FINITE AUTOMATA

Based on Chapter 3 of Aho, Lam, Sethi, Ullman:

Compilers: Principles, Techniques, & Tools

2nd Ed, Addison Wesley, 2007

Table of Contents

Introduction

- Finite Automata or Finite State Machine
  - a mathematical way of describing particular kinds of algorithms (or machines)
  - to describe the process of recognizing patterns in input strings
  - to construct scanners
  - strong relationship with regular expressions
  - any regular expression can be converted into an equivalent finite automata
- Two Types
  - deterministic finite automata(DFA) vs nondeterministic finite automata(NFA)
  - any NFA can be converted into an equivalent DFA
  - both automata are capable of recognizing the same languages, called the regular languages

Nondeterministic Finite Automata

- An NFA *M* consists of
  - S : a set of states
  - Σ : a set of input symbol(alphabet)
  - T : a transition function T: $S \times (\Sigma \cup \{ \varepsilon \}) \rightarrow P(S)$
  - $s_0 \in S$, a start state
  - $F \subseteq S$: a set of accepting states

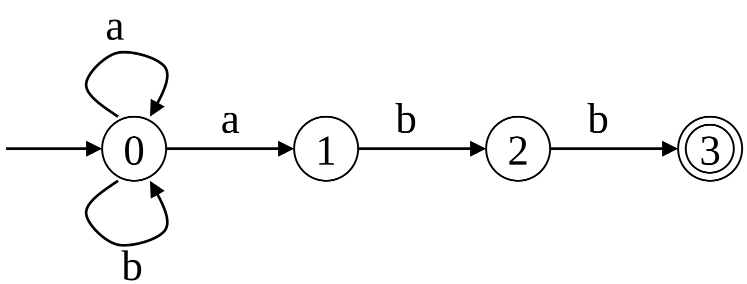
- *L(M)* is the set of strings $c_1c_2..c_n$, $c_i \in \Sigma \cup \{ \varepsilon \})$

  $s_1 \in T(s_0,c_1)$, $s_2 \in T(s_1, c_2)$, …, $s_n \in T(s_{n-1}, c_n)$, $s_n \in F$

- An NFA *M accepts* an input string s if an only if s is in *L(M)* and *rejects* otherwise.

Nondeterministic Finite Automata

- An NFA can be represented diagrammatically by a transition diagram
  - state: node, transition function: labeled edges



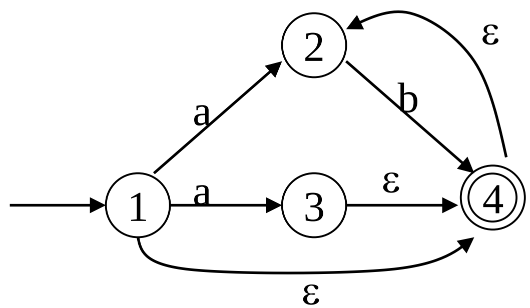
M1: (a|b)*abb

- Characteristics
  - ε-transition
  - more than one transitions over an input character
  - more than one sequence of transitions can lead to an accepting state
  - other path that can lead to non-accepting state may be made
  - NFA does not represent an algorithm, but can be simulated by backtracking or subset construction

Nondeterministic Finite Automata

- Example
  - string *aabb* in M1 can lead to state 3 or to state 0
  - string *abb* in M2 can be accepted by either of two sequences

1->2->4->2->4
1->3->4->2->4->2->4



M2: (a| ε)b*

- Transition Table

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

M1

State	Input Symbol		
	a	b	ε
1	**{2,3}**	-	{4}
2	-	{4}	-
3	-	-	**{ε}**
4	-	**{ε}**	-

M2

Deterministic Finite Automata

- An DFA *M* consists of
  - S : a set of states
  - Σ : a set of input symbol(alphabet)
  - **T : a transition function T: S x $\Sigma \rightarrow$ S**
  - $s_0 \in$ S, a start state
  - F $\subseteq$ S : a set of accepting states

- Characteristics
  - no ε-transition
  - at most one edge from each state *s* on an input symbol *a*
  - DFA represents an algorithm for the recognizer

