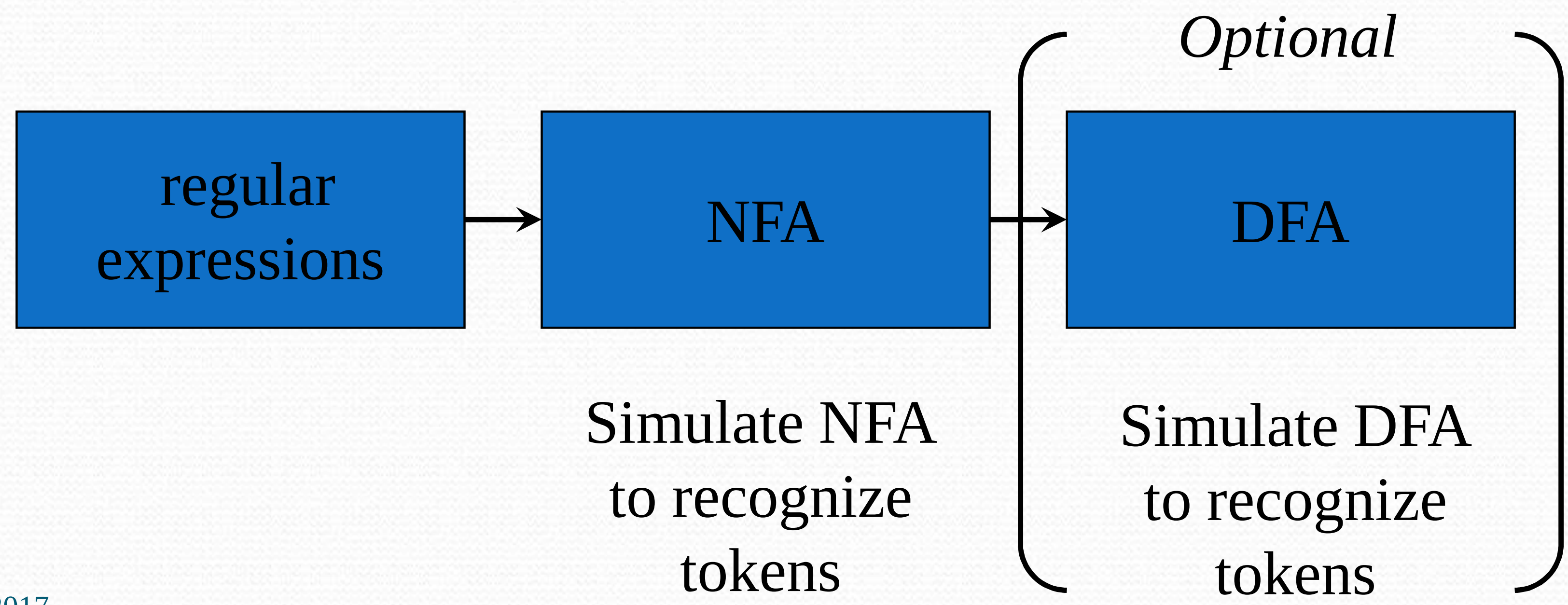
Finite Automata

Outline

- Regular expressions = specification
- Finite automata = implementation
- 2 Types DFA, NFA
- Deterministic Finite Automata (DFA)
  - One transition per input per state
  - No ε-moves
- Nondeterministic Finite Automata (NFA)
  - Can have multiple transitions for one input in a given state
  - Can have ε-moves
- *Finite* automata have *finite* memory
  - Need only to encode the current state

Design of a Lexical Analyzer Generator

- Translate regular expressions to NFA
- Translate NFA to an efficient DFA



Deterministic Finite Automata

- A DFA is a 5-tuple (S, Σ, δ, s_0, F) where

  S is a finite set of *states*
  Σ is a finite set of symbols, the *alphabet*
  δ is a *mapping* from $S \times \Sigma$ to a state
  $s_0 \in S$ is the *start state*
  $F \subseteq S$ is the set of *accepting* (or *final*) *states*

