Compiler Construction

(Week 2, Lecture 2)

DFA

Deterministic Finite Automata

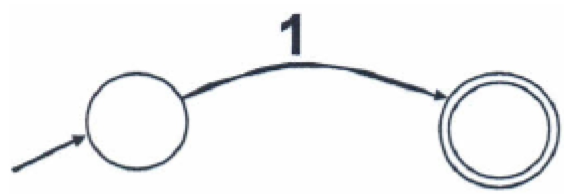
Definition (Deterministic Finite Automaton)

A determimistic finite automaton, or DFA, is the same as an NFA except

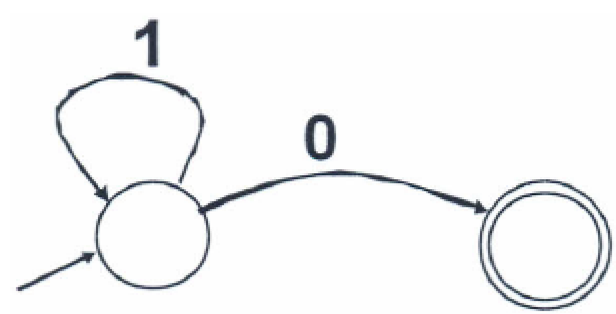
- The empty string ε is not included in the set of symbols. (No ε-moves.)
- For each state and for each symbol, there is *exactly one* next state.

DFA

- FA that accepts 1.

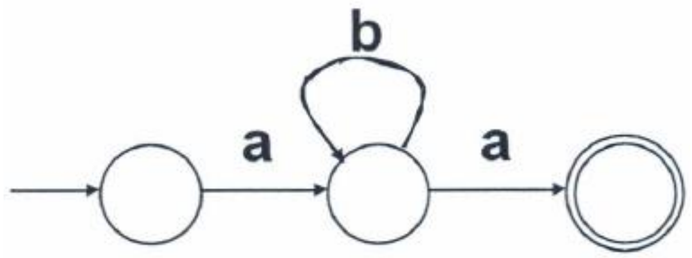


- An FA that accepts any number of 1s followed by a 0.



DFA

- An FA that accepts ab*a defined over $\Sigma=\{a,b\}$.



- Transition Table.

	a	b
0	*1*	err
1	*2*	*1*
2	err	err

```
int trans_table[NSTATES][NCHARS];
int accept_states[NSTATES];
int state = INITIAL;
while(state != err){
      c = input.read();
      if(c == EOF ) break;
      state=trans_table[state][c];
}
return accept_states[state];
```

- Examples: Logic 1, Logic 2

NFA

Nondeterministic Finite Automata

Definition (Nondeterministic Finite Automaton)

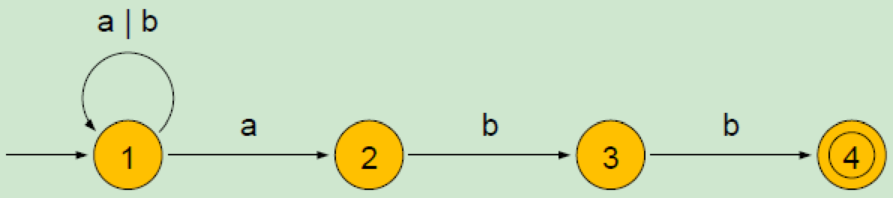
A nondeterministic finite automaton, or NFA, consists of

- A finite set S of states.
- An input alphabet Σ.
- A transition function δ that maps to each state-symbol pair a set of next states. The "symbol" may be ε.
- A set F of final states (or accepting states), where $F \subseteq S$.

NFA

The Transition Function

Example (The Transition Function)



- The transition function can be represented as a transition diagram.