Deterministic Finite Automata

- Simulating a DFA *M*

Input: a string x terminated *eof*

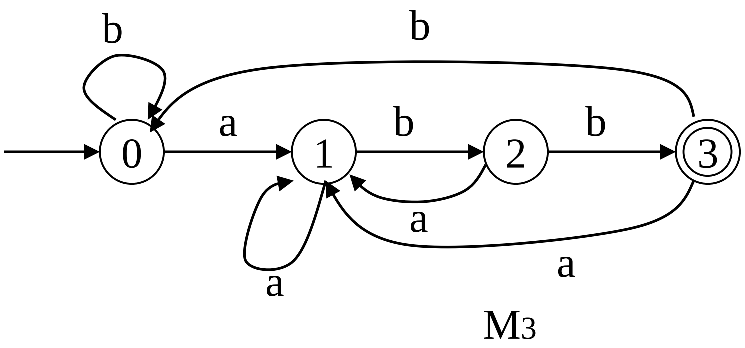
Output: "yes" if *M* accepts the string x; "no" otherwise

Algorithm:

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

Deterministic Finite Automata

- Example : (a|b)*abb

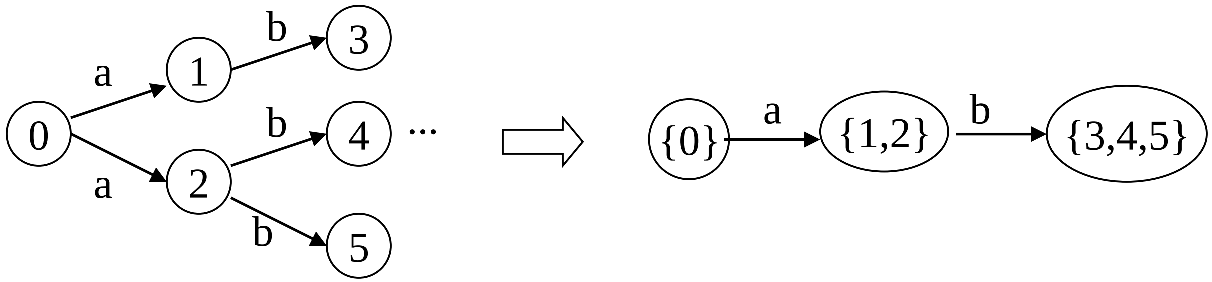


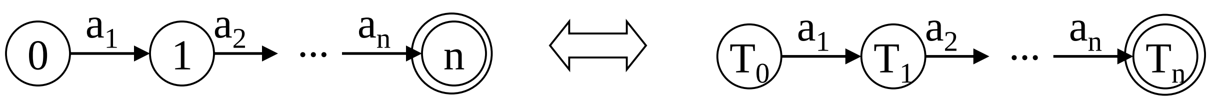
- Transition Diagram

State	Input Symbol	
	a	b
0	1	0
1	1	2
2	1	3
3	1	0

From an NFA to a DFA

- Subset Construction
  - general idea





T_i is a subset of S_N

$i \in Ti$

From an NFA to a DFA

- Subset Construction
  - ε-closure(s) : set of NFA states reachable form NFA state s on ε-transitions
  - ε-closure(T): set of NFA states reachable from NFA states in T on ε-transitions
  - move(T,a) : set of NFA states to which there is a transition on input symbol *a* from some NFA state s in T

Algorithm:

initially, ε-*closure*(s_0) is 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*;

