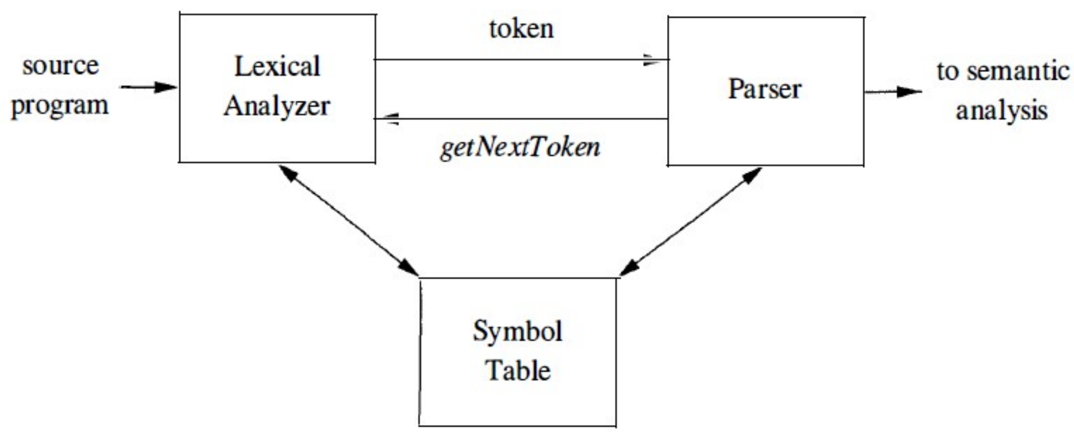
CSE504 Compiler Design
Lexical Analysis

YoungMin Kwon

The Role of the Lexical Analyzer



- Why separating lexical analysis and parsing
  - Simplify design (comments, white spaces…)
  - Improve compiler efficiency (simpler algorithm)
  - Improve compiler portability

Specification of Tokens

- String and Language
  - Alphabet (character class): any finite set of symbols.
  - A string of some alphabet: a finite sequence of symbols drawn from the alphabet.
  - Language: any set of strings over some fixed alphabet.
- Operations on Language

OPERATION	DEFINITION AND NOTATION
Union of L and M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L	$L^* = \cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+ = \cup_{i=1}^{\infty} L^i$

Regular Expressions

- Rules that define the regular expression over alphabet Σ

  - ϵ is a regular expression denoting $\{\epsilon\}$
  - If $a \in \Sigma$, a is a regular expression denoting $\{a\}$
  - $(r)|(s)$ is a regular expression denoting $L(r) \cup L(s)$
  - $(r)(s)$ is a regular expression denoting $L(r)L(s)$
  - $(r)*$ is a regular expression denoting $(L(r))*$
  - (r) is a regular expression denoting $L(r)$,

    where r and s are regular expressions denoting $L(r)$ and $L(s)$

Quiz: Find the language `L( a(b|c)* )`

Nonregular Sets

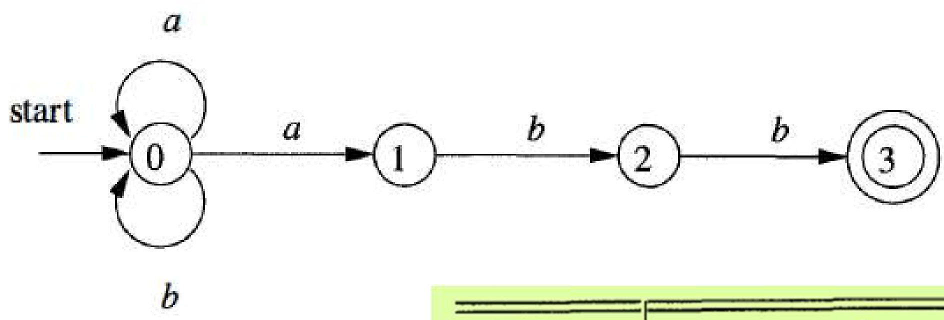
- Balanced or nested structure
  - e -> (e)
- Repeating strings
  - {wcw | w is a string of a's and b's}
- Arbitrary number of repetitions
  - n H a_1 a_2 … a_n

Finite Automata

- A recognizer for a language L is a program that takes a string x as an input and answers "yes" if x $\in$ L and "no" otherwise.
- Nondeterministic Automata (NFA) consist of

1. a set of status S
2. a set of input symbol Σ
3. a transition function *move*: maps (S, Σ) to S
4. an initial state $s_0 \in S$
5. a set of final states $F \subseteq S$

NFA example



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

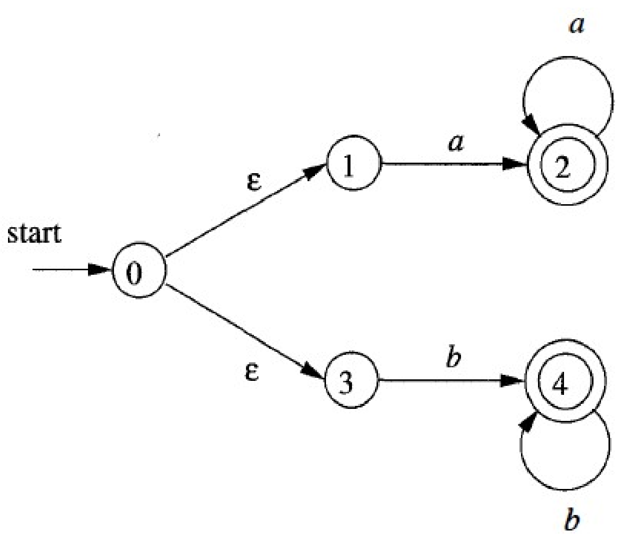
- In NFA, the same input string can result in different states.