Simulating a DFA

- Input string x terminated by an eof. A DFA D
- Output yes if accepts else no

$S := s0$

$a := nextchar()$

while $a \neq$ eof **do**

        $S := move(S,a)$

        $a := nextchar()$

end do

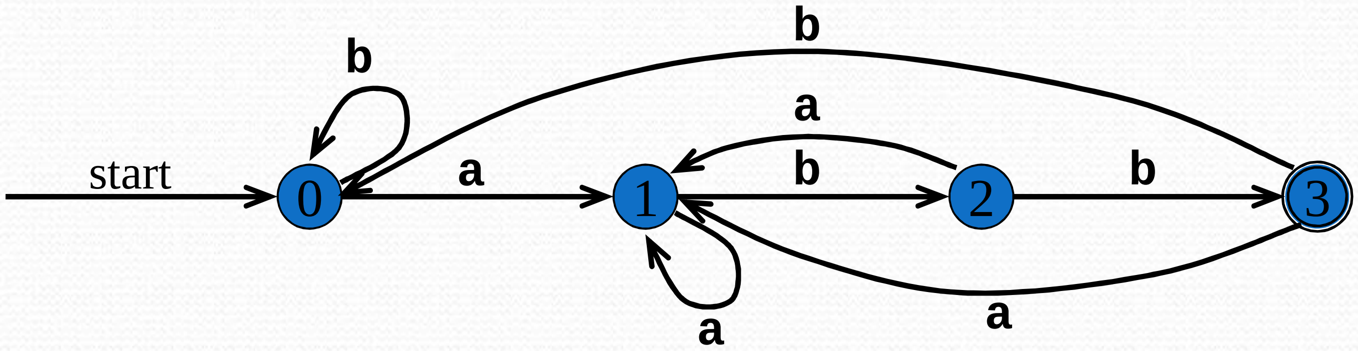
if S is in F **then**

        return "yes"

else **return** "no"

Example DFA

A DFA that accepts (**a** | **b**)***abb**



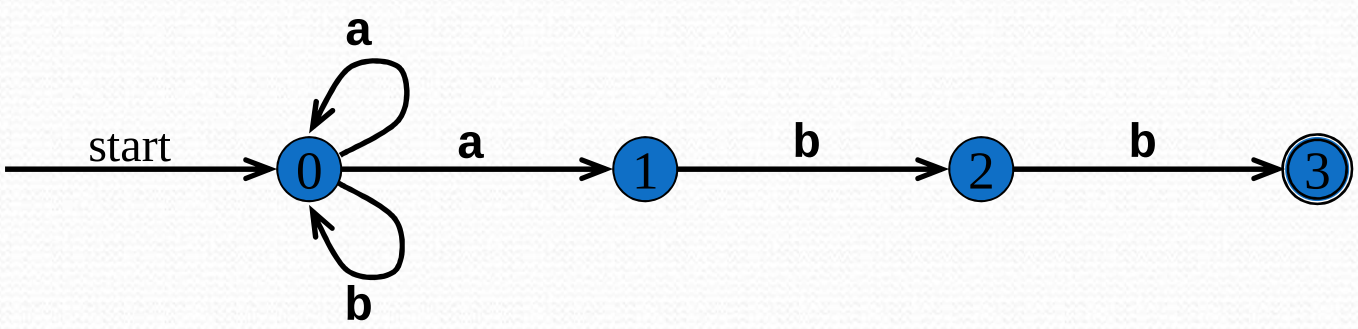
Nondeterministic Finite Automata

- An NFA is a 5-tuple (S, Σ, δ, s_0, F) where

  S is a finite set of *states*
  Σ is a finite set of symbols, the *alphabet*
  δ is a *mapping* from $S \times \Sigma$ to a set of states
  $s_0 \in S$ is the *start state*
  $F \subseteq S$ is the set of *accepting* (or *final*) *states*

Transition Graph

- An NFA can be diagrammatically represented by a labeled directed graph called a *transition graph*



$S = \{0,1,2,3\}$

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

$s_0 = 0$

$F = \{3\}$

Transition Table

- The mapping δ of an NFA can be represented in a *transition table*

$$\delta(0,\mathbf{a}) = \{0,1\}$$
$$\delta(0,\mathbf{b}) = \{0\}$$
$$\delta(1,\mathbf{b}) = \{2\}$$
$$\delta(2,\mathbf{b}) = \{3\}$$

$\longrightarrow$

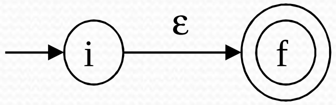
State	*Input* **a**	*Input* **b**
0	{0, 1}	{0}
1		{2}
2		{3}

Conversion of RE to NFA (Thompson's Construction)

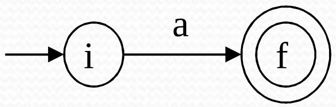
- Input: A RE r over an alphabet Σ
- Output: An NFA N for r
- It guarantees that the resulting NFA will have exactly one final state, and one start state.
- Decompose r into sub expressions.