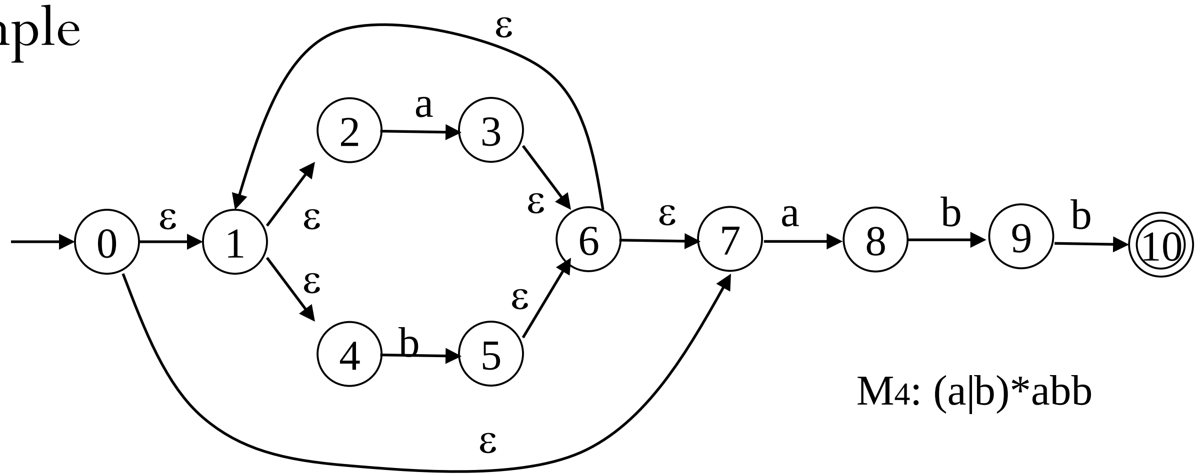
    Dtran[T, a] = U

  }

}

From an NFA to a DFA

- Example



M4: (a|b)*abb

ε-*closure*(0) = {0,1,2,4,7} = A

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

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

ε-*closure*(*move*(B,a)) = ε-*closure*({3,8}) = B

ε-*closure*(*move*(B,b)) = ε-*closure*({5,9}) = {1,2,4,5,6,7,9} = D

ε-*closure*(*move*(C,a)) = ε-*closure*({3,8}) = B

ε-*closure*(*move*(C,b)) = ε-*closure*({5}) = C

ε-*closure*(*move*(D,a)) = ε-*closure*({3,8}) = B

ε-*closure*(*move*(D,b)) = ε-*closure*({5,10}) = {1,2,4,5,6,7,**10**}= E <- **final state**

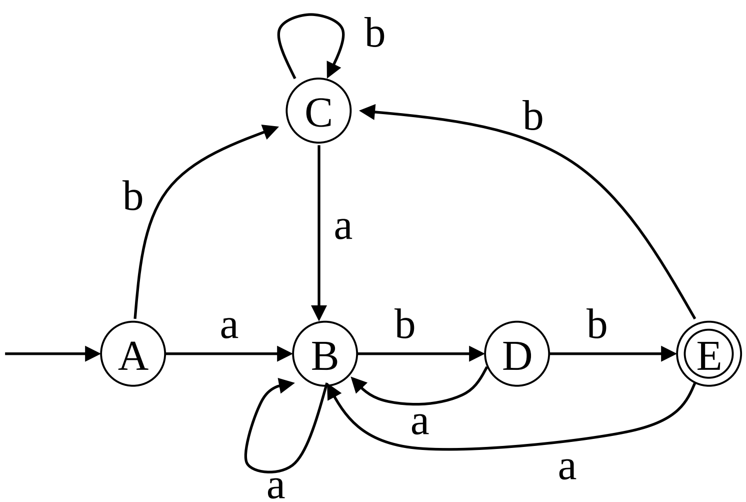
ε-*closure*(*move*(E,a)) = ε-*closure*({3,8}) = B

ε-*closure*(*move*(E,b)) = ε-*closure*({5}) = C

From an NFA to a DFA

- Example: M5

State	Input Symbol	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



cf. compare it with M3

Simulation of NFA

- Simulating an NFA

Input: NFA N with start state s_0, accepting states F, and input x terminated by eof

Output: yes or no

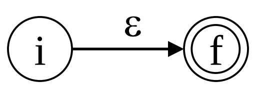
Algorithm:

```
S = ε-closure({s0});
a = nextchar();
while (a != eof ){
   S = ε-closure(move(S,a));
   a = nextchar();
}
if (S ∩ F is not empty )
    return "yes"
else
    return "no"
```