$$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3$$

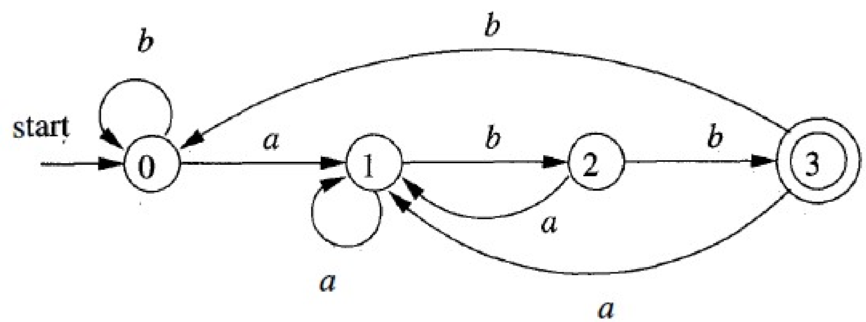
$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 0 \xrightarrow{b} 0$$

Finite Automata

- Deterministic Finite Automata (DFA)
  - DFA is a special case of NFA with
    - No state has an ϵ-transition
    - Each state has at most 1 edge for each input symbol.



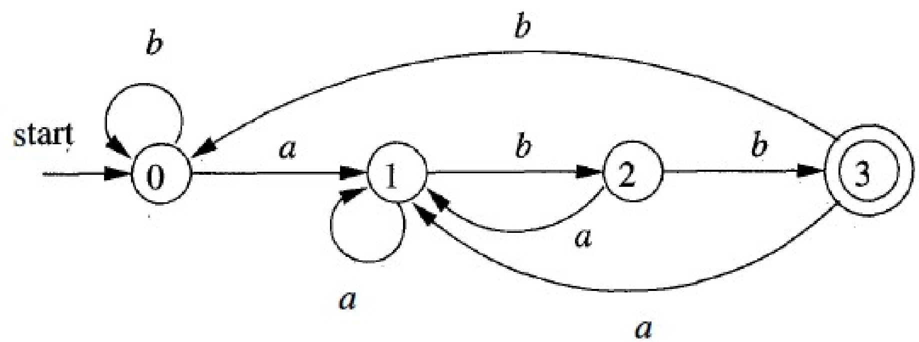
NFA accepting $\mathbf{aa}^*|\mathbf{bb}^*$



DFA accepting $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

Simulating DFA

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



DFA accepting $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

Quiz:

1. Check if aabbabb is in the language of (a|b)*abb
2. Check if aabbaa is in the language of (a|b)*abb

NFA to DFA

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.

```
initially, ε-closure(s0) 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 = ε-closure(move(T, a));
                if ( U is not in Dstates )
                        add U as an unmarked state to Dstates;
                Dtran[T, a] = U;
        }
}
```