NFA

The Transition Function

Example (The Transition Function)

State	Next State
1	$\{1, 2\}$
2	$\{3\}$
3	$\{4\}$
4	$\varnothing$

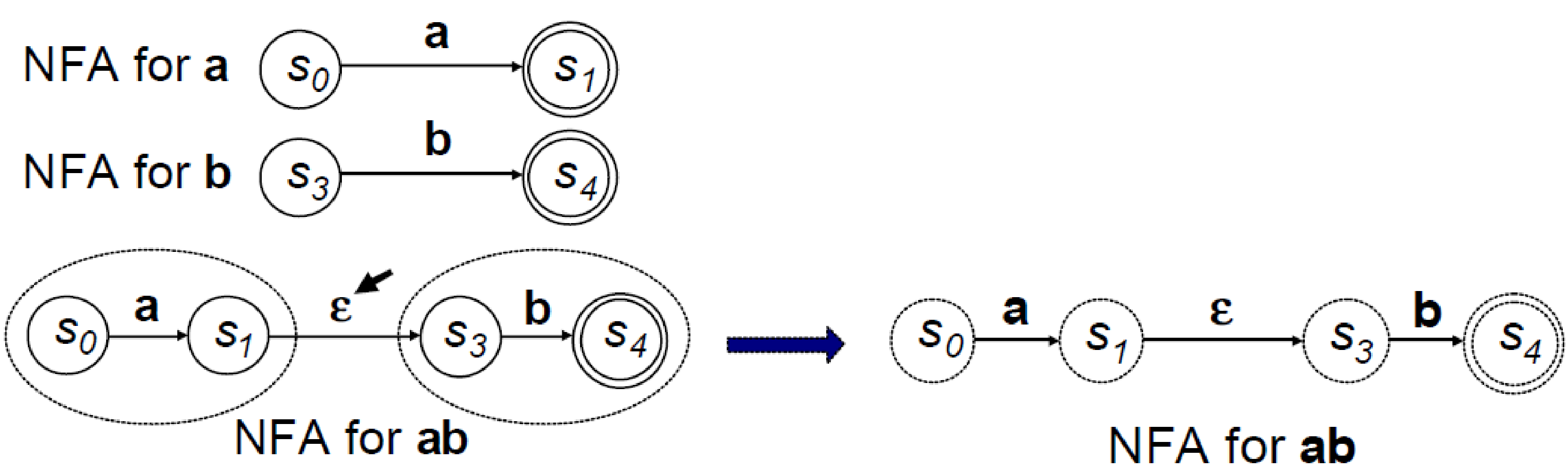
- The transition function can also be represented as a transition table.