Thompson's Construction cont...

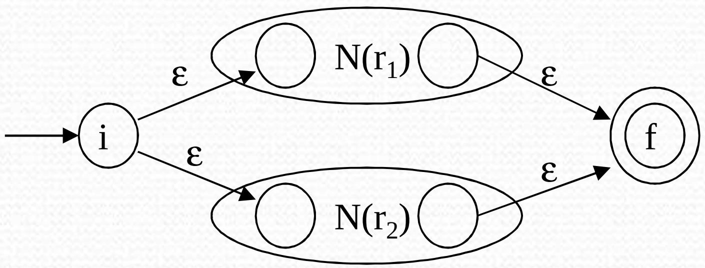
- To recognize an empty string ε



- To recognize a symbol a in the alphabet Σ



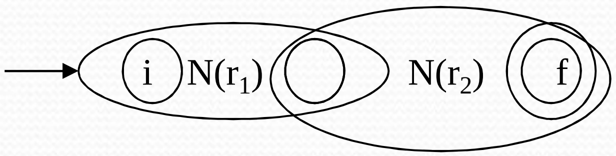
- If $N(r_1)$ and $N(r_2)$ are NFAs for regular expressions r_1 and r_2
  - For regular expression $r_1 \mid r_2$



NFA for $r_1 \mid r_2$

Thompson's Construction (cont.)

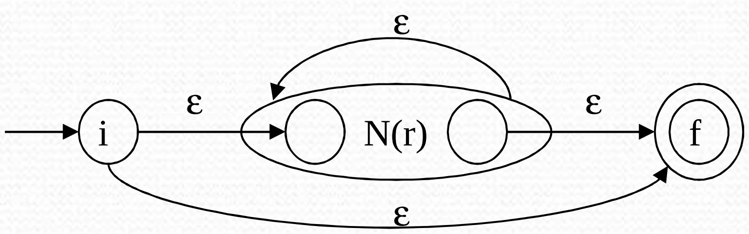
- For regular expression r_1
r_2



Final state of $N(r_2)$ become final state of $N(r_1r_2)$

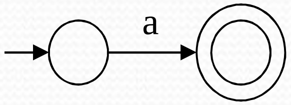
NFA for $r_1\ r_2$

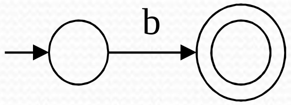
- For regular expression
r^*

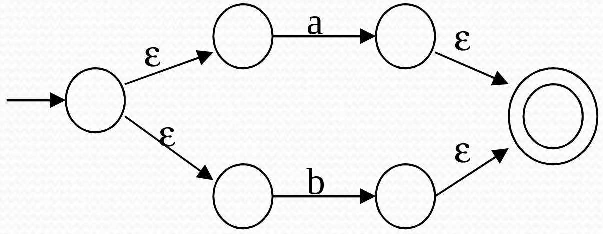


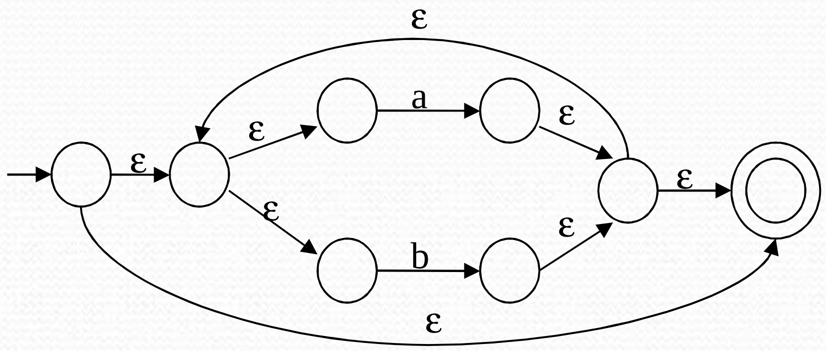
NFA for r^*

(Example - **(a|b) * a**)

a: 

b: 

(a | b) 

(a|b) * 

(a|b) * a