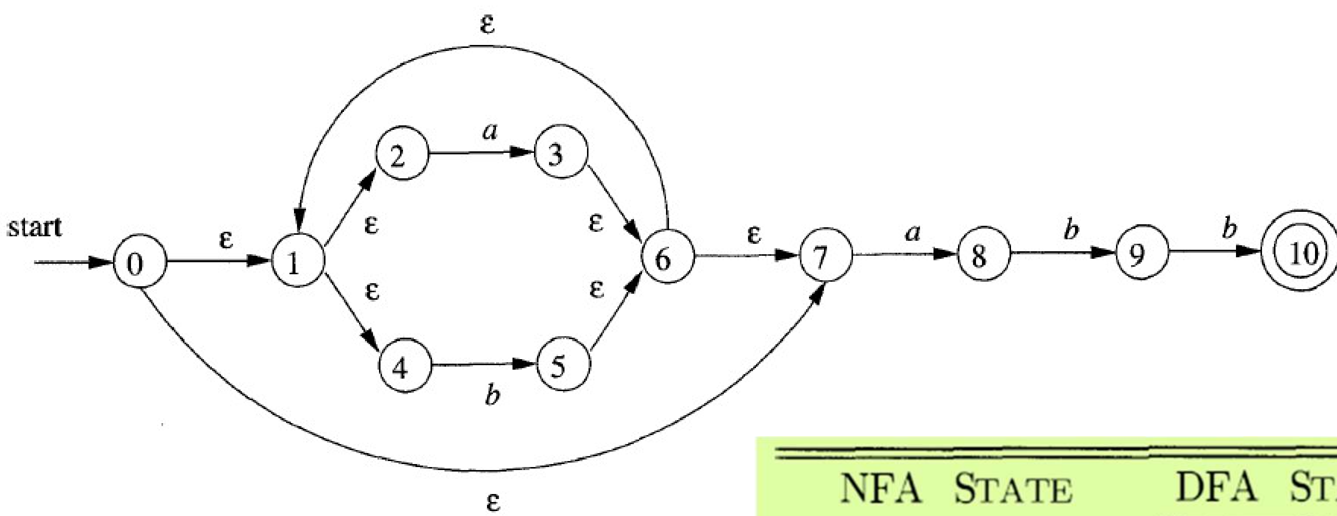
The subset construction

NFA to DFA Conversion

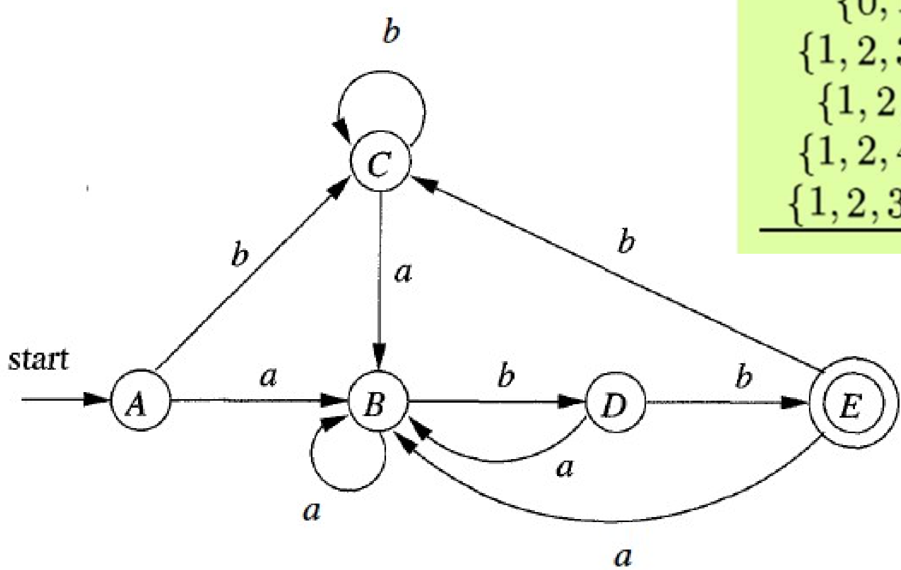
```
push all states of T onto stack;
initialize ε-closure(T) to T;
while ( stack is not empty ) {
        pop t, the top element, off stack;
        for ( each state u with an edge from t to u labeled ε )
                if ( u is not in ε-closure(T) ) {
                        add u to ε-closure(T);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

Computing ϵ-*closure*(T)

NFA to DFA Conversion example



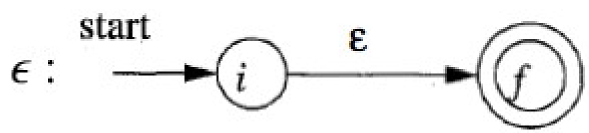
NFA STATE	DFA STATE	a	b
$\{0,1,2,4,7\}$	A	B	C
$\{1,2,3,4,6,7,8\}$	B	B	D
$\{1,2,4,5,6,7\}$	C	B	C
$\{1,2,4,5,6,7,9\}$	D	B	E
$\{1,2,3,5,6,7,10\}$	E	B	C