NFA

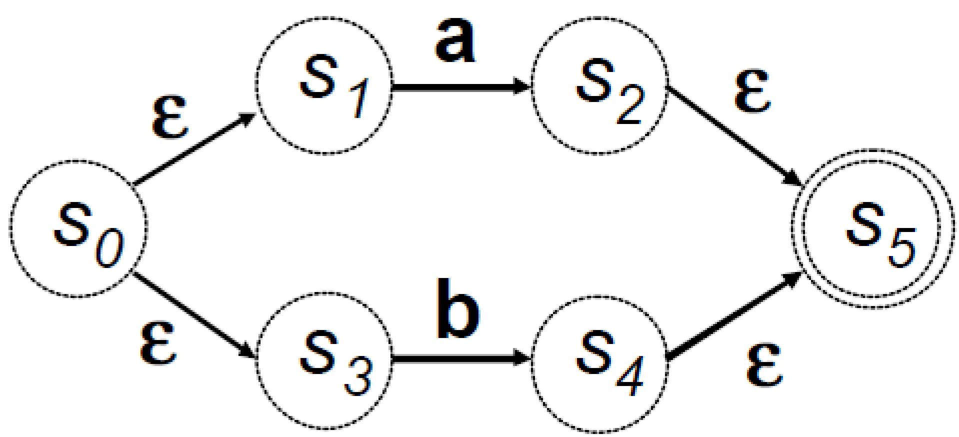
- Transition Table

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

RE to NFA

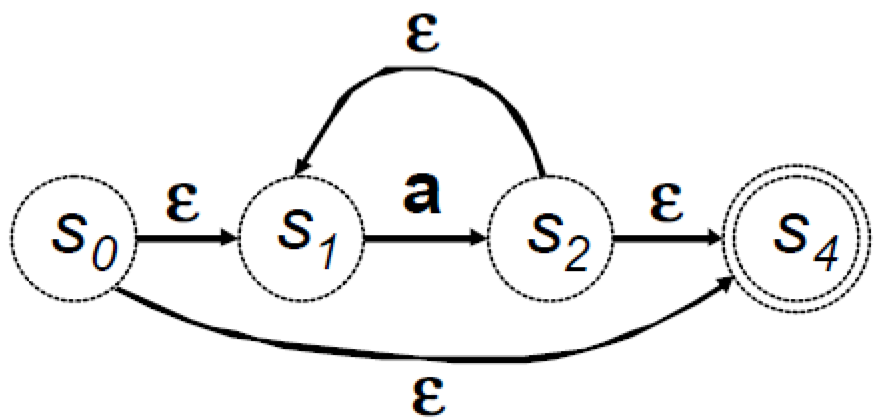


RE to NFA



NFA for **a | b**

RE to NFA



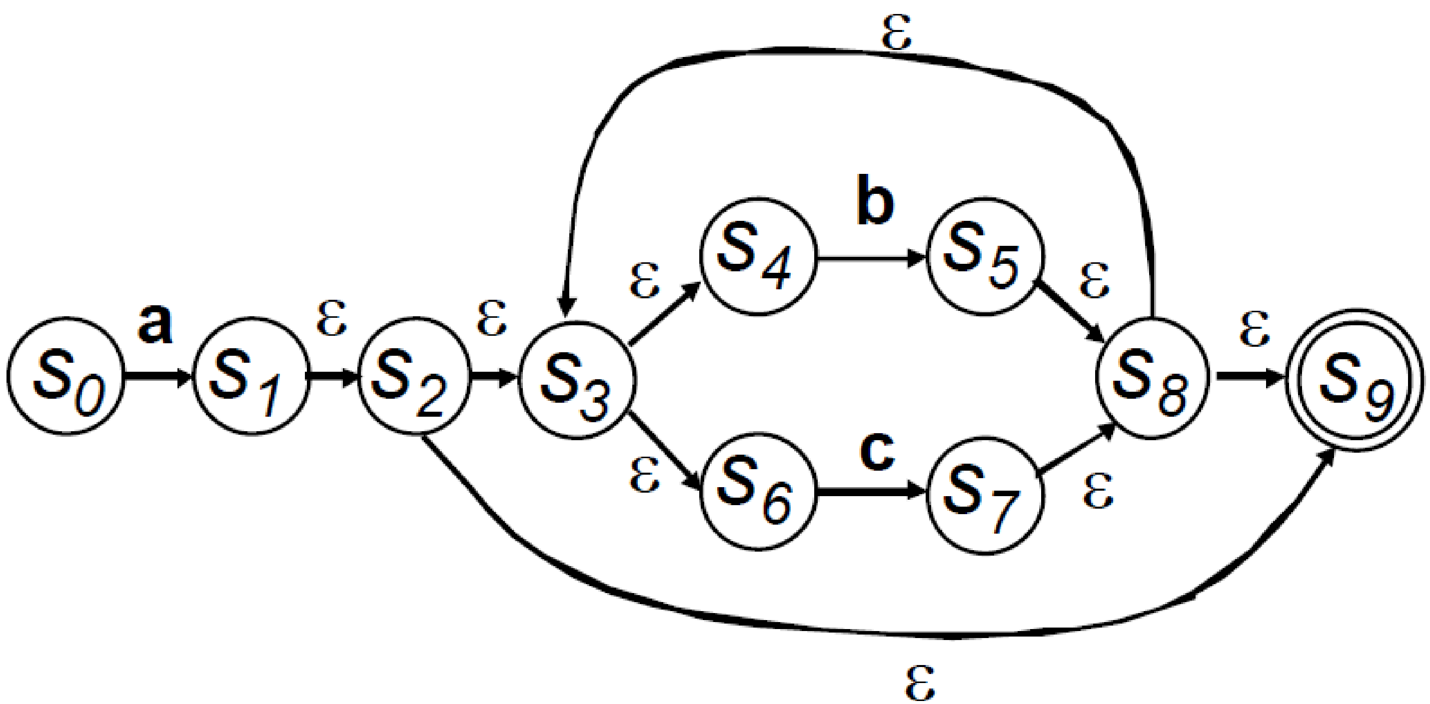
NFA for **a***

RE to NFA

- Q: NFA for RE a(b|c)*

RE to NFA

- Q: NFA for RE a(b|c)*



RE to NFA

Example

Example (Building a State Diagram)

- Build a state diagram from the regular expression

$$\mathbf{b}^*(\mathbf{ab}^*\mathbf{a})^*\mathbf{b}^*.$$

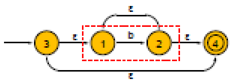
RE to NFA

Building State Diagrams

Create **b**

RE to NFA

Building State Diagrams

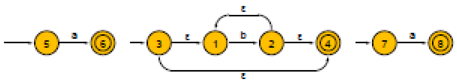


Form the Kleene closure **b***

Robb T. Koether (Hampden-Sydney College)