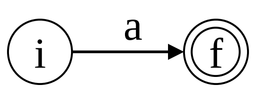
From a Regular Expression to an NFA

- Thompson's Construction

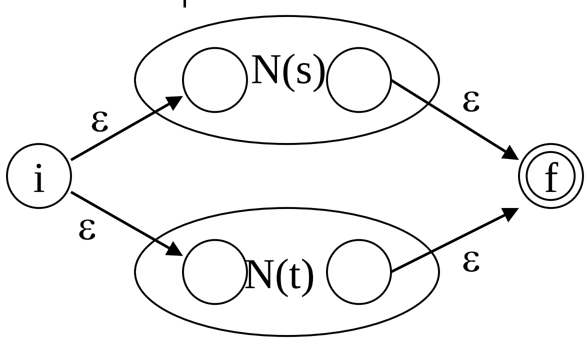
1. for ε



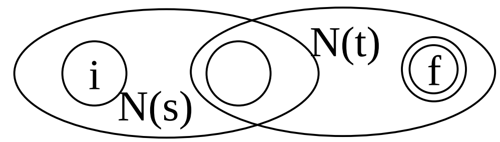
2. for a in Σ



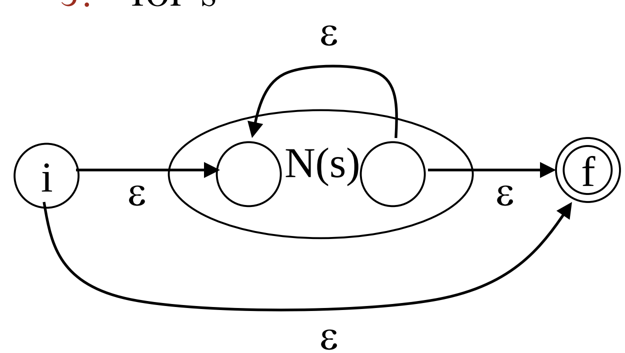
3. for s | t



4. for st

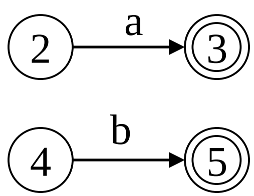


5. for s*

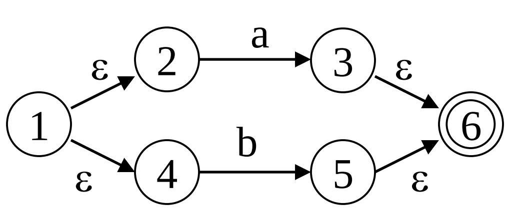


From a Regular Expression to an NFA

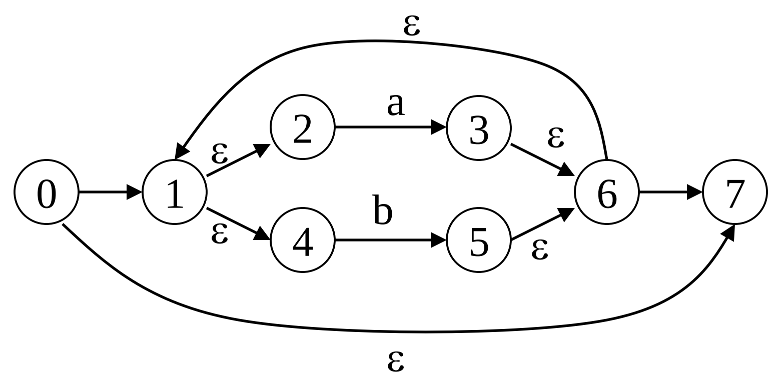
- Example: (a|b)*abb
  - for a and b



  - for a | b



- for (a|b)*



- for (a|b)*abb

see M4

Time-Space Tradeoffs

- NFA
  - space : $O(|r|)$
  - time : $O(|r| * |x|)$

- DFA
  - space: $O(2^{|r|})$
  - time: $O(|x|)$

x: input string

r: regular expression

Optimizing DFA

1. Construct an initial partition Π of the set of states with two groups: accepting states F and non-accepting states S-F
2. Apply the right procedure to construct new partition Π_{new}
3. If $\Pi_{new} = \Pi$, let $\Pi_{final} = \Pi$ and continue with step (4), otherwise, repeat step (2) with $\Pi = \Pi_{new}$
4. Choose representative states in each group
5. Remove dead states