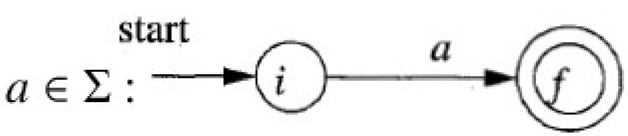


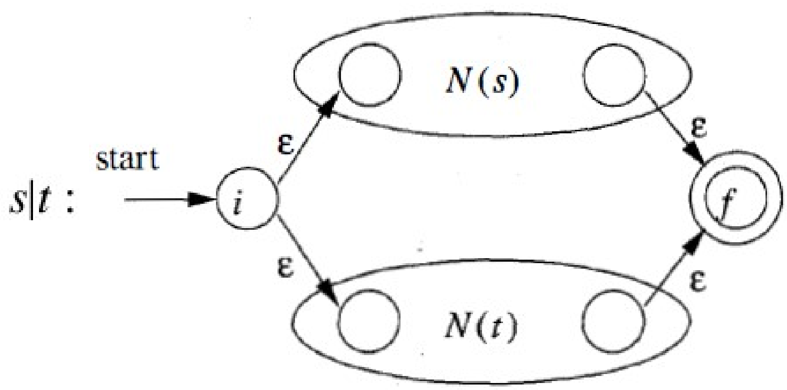
Regular Expression to NFA

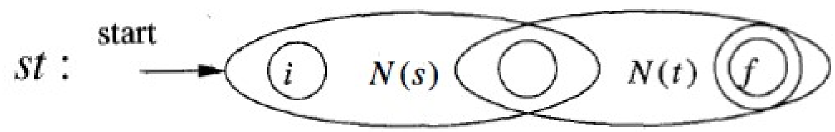
(Thompson’s Construction Algorithm)

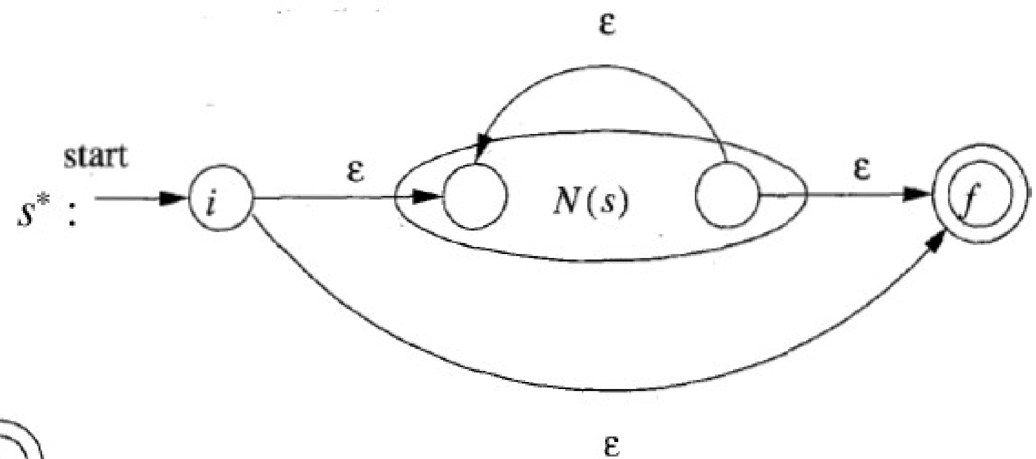
- Let N(s) and N(t) be NFAs for s and t





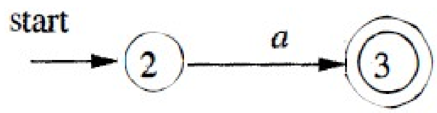


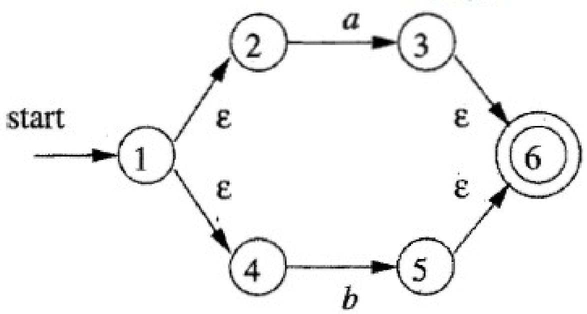


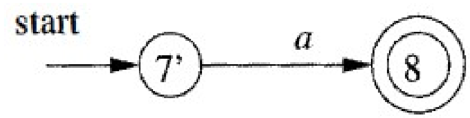


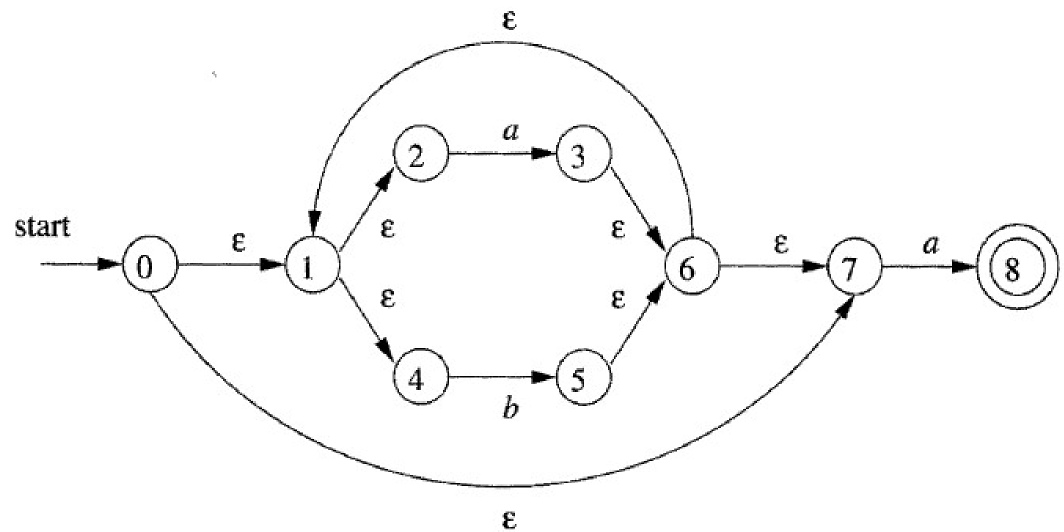
$(s) : N((s)) = N(s)$

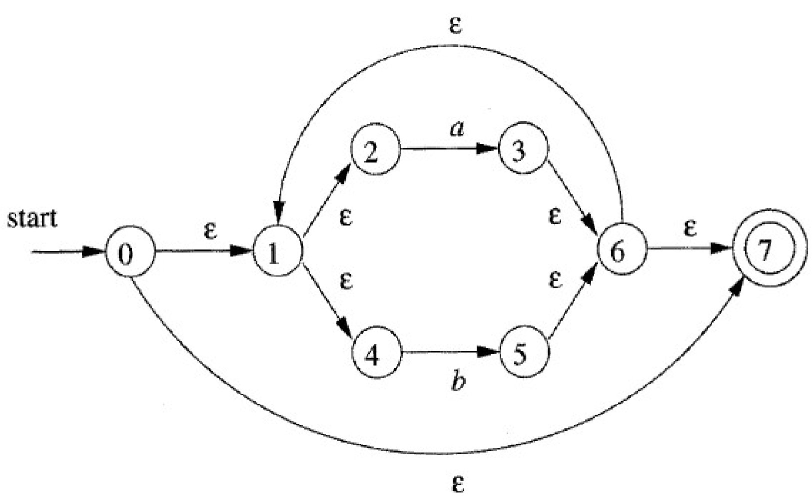
Regular Expression to NFA: (a|b)*abb









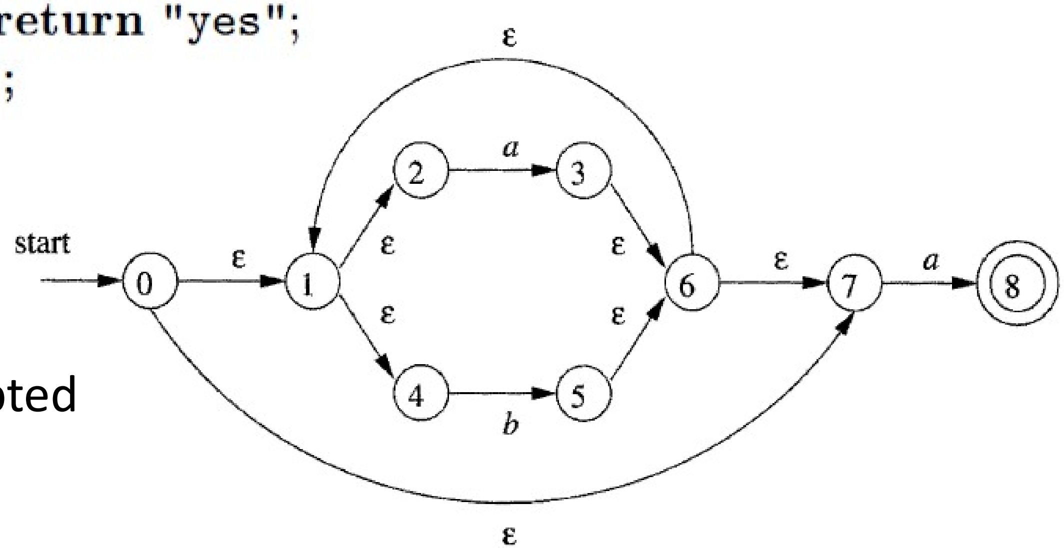


Quiz:

- Build an NFA for a(a|b)*b
- Convert the NFA to a DFA

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";
```