RE to NFA

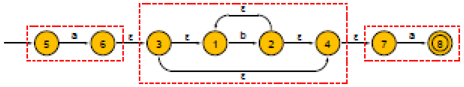
Building State Diagrams



Create two copies of **a**

RE to NFA

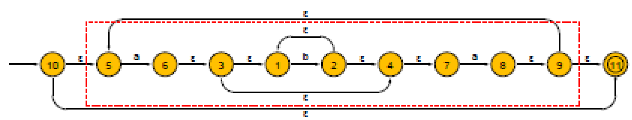
Building State Diagrams



Concatenate **a**, **b***, and **a**

RE to NFA

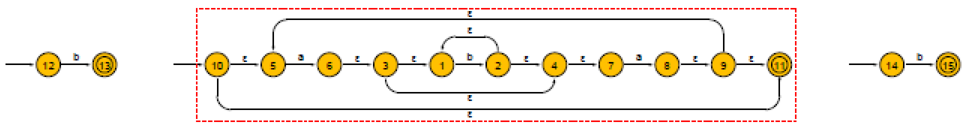
Building State Diagrams



Form the Kleene closure (**ab***a**)*

RE to NFA

Building State Diagrams

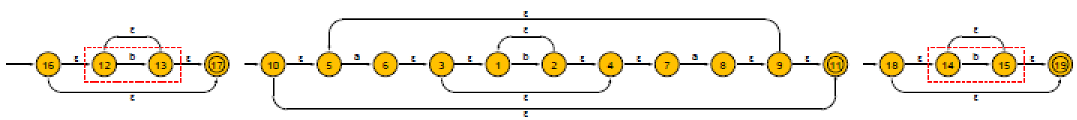


Create two copies of **b**

Robb T. Koether (Hampden-Sydney College)

RE to NFA

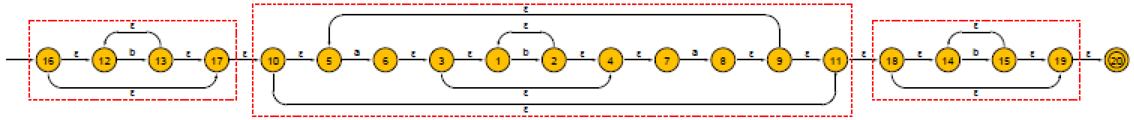
Building State Diagrams



Form the Kleene closures **b***

RE to NFA

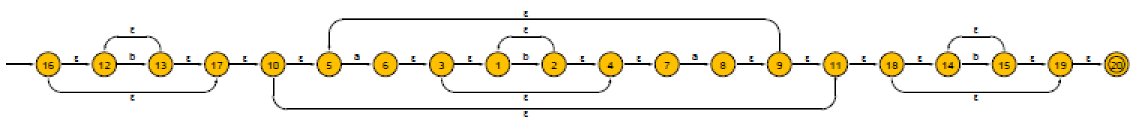
Building State Diagrams



Concatenate **b***, (**ab*****a**)*, and **b***

RE to NFA

Building State Diagrams

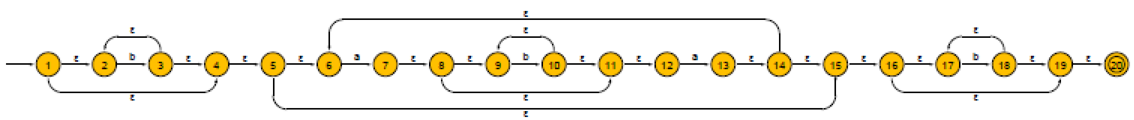


The regular expression **b***(**ab*****a**)***b***

Robb T. Koether (Hampden-Sydney College)