- Construction of Π_{new}

for each group G of Π do

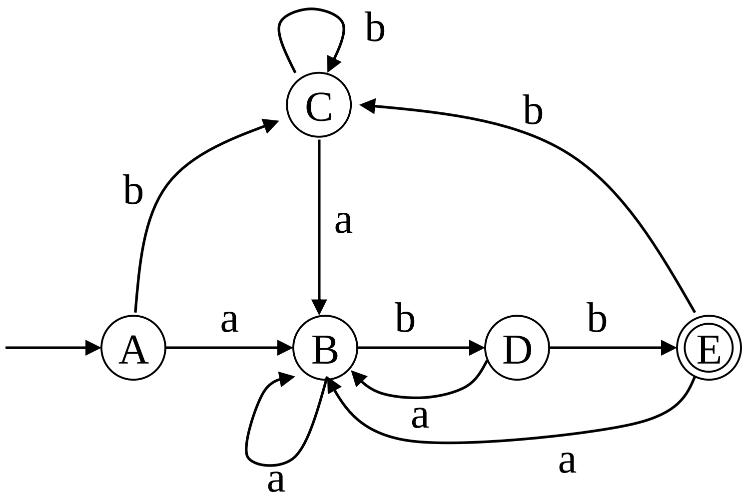
partition G into subgroups such that two states s and t of G are in the same subgroup if and only if for all input symbols a, state s and t have transitions on a to states in the same group of Π ;

replace G in Π_{new} by the set of all subgroups formed

end

Optimizing DFA

- Example (slide 13)



M5

- Initial Partition

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

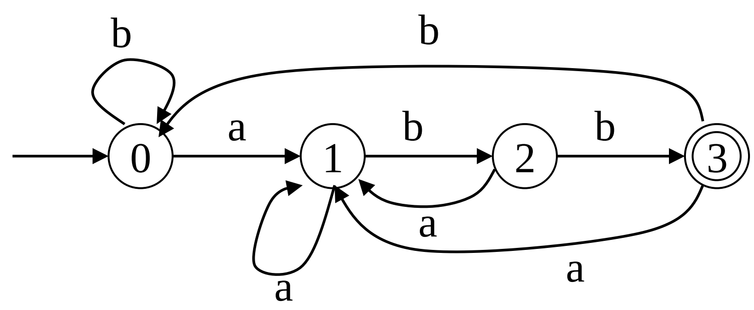
- Next Partitions

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

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

- Final States

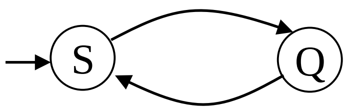
  0,1,2,3

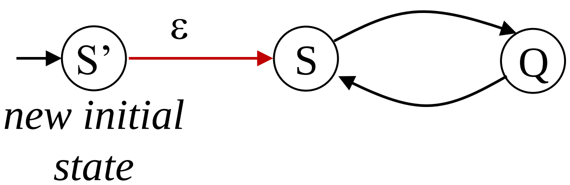


M3

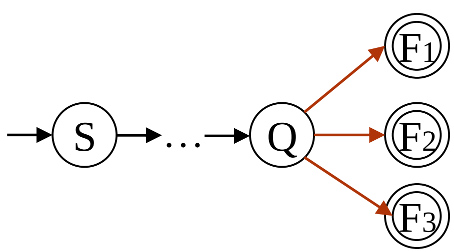
Converting DFA to Regular Expressions

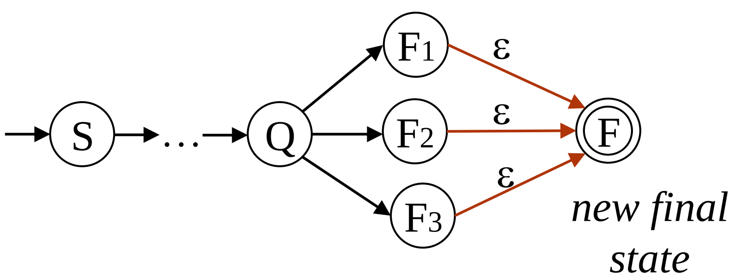
- State Elimination Method
  - Rule 1: *There should not be any incoming edge to the initial state*
    - If there are incoming edges to the initial state, create a new initial state with no incoming edges.





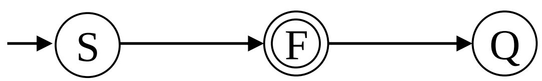
  - Rule 2: *There must be only one final state.*
    - If there is more than one final state, convert all final states to non-final states and create a new single final state.
  -