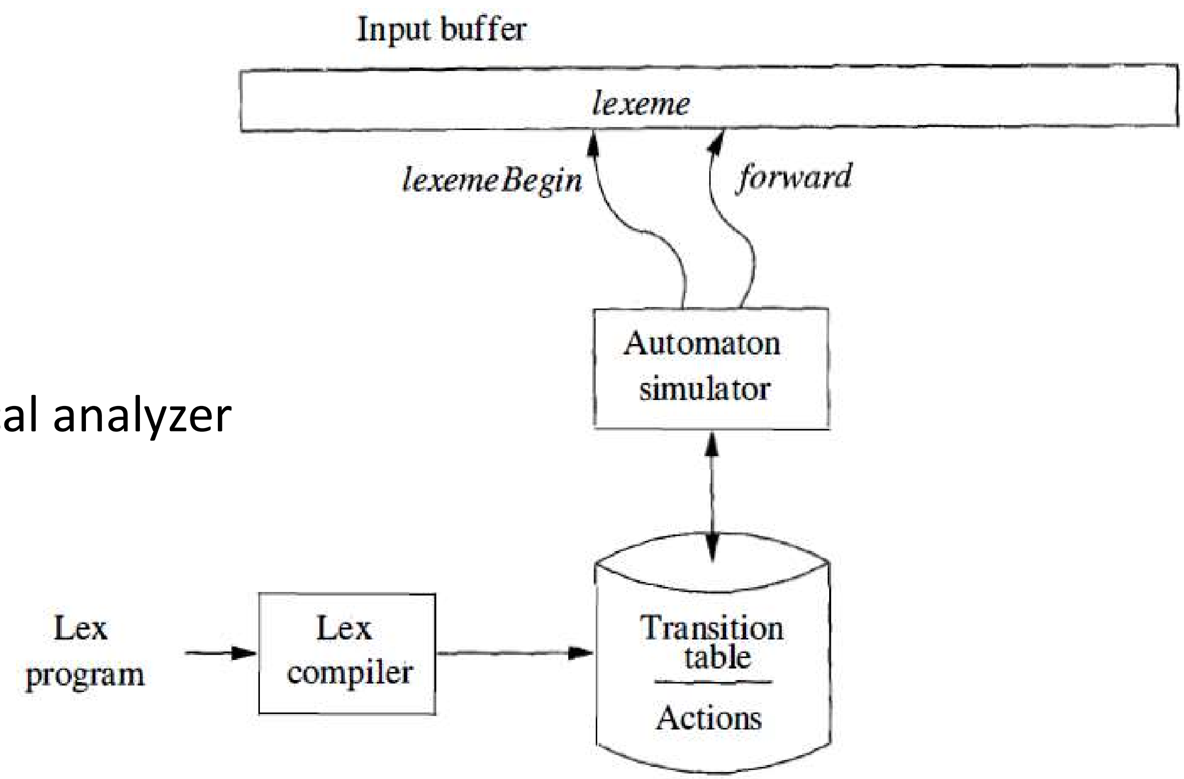
Quiz:

- Check if ababa will be accepted by the NFA on the right

Lexical Analyzer

p_1	{ $action_1$ }
p_2	{ $action_2$ }
. . .	. . .
p_n	{ $action_n$ }

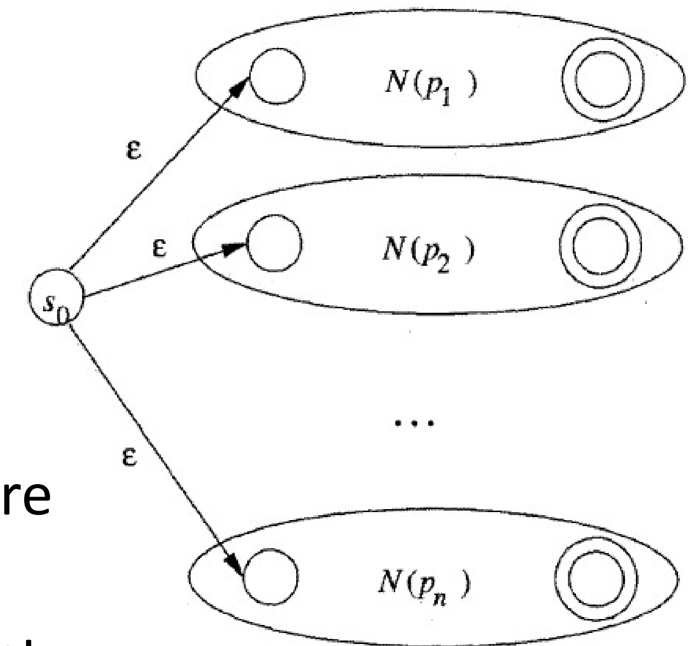
Specification of a lexical analyzer



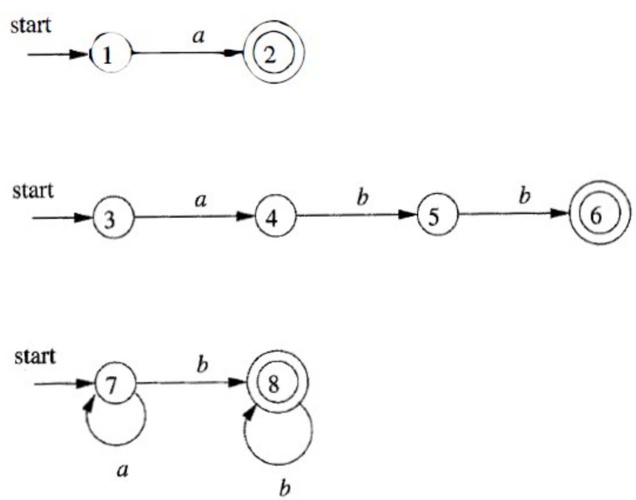
Model of Lex compiler

Pattern Matching with NFAs

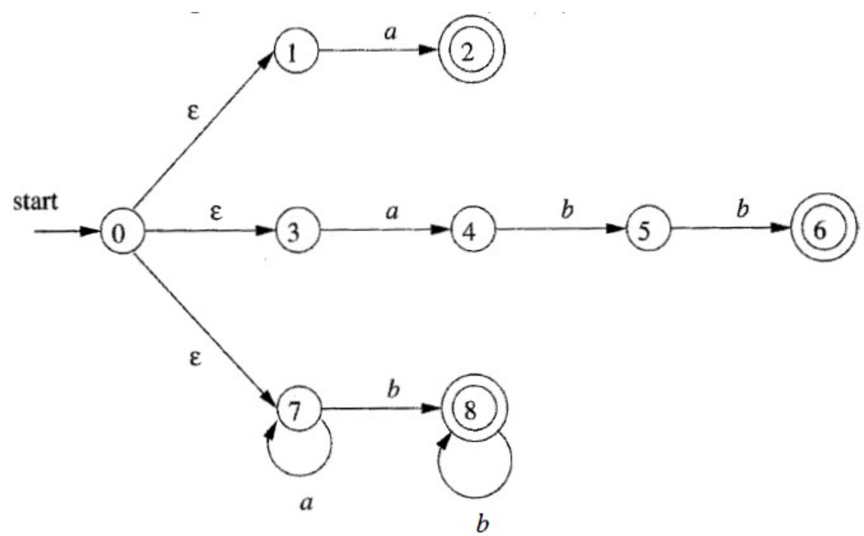
- For patterns p_1, … , p_n,
  - Construct NFAs $N(p_1)$, … ,$N(p_n)$
  - Add a start state s_0 and add ϵ transitions from s_0 to each $N(p_i)$.
  - To match the longest pattern, keep simulate NFA until there are no more transitions.
  - Move backward to the last state with an accepting state.



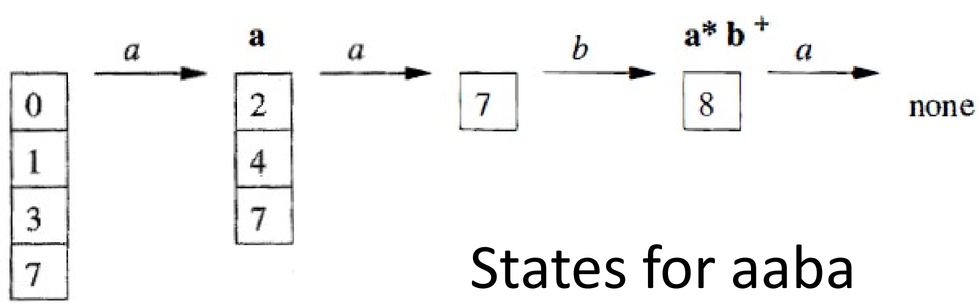
Pattern Matching Example



NFAs for a, abb, a*b+



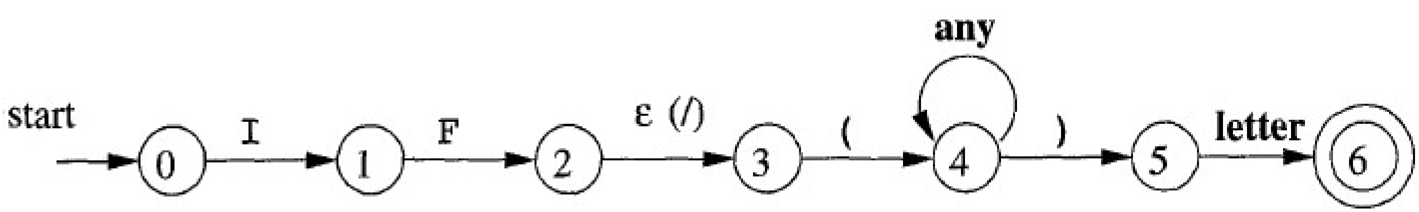
Combined NFA



States for aaba

The lookahead operator

- `r1/r2`: match a string in `r1` only if followed by a string in `r2`
  - E.g. in Fortran: `DO5I=1.25 vs DO5I=1,25`
    `DO/{letter_or_digit}*={letter_or_digit}*,`
- Implementing lookahead operator
  - When converting to NFA, treat / as ϵ
  - When a string is recognized, truncate the lexeme at the position where the last transition on the (imaginary) / occurred.
- E.g. `IF / \( .* \) {letter}`
  - `IF ( 123 ) a`

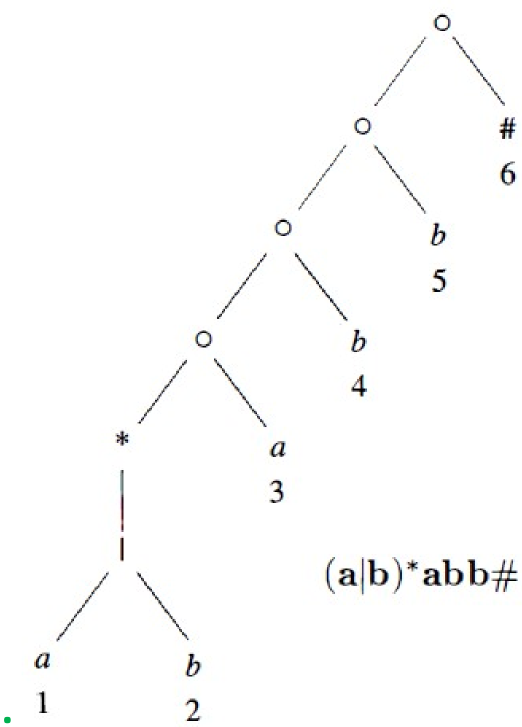


Regular Expressions to DFA

- Important States of NFA
  - An NFA state is important if it has a non-ϵ transition
  - Subset construction algorithm uses only important states (ϵ-closure(move(T,a)))
  - Two subsets can be identified if
    1. They have the same important states and
    2. They both have an accepting state or neither have one.
  - Thompson's construction builds an important state exactly when a symbol in the alphabet appears.
- Augmented regular expression
  - Append a unique marker # to a regular expression r: (r)#
  - Any DFA state with a transition on # is an accepting state.

