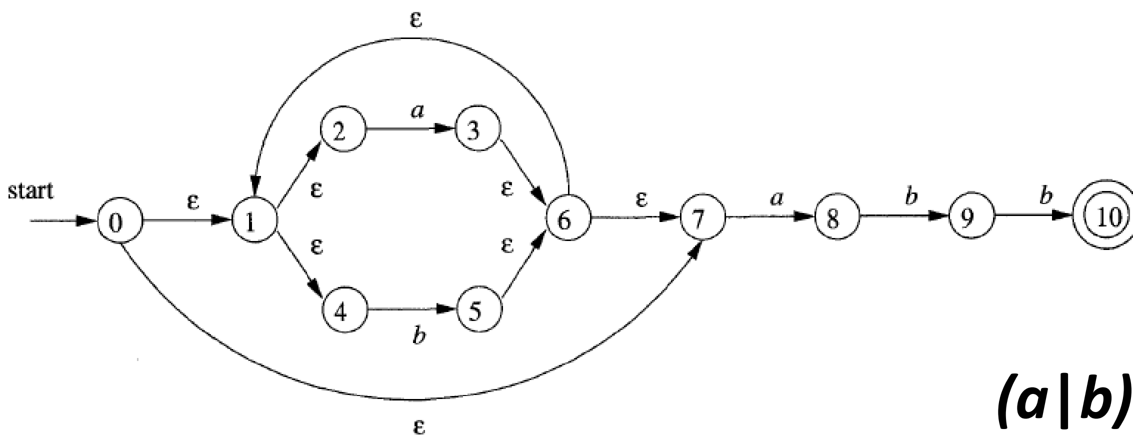
- Subset construction: each state of DFA corresponds to a set of NFA states
- For real languages NFA and DFA have approximately the same number of states (although theory has another opinion!)

Subset Constructions

initially, ϵ-*closure*(s_0) is the only state in *Dstates*, and it is unmarked;
while (there is an unmarked state T in *Dstates*) {
    mark T;
    for (each input symbol a) {
        $U = \epsilon\text{-}closure(move(T, a))$;
        if (U is not in *Dstates*)
            add U as an unmarked state to *Dstates*;
        $Dtran[T, a] = U$;
    }
}

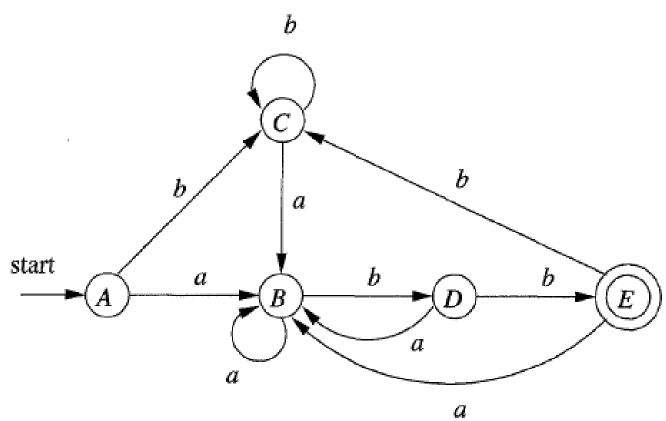
States of
the DFA
we are constructing



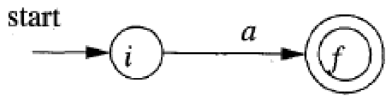


(a|b)*abb

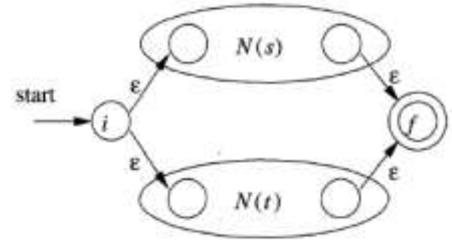
NFA STATE	DFA STATE	a	b
$\{0,1,2,4,7\}$	A	B	C
$\{1,2,3,4,6,7,8\}$	B	B	D
$\{1,2,4,5,6,7\}$	C	B	C
$\{1,2,4,5,6,7,9\}$	D	B	E
$\{1,2,3,5,6,7,10\}$	E	B	C