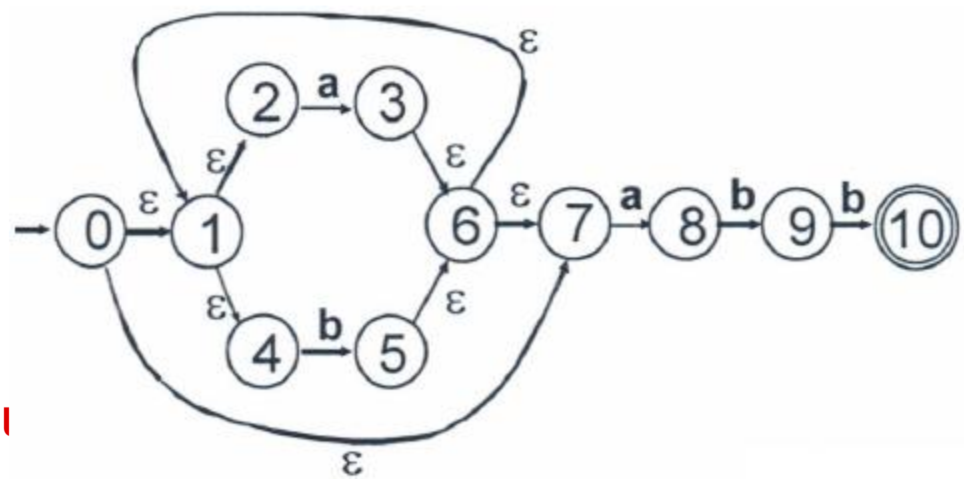
RE to NFA

Building State Diagrams



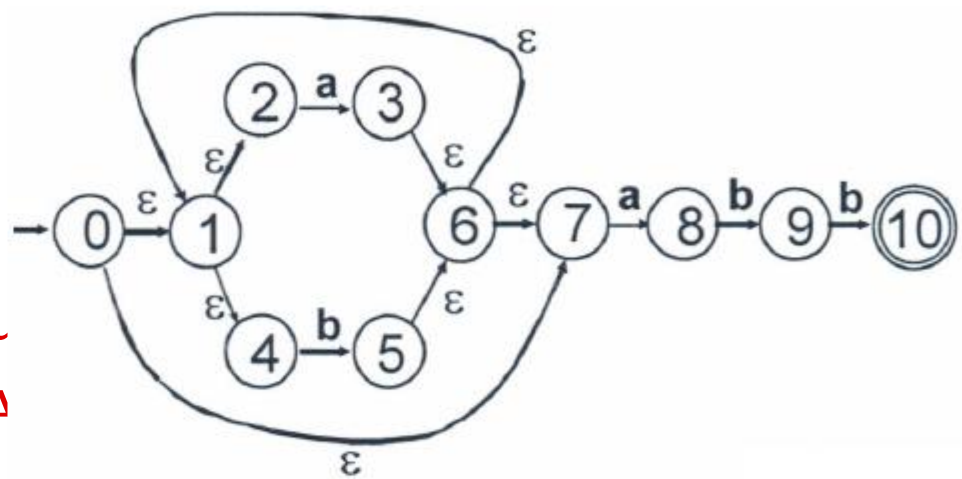
The states relabeled

NFA to DFA



- ε-clos
- Move(A,a) = {3,8}, ε-closure(Move(A,a)) = B
- Move(A,b) = {5}, ε-closure(Move(A,b)) = C

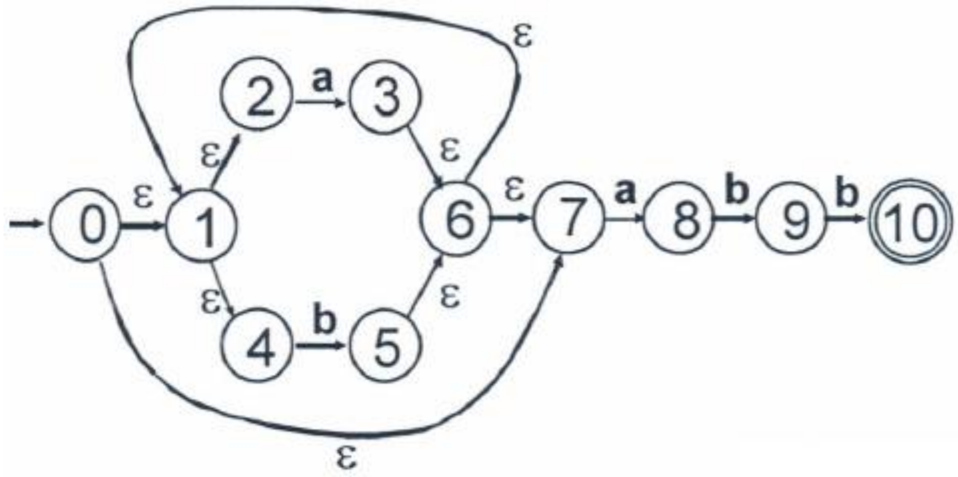
NFA to DFA



- ε-closu
- Move(A
- ε-closure(Move(A,a))