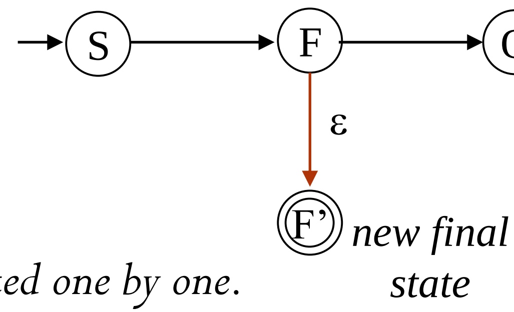




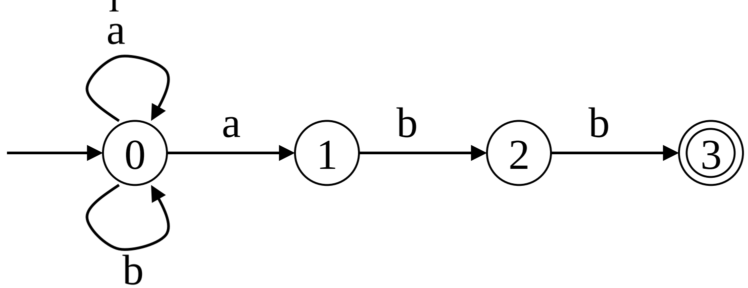
Converting DFA to Regular Expressions

- State Elimination Method
  - Rule 3: *From the final state, there should be no outgoing edges.*
    - If there are outgoing edges from the final state, convert all final states to non-final states and create a new final state having no outgoing edges.





  - Rule 4: *All intermediate states are eliminated one by one.*
    - Example: M1



M1: (a|b)*abb

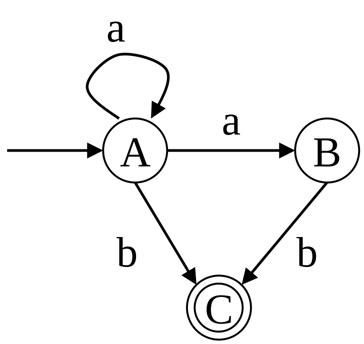
Converting DFA to Regular Expressions

- Arden’s Rule
  - to resolve recursions

$X = AX + B$ $\rightarrow X = A^*B$

$X = XA + B$ $\rightarrow X = BA^*$

- Example 1



(1) $A = aA + aB + bC$

(2) $B = bC$

(3) $C = \lambda$ (final state)

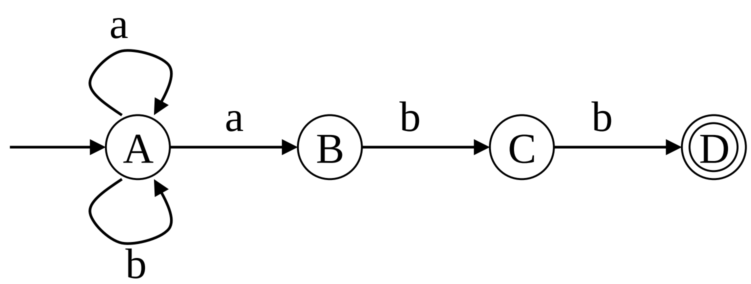
(4) $B = b$ 1. substituting (3) into (2)

(5) $A = aA + bb + b$ 2. substituting (4) into (1)

(6) $A = a^*(bb+b)$ 3. by Arden rule

Converting DFA to Regular Expressions

- Example 2



M1: (a|b)*abb

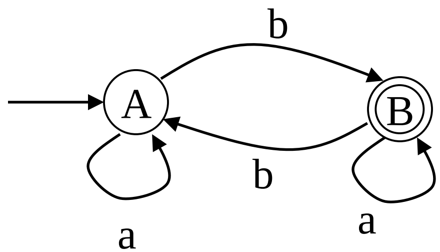
(1) A = aA + bA +aB
(2) B = bC
(3) C = bD
(4) D = λ

(5) C = b
(6) B = bb
(7) A = (a+b)A + abb
(8) A = (a+b)*abb
== (a | b)*abb

1. substituting (4) into (3)
2. substituting (5) into (2)
3. substituting (6) into (1)
4. by Arden's rule

Converting DFA to Regular Expressions

- Example 3



(1) $A = aA + bB$

(2) $B = bA + aB + \lambda$

(3) $B = aB + (bA + \lambda)$	1. by (2)
(4) $B = a^*(bA + \lambda) = a^*bA + a^*$	2. by Arden's rule
(5) $A = aA + b(a^*bA+a^*) = aA + ba^*bA + ba^*$	3. substituting (4) into (1)
$\quad = (a+ba^*b)A + ba^*$	
(6) $A = (a+ba^*b)^*ba^* = (a \mid ba^*b)^*ba^*$	4. by Arden's rule