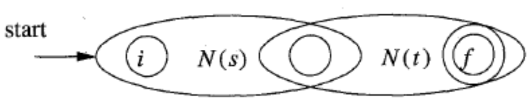
Regular Expression -> NFA
(McNaughton-Yamada-Thompson algorithm)



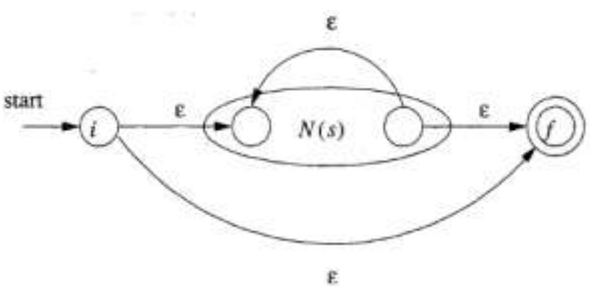
r = a



r = s|t



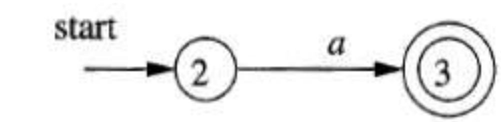
r = st

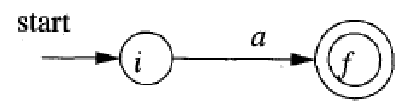


r = s*

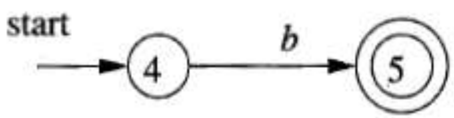
Example: (a|b)*abb

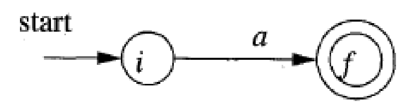
a



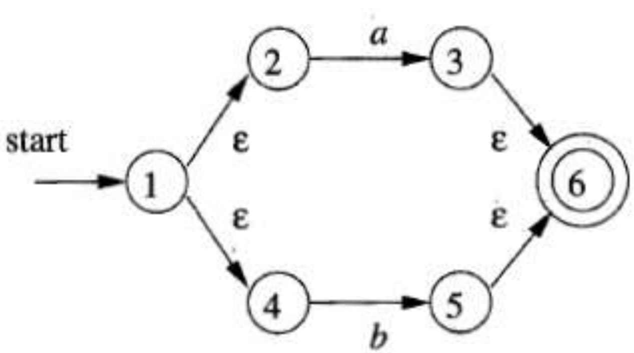


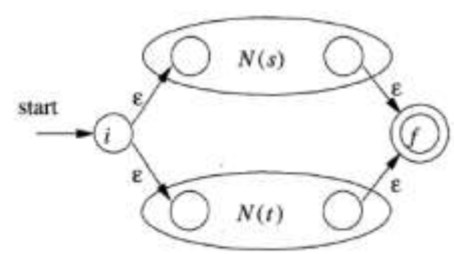
b





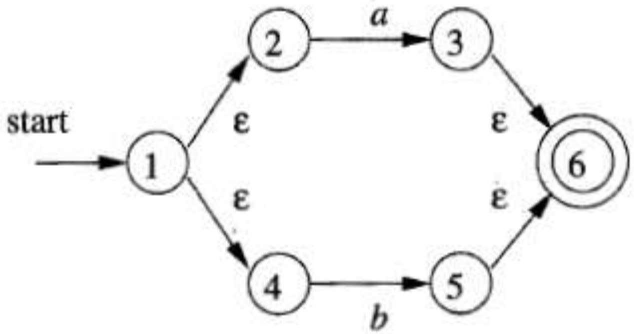
a | b

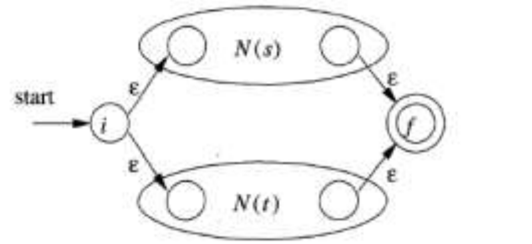




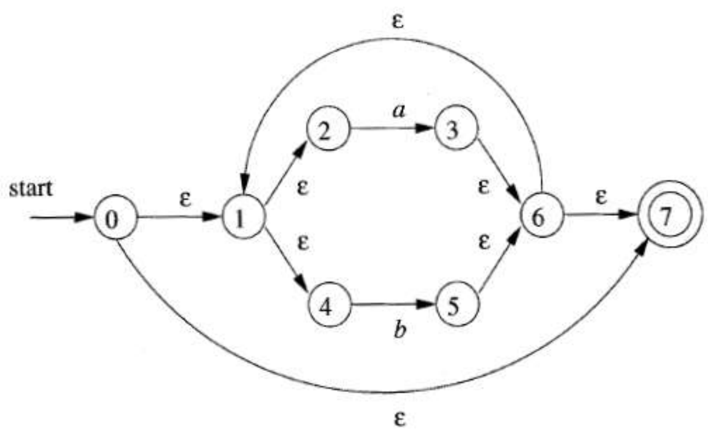
Example: (a|b)*abb

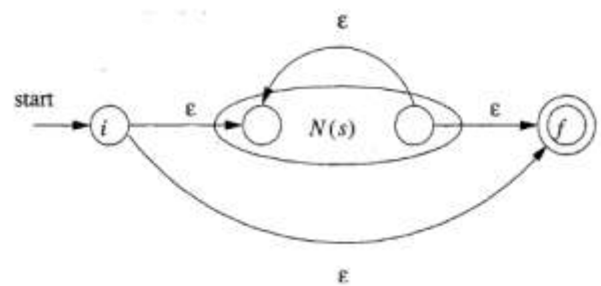
a|b