  = ε-closure({3,8}) = {1,2,3,4,6,7,8} = B

NFA to DFA

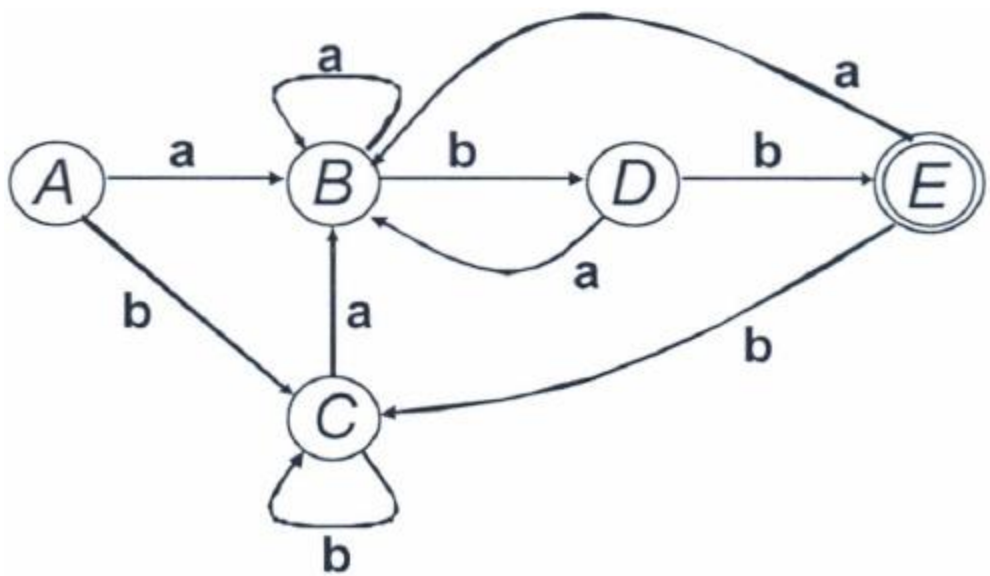


- ε-closure(0) = {0,1,2,4,7} = A
- Move(A, b) = {5}
- ε-closure(Move(A,b))

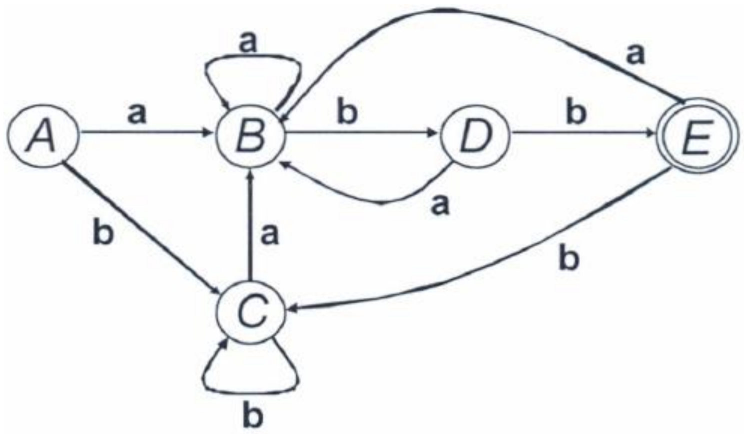
  = ε-closure({5}) = {1,2,4,5,6,7} = C

NFA to DFA

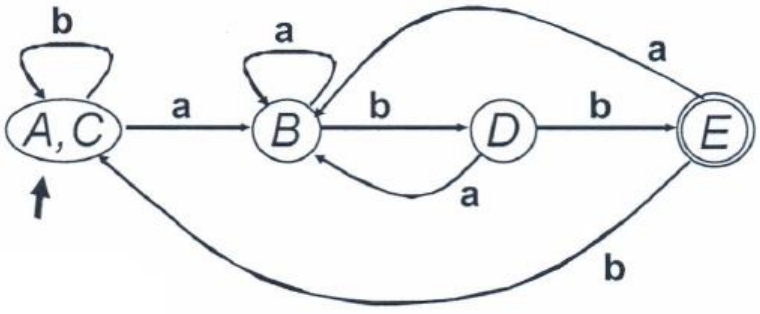
NFA STATE	DFA STATE	a	b
$\{0, 1, 2, 4, 7\}$	A	B	C
$\{1, 2, 3, 4, 6, 7, 8\}$	B	B	D
$\{1, 2, 4, 5, 6, 7\}$	C	B	C
$\{1, 2, 4, 5, 6, 7, 9\}$	D	B	E
$\{1, 2, 3, 5, 6, 7, 10\}$	E	B	C



Minimize DFA



- Minimized DFA



RE to NFA

Example

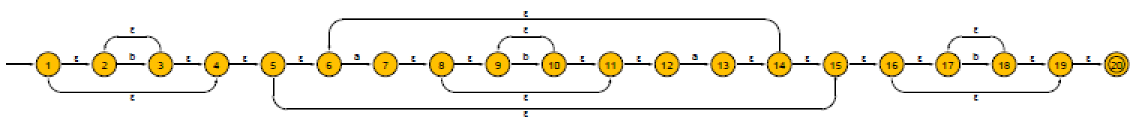
Example (Building a State Diagram)

- Build a state diagram from the regular expression

$$\mathbf{b}^*(\mathbf{ab}^*\mathbf{a})^*\mathbf{b}^*.$$

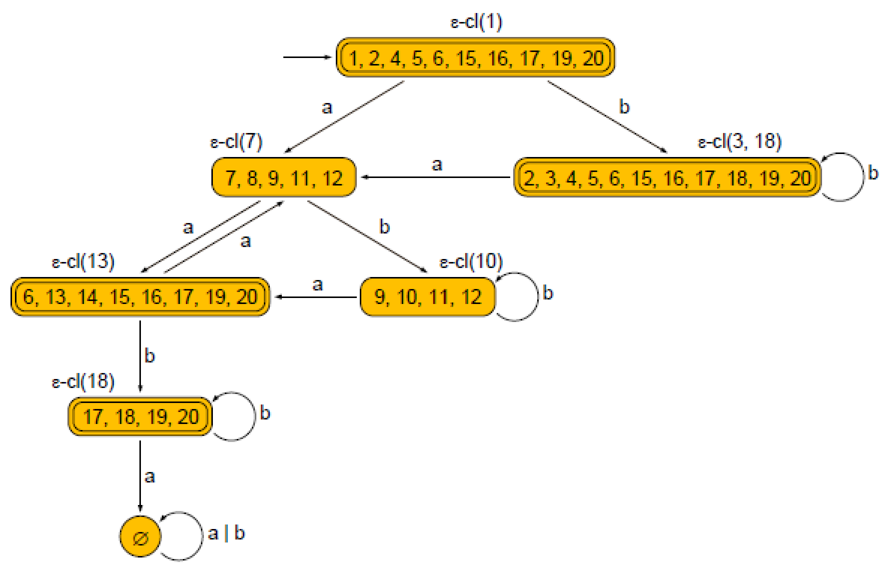
RE to NFA

Building State Diagrams



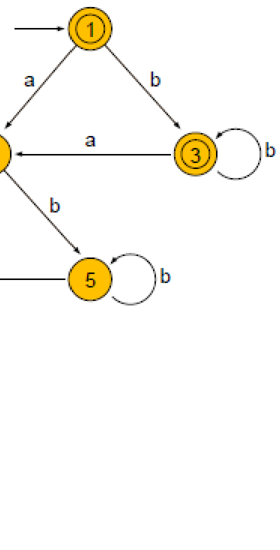
The states relabeled

Building State Diagrams



The DFA

Building State Diagrams

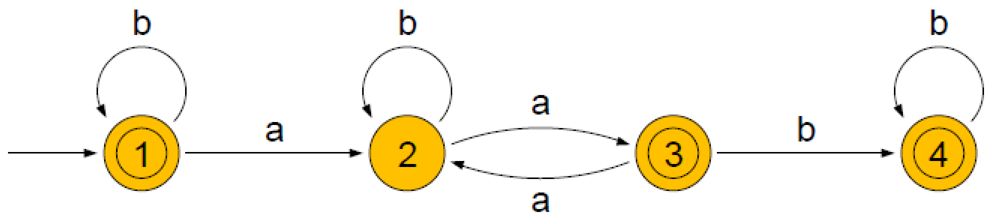


Relabel the states

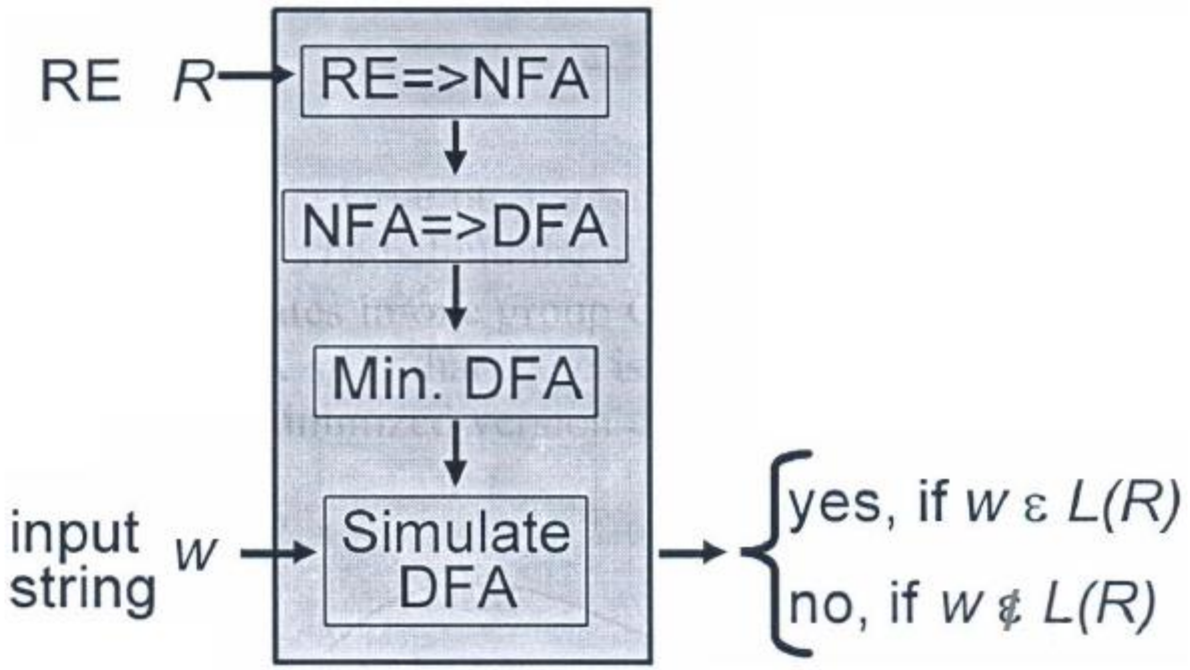
Building State Diagrams

Eliminate the unnecessary states $(\{1, 3\}, \{2, 5\})$

Building State Diagrams



Arrange in a simpler form


RE R
RE=>NFA
NFA=>DFA
Min. DFA
input string w
Simulate DFA
yes, if w ε L(R)
no, if w ∉ L(R)

FLex

- To use Flex, one has to provide a specification file as input to Flex.
- Flex reads this file and produces an output file contains the lexical analyzer source in C or C++.

FLex

- Detailed guide to Flex included in supplementary reading material.
- The input specification file consists of three sections.
- The symbols “%%” mark each section.

FLex

C or C++ and flex definitions

%%

token definitions and actions

%%

user code

Flex (RE)

RE	Description
a	Ordinary character from S
ε	The empty string
R|S	Either R or S
RS	R followed by S (concatenation)
R*	Concatenation of R zero or more times (R* = e|R|RR|RRR...)

Flex (RE)

RE	Description
R?	ε | R (Zero or one R)
R+	RR* (one or more R)
(R)	R (grouping)
[abc]	a|b|c (Any of listed)
[a-z]	a|b|....|z (Range)
[^ab]	c|d|... (Anything but 'a''b')

Flex (RE)

RE	Description
a	a
ab	ab
a|b	a or b
(ab)*	ε, ab, abab, ababab, …
(a|ε)b	ab or b
digit	0|1|2|3|4|5|6|7|8|9
integer	digit digit*
identifier	[a-zA-Z_][a-zA-Z0-9_]*

Flex

```
%{
#include <stdio.h>
%}

[a-zA-Z_][a-zA-Z0-9_]* printf("Valid Identifier");
%%

main()
{
    yylex();
}
```