Regular Expressions to DFA

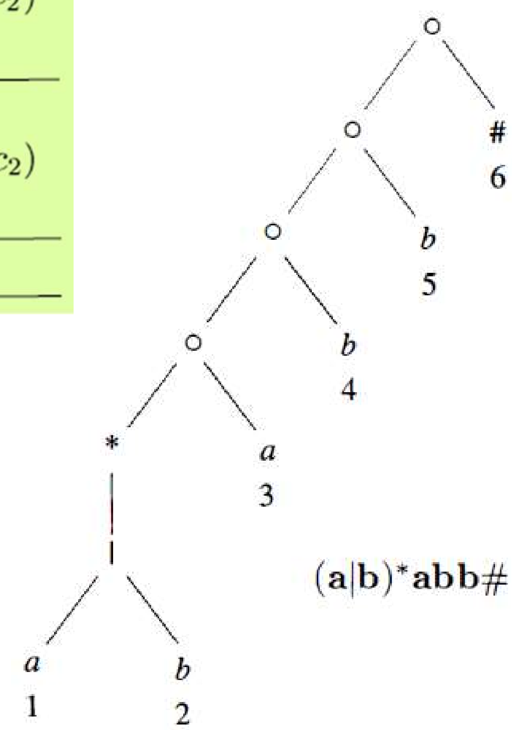
- Position: label non-ϵ leaves of a syntax tree for a regular expression with a unique number.
- For a node n in a syntax tree, let r be the subexpression corresponding to n.
  - nullable(n): if r can generate an empty string.
  - firstpos(n): the set of positions that can match the first symbols of the strings generated by r.
  - lastpos(n): the set of positions that can match the last symbols of the strings generated by r.
- For a position i, followpos(i): the set of positions j such that there is some input string ...cd... such that i corresponds to c and j to d.



Regular Expressions to DFA

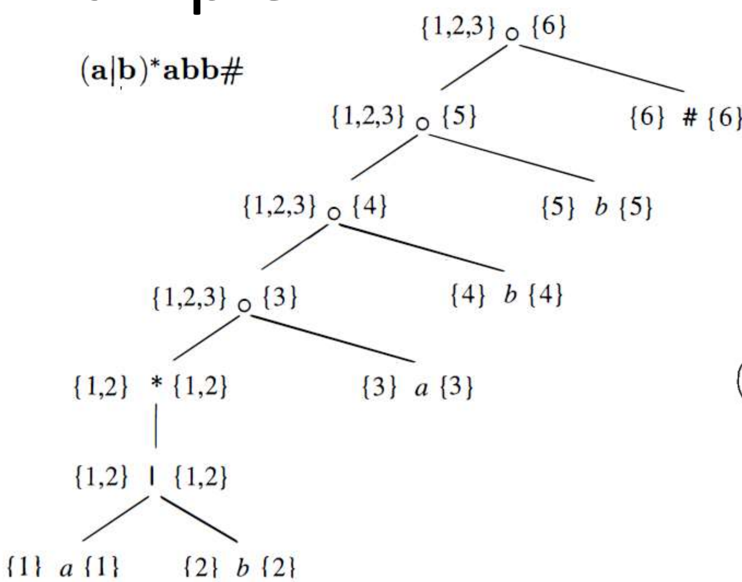
NODE n	$nullable(n)$	$firstpos(n)$
A leaf labeled ϵ	**true**	$\emptyset$
A leaf with position i	**false**	$\{i\}$
An or-node $n = c_1 \vert c_2$	$nullable(c_1)$ **or** $nullable(c_2)$	$firstpos(c_1) \cup firstpos(c_2)$
A cat-node $n = c_1c_2$	$nullable(c_1)$ **and** $nullable(c_2)$	**if** ($nullable(c_1)$) $firstpos(c_1) \cup firstpos(c_2)$ **else** $firstpos(c_1)$
A star-node $n = c_1{}^*$	**true**	$firstpos(c_1)$

- followpos(i)
  - If n is a cat-node with left c1 and right c2, and i ∈ lastpos(c1), then all positions in firstpos(c2) are in followpos(i).
  - If n is a star-node and i ∈ lastpos(n), then all positions in firstpos(n) are in followpos(i).

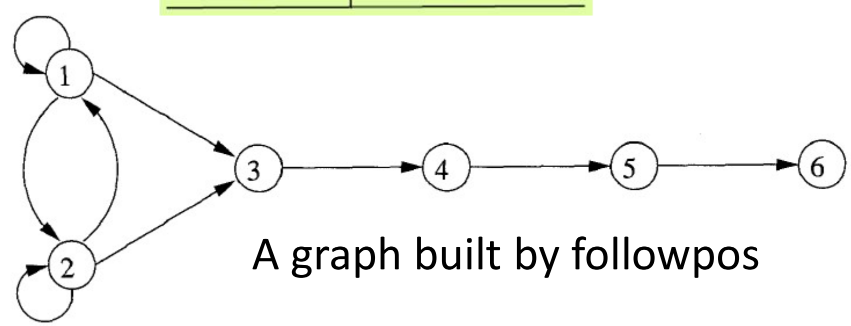


Regular Expressions to DFA

- Example



NODE n	*followpos*(n)
1	$\{1,2,3\}$
2	$\{1,2,3\}$
3	$\{4\}$
4	$\{5\}$
5	$\{6\}$
6	$\emptyset$



A graph built by followpos

Construct NFA without ϵ-transition

1. Make all positions in the firstpos of the root initial states
2. Label each edge (i,j) with the symbol at position i.
3. Make the position for # the only accepting state.