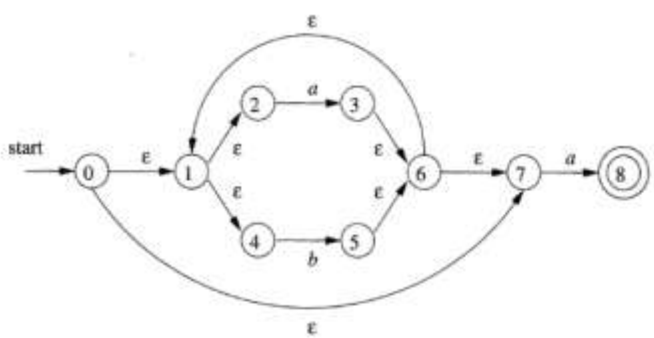


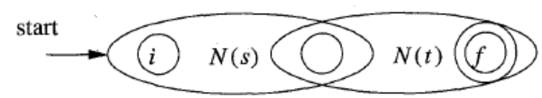
*(a|b)**





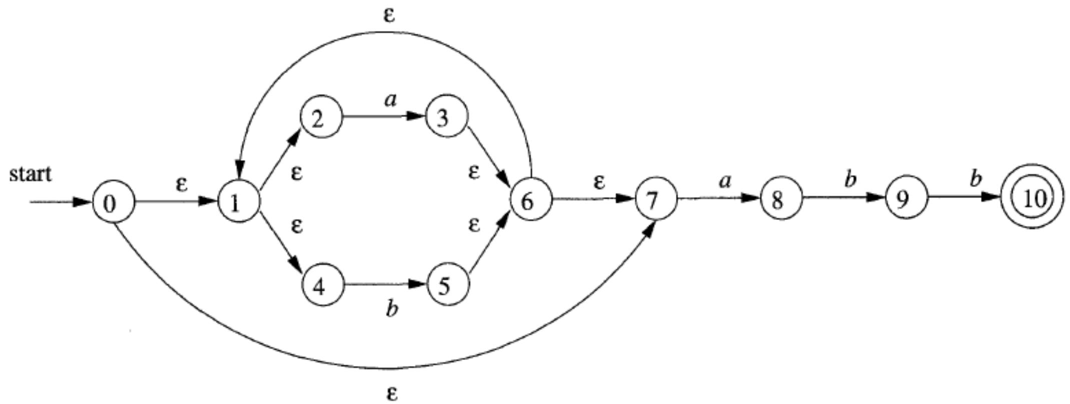
*(a|b)*a*





Example: (a|b)*abb

*(a|b)*abb*



State Minimization of DFA

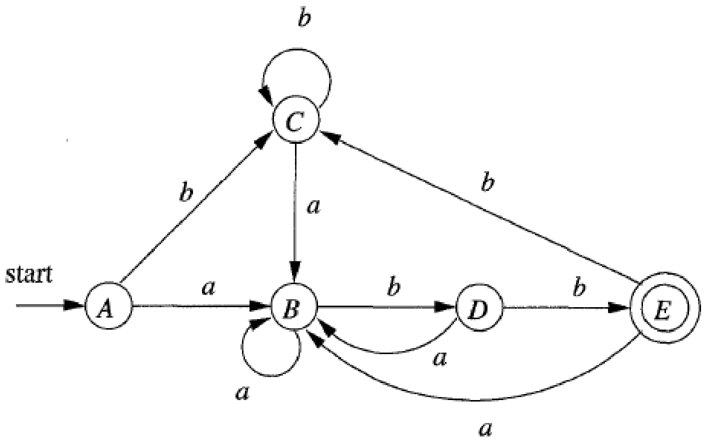
- There can be many DFAs that recognize the same language.
- Smaller DFAs are more efficient (storage, speed)
- There is always a unique minimum state DFA
- This minimum-state DFA can be constructed from any DFA that recognizes the language.

How to Do It?

1. Given DFA: start with at least two subgroups: S and S-F
2. Repeat the following algorithm until no more progress can be made

initially, let $\Pi_{\text{new}} = \Pi$;
for (each group G of Π) {
    partition G into subgroups such that two states s and t
        are in the same subgroup if and only if for all
        input symbols a, states s and t have transitions on a
        to states in the same group of Π;
    /* at worst, a state will be in a subgroup by itself */
    replace G in Π_{new} by the set of all subgroups formed;
}

Example



{A,B,C,D} {E}

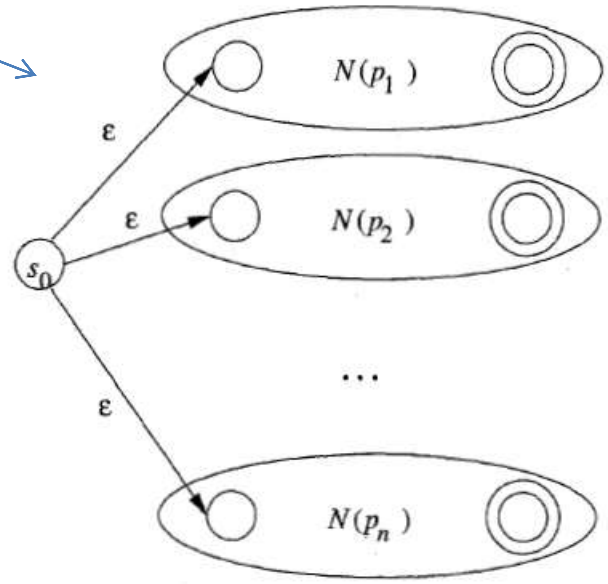
{A,B,C} {D} {E}

{A,C} {B} {D} {E}

STATE	a	b
A	B	A
B	B	D
D	B	E
E	B	A

Lexical Analyzer Generators

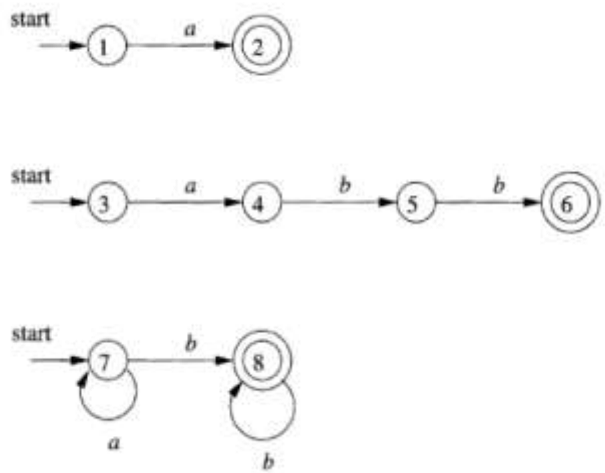
- Each regular expression -> NFA
- Combine all NFAs as
- In case of several matches
  - Pick longest
  - Pick earliest in file

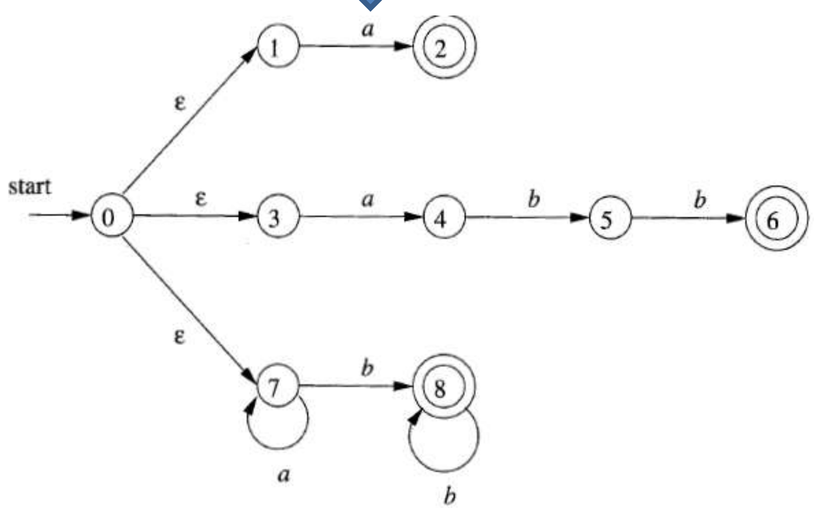


$\mathbf{a}$ $\quad$ { action A_1 for pattern p_1 }

$\mathbf{abb}$ $\quad$ { action A_2 for pattern p_2 }

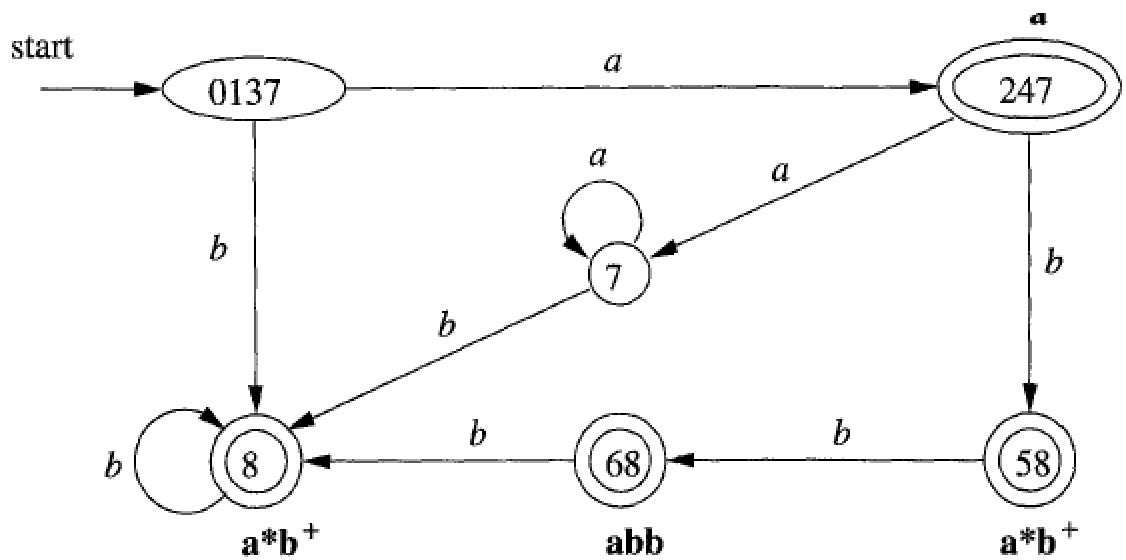
$\mathbf{a^*b^+}$ $\quad$ { action A_3 for pattern p_3 }





Lex

- Based on DFA not NFA
- Handling lookahead
- For state minimization, initial partition:
  - groups all states that recognizes a particular token
  - places in one group those states that do not indicate any token


start
0137
a
247
a
a
7
b
b
b
b
b
8
b
68
b
58
a*b+
abb
a*b+

So

- We have covered Sections 3.6 -> 3.9
- Skim: 3.7.3, 3.7.5, 3.9.1->3.9.5 and 3.9.8
- Read carefully the rest of: 3.6, 3.7, 3.8, 3.9.6, and 3.9.7