Regular Expressions to DFA

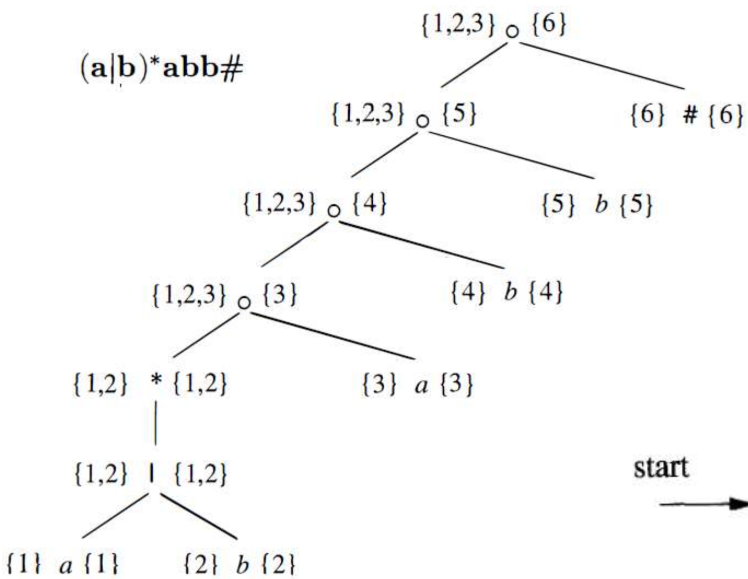
- Apply the subset construction algorithm directly to the implicit NFA.
  1. Construct a syntax tree for (r)#
  2. Construct, nullable, firstpos, lastpos, and followpos
  3. Construct Dstates and Dtran using the algorithm below. The start state is firstpos(root), the accepting states are the ones with the position for #.

```
initialize Dstates to contain only the unmarked state firstpos(n0),
        where n0 is the root of syntax tree T for (r)#;
while ( there is an unmarked state S in Dstates ) {
        mark S;
        for ( each input symbol a ) {
                let U be the union of followpos(p) for all p
                        in S that correspond to a;
                if ( U is not in Dstates )
                        add U as an unmarked state to Dstates;
                Dtran[S, a] = U;
        }
}
```

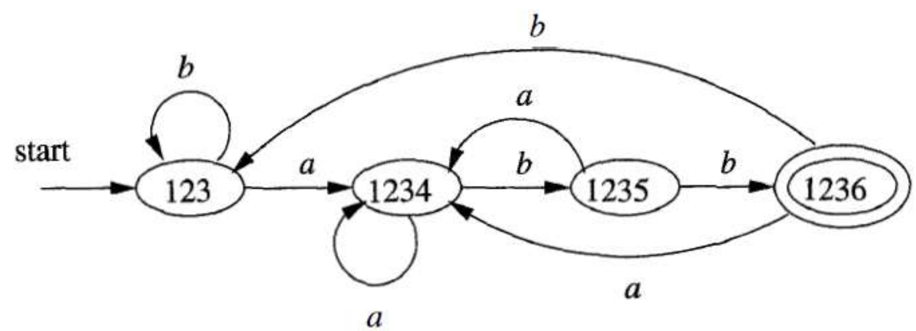
Regular Expressions to DFA

- Example

(a|b)*abb#



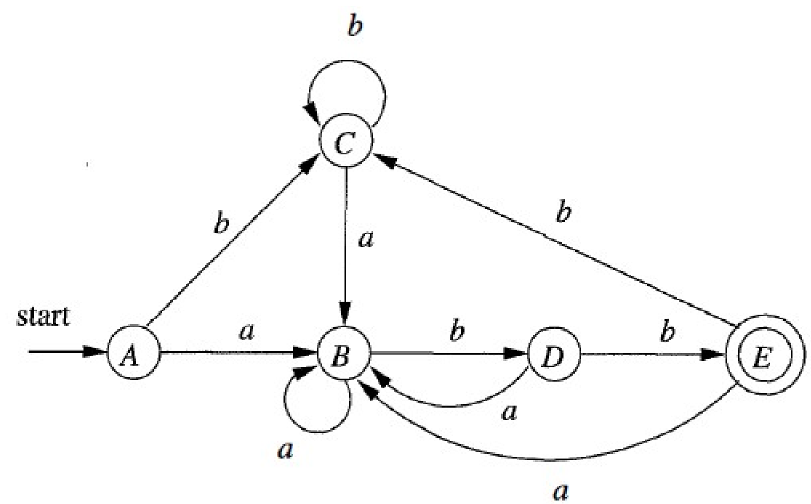
NODE n	$followpos(n)$
1	$\{1,2,3\}$
2	$\{1,2,3\}$
3	$\{4\}$
4	$\{5\}$
5	$\{6\}$
6	$\emptyset$



Quiz: Build a DFA for a(a|b)*b

Minimizing the number of DFA states

- Make every state has a transition on every input symbol. (add a dead state d if necessary)
- String w distinguishes states s and t if feeding w from the states ended up with an accepting state in one case and a non-accepting state in the other.
- Starting from F and S-F, keep partitioning the states until they are not distinguishable.



Minimizing the number of DFA states

1. Start with an initial partition Π with two groups, F and $S - F$, the accepting and nonaccepting states of D.

2. Apply the procedure of Fig. 3.64 to construct a new partition Π_{new}.

```
initially, let Π_new = Π;
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;
}
```

Figure 3.64: Construction of Π_{new}

3. If $\Pi_{\text{new}} = \Pi$, let $\Pi_{\text{final}} = \Pi$ and continue with step (4). Otherwise, repeat step (2) with Π_{new} in place of Π.

4. Choose one state in each group of Π_{final} as the *representative* for that group. The representatives will be the states of the minimum-state DFA D'. The other components of D' are constructed as follows: