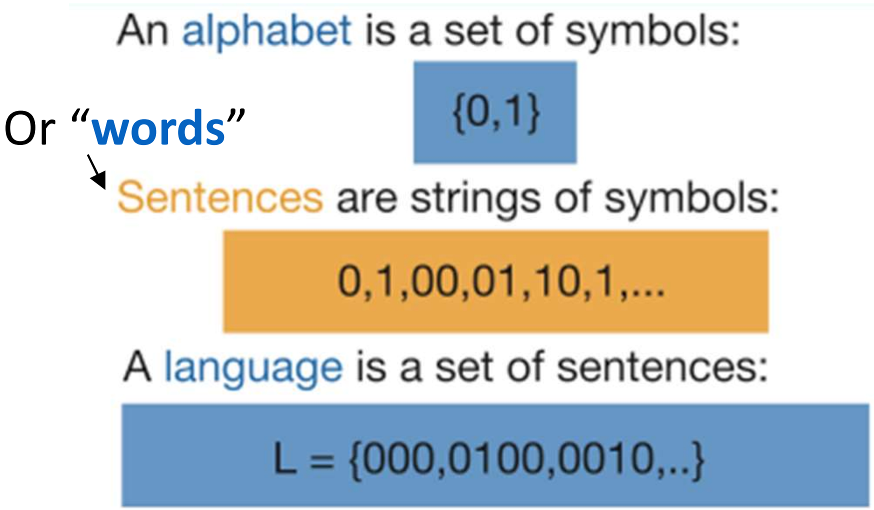
Data Engineering

Lecture 5: Regular expressions and Automatas

Languages



- Languages: *“A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols”*
- *N. Chomsky, Information and Control, Vol 2, 1959*

Alphabet

An alphabet is a finite, non-empty set of symbols

- We will use the symbol A to denote an alphabet
- Examples:
  - Binary: A = {0,1}
  - All lower case letters: A = {a,b,c,..z}
  - Alphanumeric: A = {a-z, A-Z, 0-9}
  - DNA molecule letters: A = {a,c,g,t}
  - …

Strings

A string or word is a finite sequence of symbols chosen from ∑

- ***Empty string is ε (or "epsilon")***

- Length of a string *w,* denoted by "|*w*|", is equal to the *number of (non- ε) characters in the string*
  - *E.g., x = 010100* *|x| = 6*
  - *x = 01 ε 0 ε 1 ε 00 ε* *|x| = ?*

Operations

Given two strings x and y, the following operations over them are defined:

- Concatenation: xy

- Alternation: x|y

- Kleene star: x* denotes the smallest superset of the set described by *x* that contains ε and is closed under string concatenation.

  (i.e. It is the set of all strings that can be made by concatenating any finite number of elements in x.)

Note: To avoid extra parentheses it is assumed the priority:

Kleene star > concatenation > alternation.

Regular expression

A regular expression (RE) over an alphabet A is defined recursively as follows:

1) ε is a regular expression.
2) Each symbol of A is a regular expression.
3) If e1 and e2 are regular expressions.
   - (e1) | (e2) is a regular expression.
   - (e1) (e2) is a regular expression.
   - (e1)* is a regular expression.
4) There are not other regular expressions that the ones constructed with rules (1)-(3).

Regular expression

A regular expression (RE) over an alphabet A is defined recursively as follows:

1) ε is a regular expression.
2) Each symbol of A is a regular expression.
3) If e1 and e2 are regular expressions.
   - (e1) | (e2) is a regular expression.
   - (e1) (e2) is a regular expression.
   - (e1)* is a regular expression.
4) There are not other regular expressions that the ones constructed with rules (1)-(3).

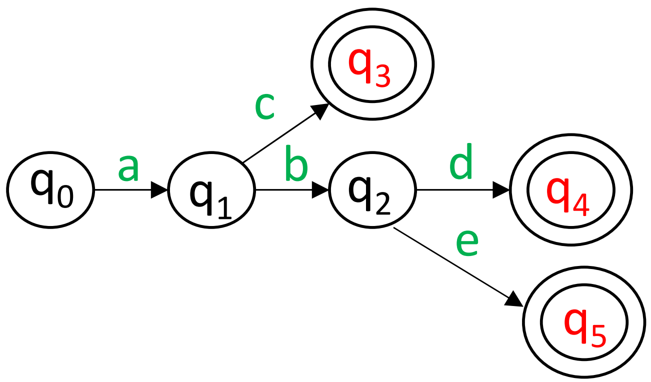
Example: The regular expression (a|b)*abb
can generate
{abb, aabb, baabb, aaabb, ababb, bbabb, aaaabb,...}

* Kleene star

Finite Automata

A finite automata is a quintuple $M=(Q,A, D ,q_0,F)$

- Q es is a finite set -set of states-.
- A is an alphabet –Input alphabet-.
- D is an applciation $D : Q \times A \rightarrow Q$ (given an state and a symbol from the alphabet produces a new state).
- q_0 is an element of Q, -initial state-.
- F is a subset of Q –set of final states-.



$Q=\{q_0, q_1, q_2, q_3, q_4, q_5\}$

$A=\{a, b, c, d, e\}$

$D(q_0,a)= q_1; D(q_1,b)= q_2; D(q_1,c)= q_3 \ldots$

$F=\{q_3, q_4, q_5\}$

Important

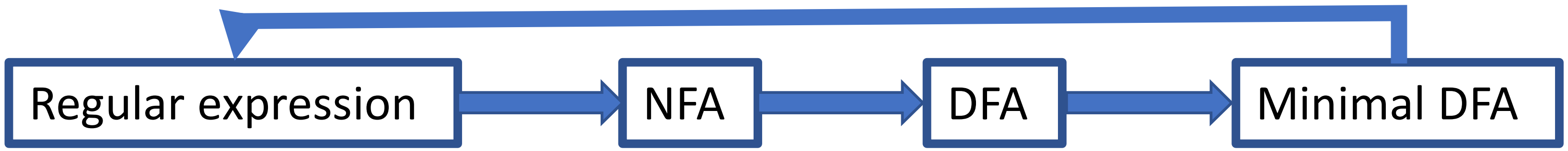
Given a regular expression it exists a finite automata able to recognize its language.

(Also, given a finite automata, it can be expressed as a regular expression).

Today goals

- Understand the conversion between regular expressions and Non-Deterministic Finite Automatas (NFA).
- Understand the conversion between NFA y Deterministic Finite Automatas (DFA).
- Being able to convert a regular expression into a DFA.
- Understand the conversión between a NFA and a minimal DFA.

Planing



RE → NFA *(Thompson construction)*

Build a **NFA** for each term in the **RE**

Combine them following the patterns marked by the operators.

NFA → DFA *(Subset construction)*

Build a **DFA** that simulates the **NFA**

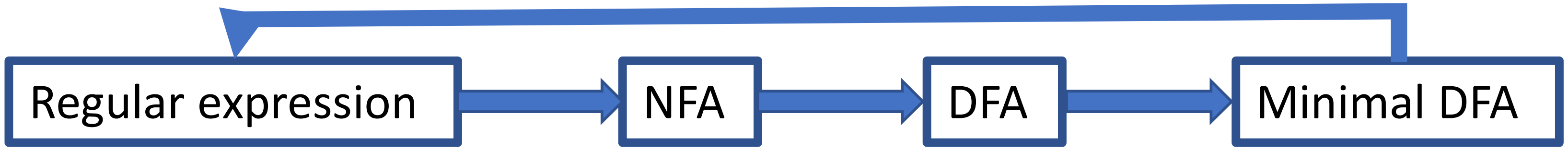
DFA → Minimal **DFA**

Brzozowski algorithm's

DFA → RE

Join all the paths from s_0 to a final state

Planing



RE → NFA *(Thompson construction)*

Build a **NFA** for each term in the **RE**

Combine them following the patterns marked by the operators.

NFA → DFA *(Subset construction)*

Build a **DFA** that simulates the **NFA**

DFA → Minimal **DFA**

Brzozowski algorithm's

DFA → RE

Join all the paths from s_0 to a final state

RE → NFA (Thompson)

Two stepes:

1) For each symbol and operators, there is a **NFA** pattern.

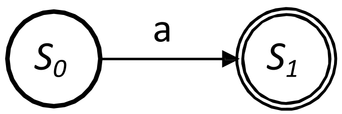
2) We join each of these patterns with ε-transitions in precedent order and we adjust the final states.

RE → NFA (Thompson)

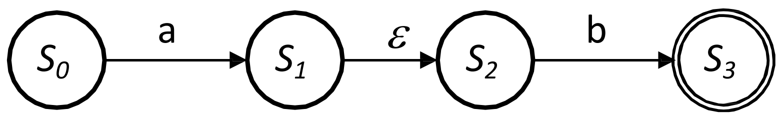
Two stepes:

1) For each symbol and operators, there is a **NFA** pattern.

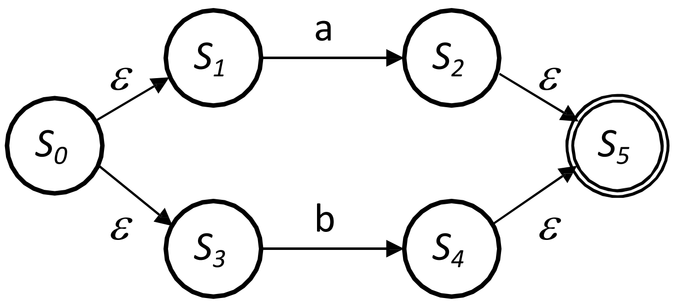
2) We join each of these patterns with ε-transitions in precedent order and we adjust the final states.



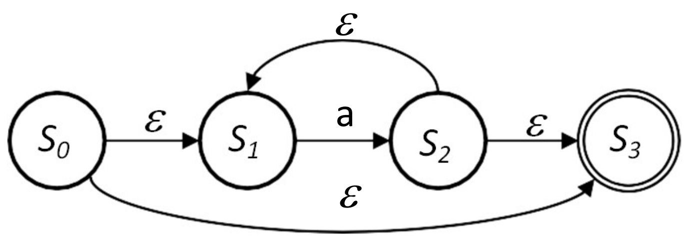
NFA for a



NFA for ab



NFA for a | b



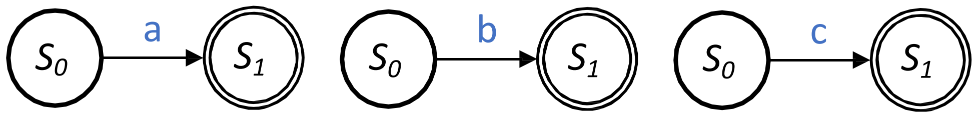
NFA for a^*

<u>**IMPORTANT:**</u> Remember the preference in REs -1) Parenthesis, 2) Kleene star, 3) concatenation, 4) alternation-.

RE → NFA –example 1-

Build a NFA for a (b | c)* :

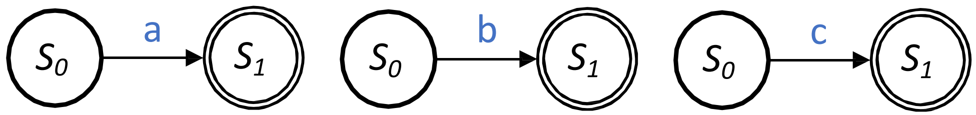
Step 1: Build the patterns for a, b, c



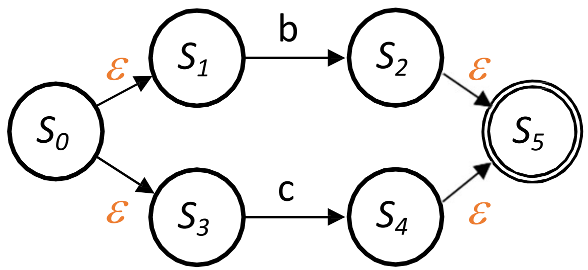
RE → NFA –example 1-

Build a NFA for a (b | c)* :

Step 1: Build the patterns for a, b, c



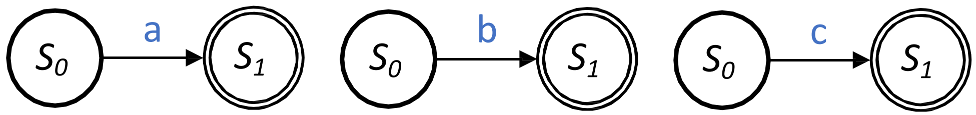
Step 2: Join the elements and add the ε transitions to build b | c



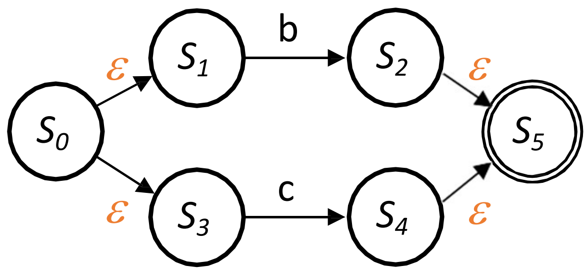
RE → NFA –example 1-

Build a NFA for a (b | c)* :

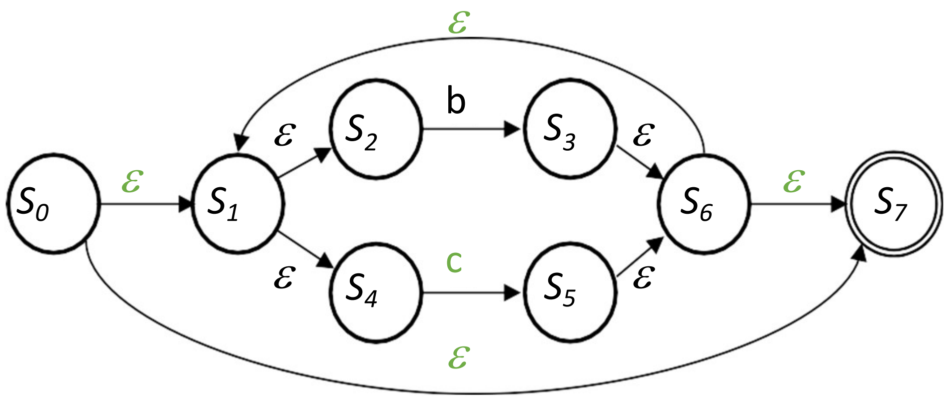
Step 1: Build the patterns for a, b, c



Step 2: Join the elements and add the ε transitions to build b | c



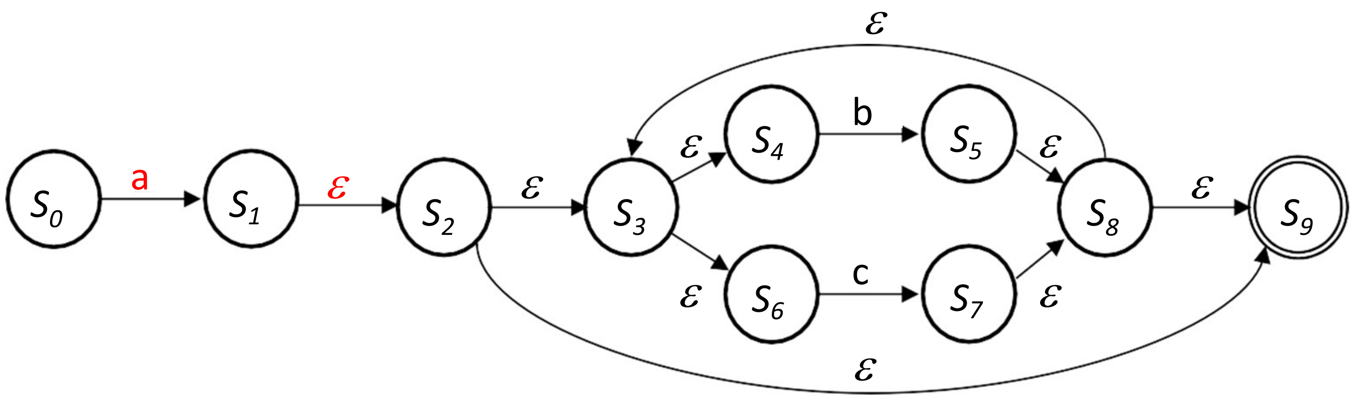
Steo 3: Join the elements and add the ε transitions to build (b | c)*



RE → NFA –example 1-

Build a NFA for a (b | c)* :

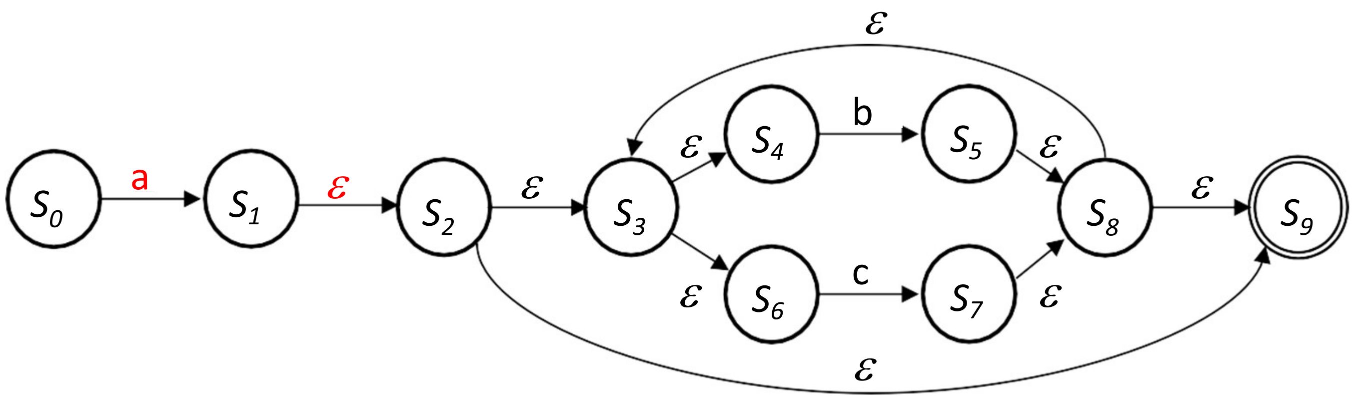
Step 4: Join the elements and add the ε transitions to build a (b | c)*



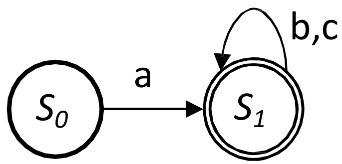
RE → NFA –example 1-

Build a NFA for a (b | c)* :

Step 4: Join the elements and add the ε transitions to build a (b | c)*



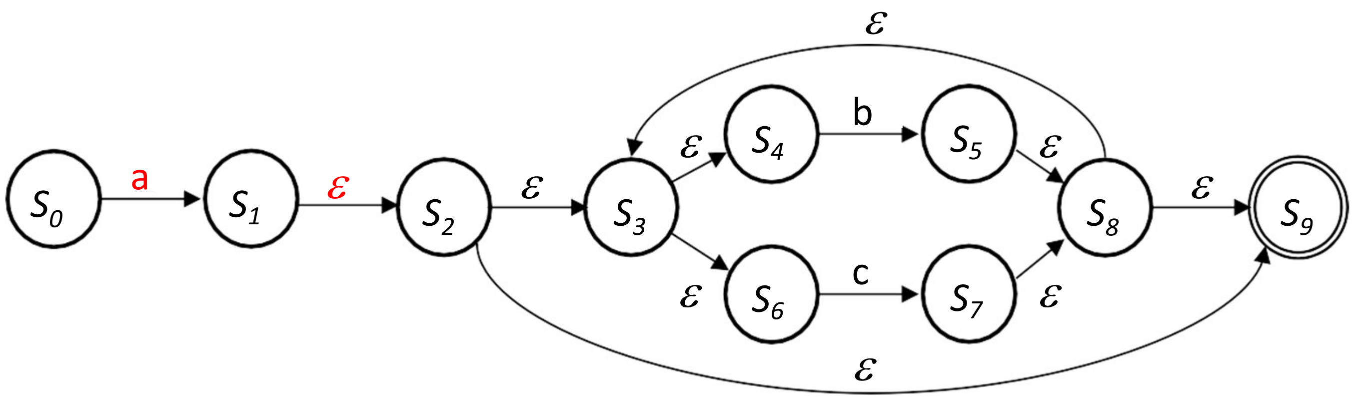
Note: In an easy case like this, we would have probably designed something like this in a direct way.



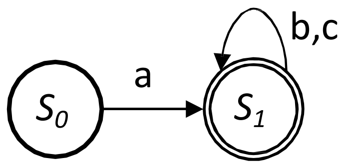
RE → NFA –example 1-

Build a NFA for a (b | c)* :

Step 4: Join the elements and add the ε transitions to build a (b | c)*

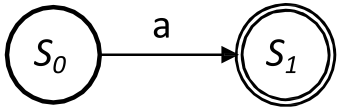


Note: In an easy case like this, we would have probably designed something like this in a direct way.

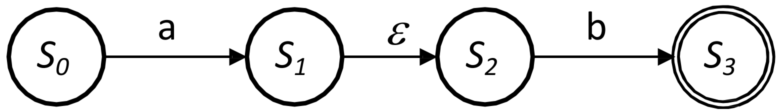


But, don’t do it!
Always follow the steps. Learn the 4 basis patterns and apply them!

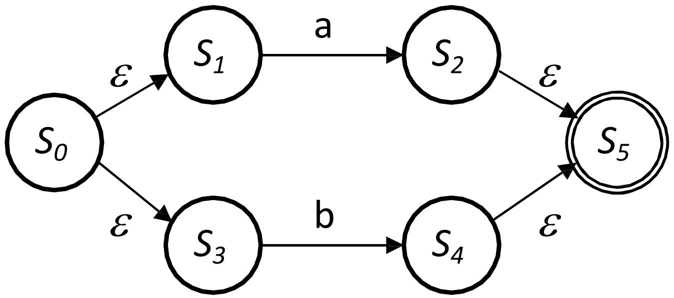
Always remember:



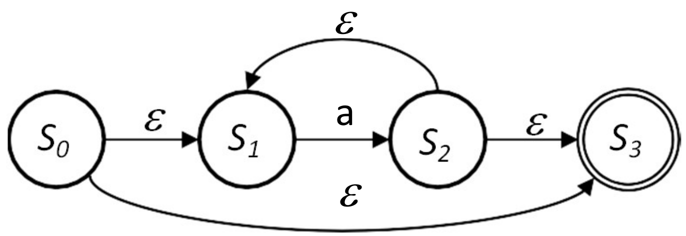
NFA for a



NFA for ab



NFA for a | b



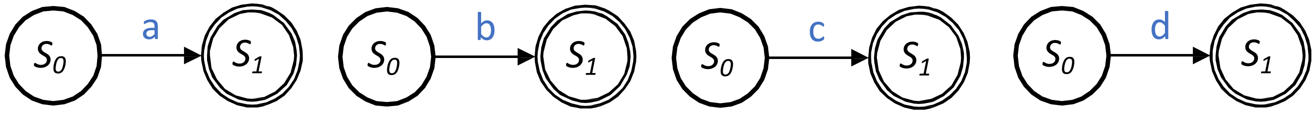
NFA for a^*

<u>IMPORTANT:</u> Remember the preference in REs -1) Parenthesis, 2) Kleene star, 3) concatenation, 4) alternation-.

RE → NFA –example 2-

Build a NFA for ((ab)*| c)d :

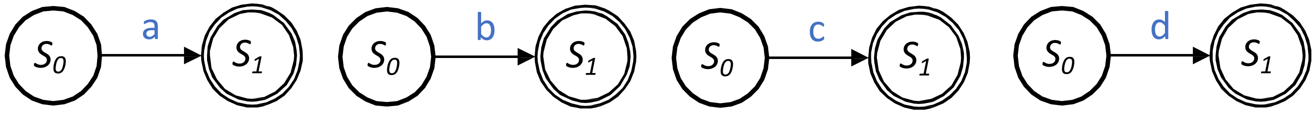
Step 1: Build the patterns for a, b, c, d



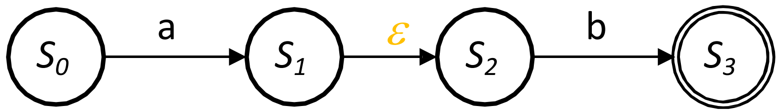
RE → NFA –example 2-

Build a NFA for ((ab)*| c)d :

Step 1: Build the patterns for a, b, c, d



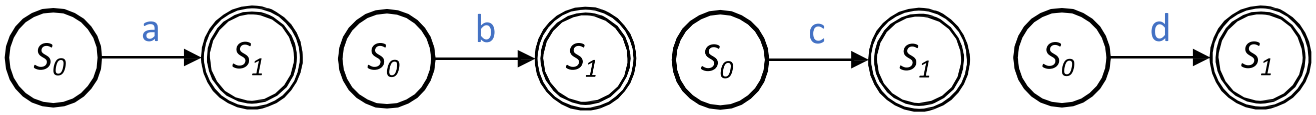
Step 2: ab



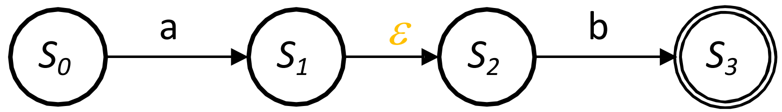
RE → NFA –example 2-

Build a NFA for ((ab)*| c)d :

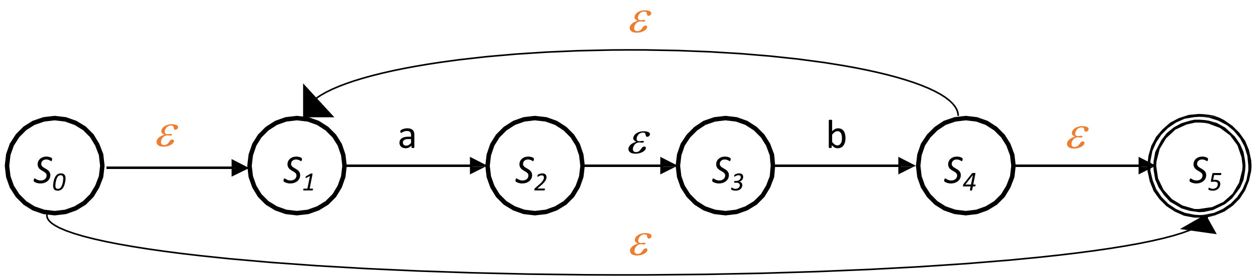
Step 1: Build the patterns for a, b, c, d



Step 2: ab



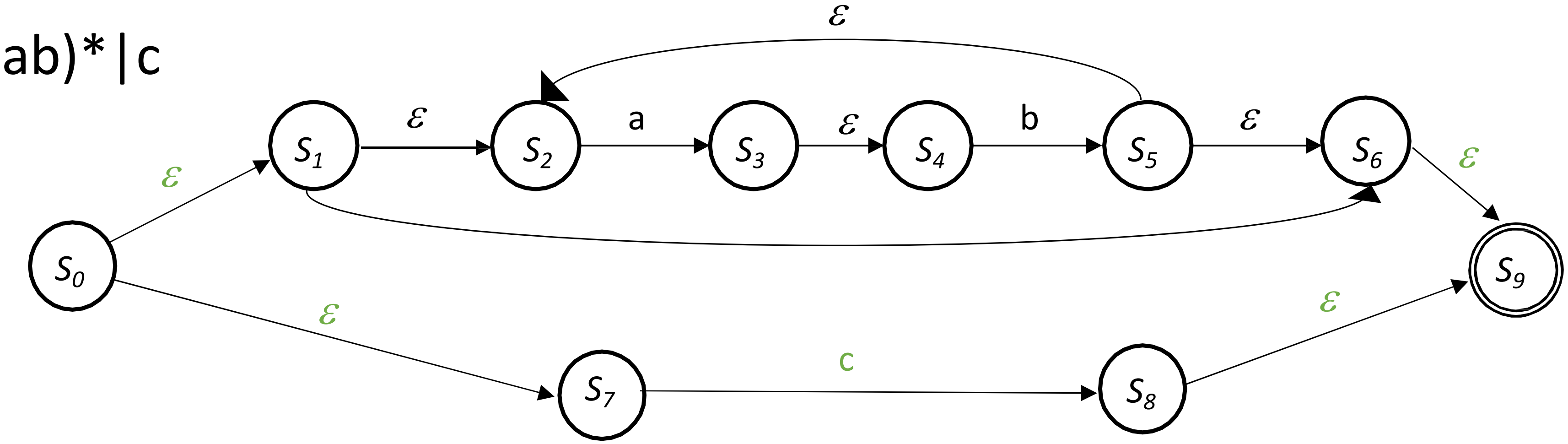
Step 3: (ab)*



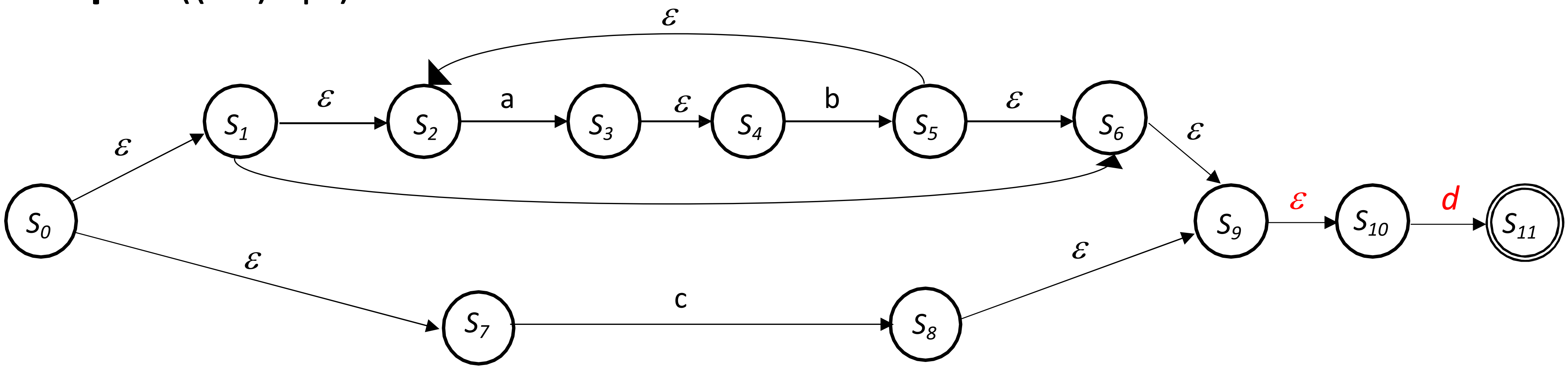
RE → NFA –example 2-

Build a NFA for ((ab)*| c)d :

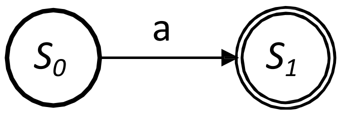
Step 4: (ab)*|c



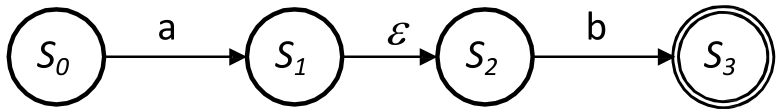
Step 5: ((ab)*|c)d



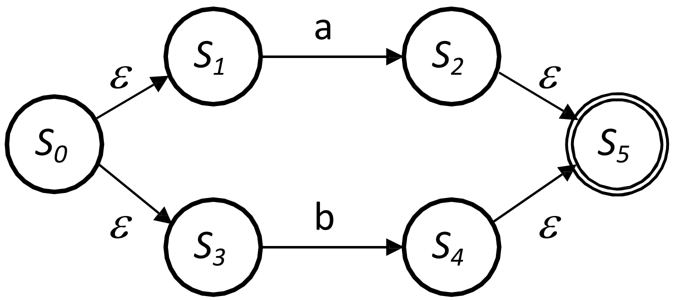
Always remember:



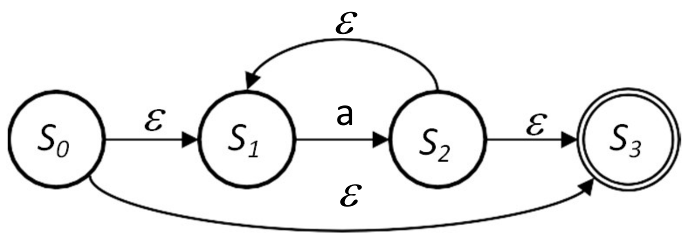
NFA for a



NFA for ab



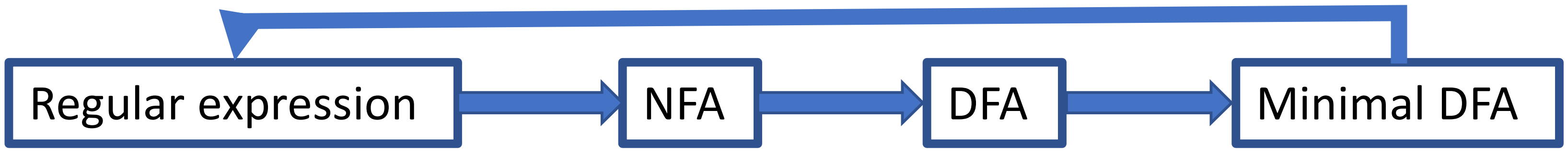
NFA for a | b



NFA for a*

<u>**IMPORTANT:**</u> Remember the preference in REs -1) Parenthesis, 2) Kleene star, 3) concatenation, 4) alternation-.

Planing



RE → NFA *(Thompson construction)*

Build a **NFA** for each term in the **RE**

Combine them following the patterns marked by the operators.

NFA → DFA (*Subset construction*)

Build a **DFA** that simulates the **NFA**

DFA → Minimal **DFA**

Brzozowski algorithm's

DFA → RE

Join all the paths from s_0 to a final state

NFA → DFA: why?

A DFA is a special case of NFA such as:

- DFA does not have ε transitions
- Transiciones in DFAs have a single value

NFA → DFA: why?

A DFA is a special case of NFA such as:

- DFA does not have ε transitions
- Transiciones in DFAs have a single value

We start building the NFA, as it has a more obvious relation with the Regular Expression (RE).

NFA → DFA: why?

A DFA is a special case of NFA such as:

- DFA does not have ε transitions
- Transiciones in DFAs have a single value

We start building the NFA, as it has a more obvious relation with the Regular Expression (RE).

In Thompson construction, the ε transitions were joining two **DFA**s to create a **NFA.**

NFA → DFA: why?

A DFA is a special case of NFA such as:

- DFA does not have ε transitions
- Transiciones in DFAs have a single value

We start building the NFA, as it has a more obvious relation with the Regular Expression (RE).

In Thompson construction, the ε transitions were joining two **DFA**s to create a **NFA.**

Luckily: Any NFA can be simulated by a DFA.

NFA → DFA: Subset construction

The subset construction createa a DFA that simulates a given NFA

Two basic functions:

- *Move*(s_i, *a*) is the set of states that can be reached from s_i applying *a*
- *FollowEpsilon*(s_i) is the set of states that can be reached from s_i applying ε

NFA → DFA: Subset construction

The subset construction createa a DFA that simulates a given NFA

Algorithm:

1) Derive the initial state of the **DFA** from the state n_0 of the **NFA**

    1a) Add all the states that can be reached from n_0 applying ε:

    $d_0 = FollowEpsilon(\{n_0\})$

    We define $\boldsymbol{D} = \{ d_0 \}$

NFA → DFA: Subset construction

The subset construction createa a DFA that simulates a given NFA

Algorithm:

1) Derive the initial state of the **DFA** from the state n_0 of the **NFA**

    1a) Add all the states that can be reached from n_0 applying ε:

    d_0 = *FollowEpsilon* ({n_0})

    We define ***D*** = { d_0 }

2) For $\alpha \in \Sigma$ (alphabet), compute *FollowEpsilon* (*Move*(d_0, α)):

    2a) If this operation defined a new state, we add it to ***D***

NFA → DFA: Subset construction

The subset construction createa a DFA that simulates a given NFA

Algorithm:

1) Derive the initial state of the **DFA** from the state n_0 of the **NFA**

   1a) Add all the states that can be reached from n_0 applying ε:

   d_0 = *FollowEpsilon* ({n_0})

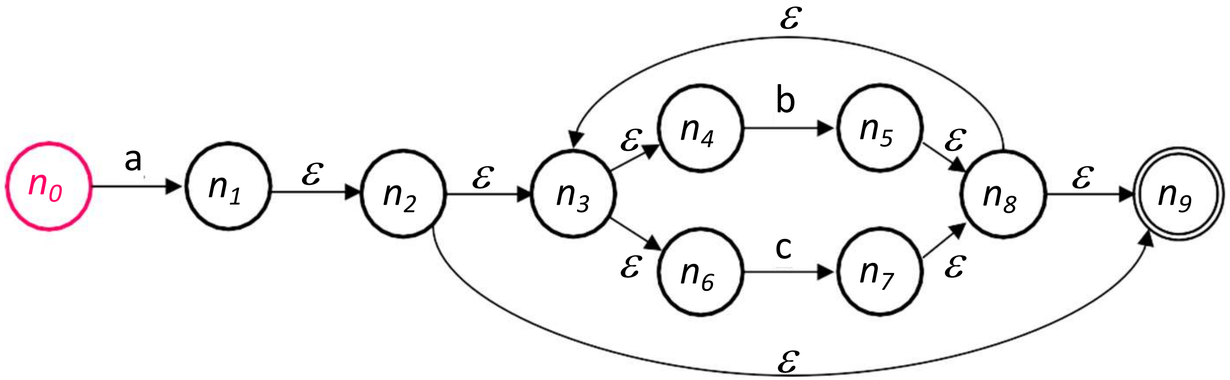
   We define $\boldsymbol{D} = \{ d_0 \}$

2) For $\alpha \in \Sigma$ (alphabet), compute *FollowEpsilon* (*Move*(d_0, α)):

   2a) If this operation defined a new state, we add it to ***D***

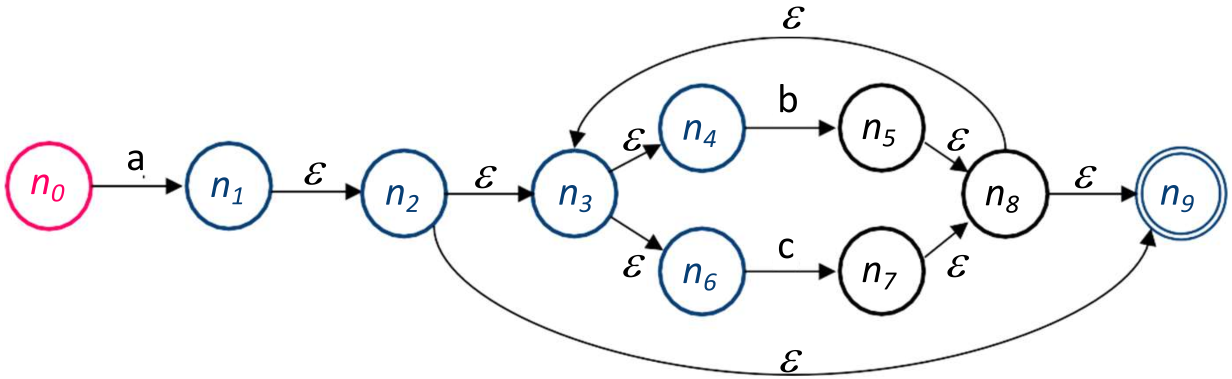
3) Iterate until it is not possible to add any new state.

NFA → DFA –example 1-



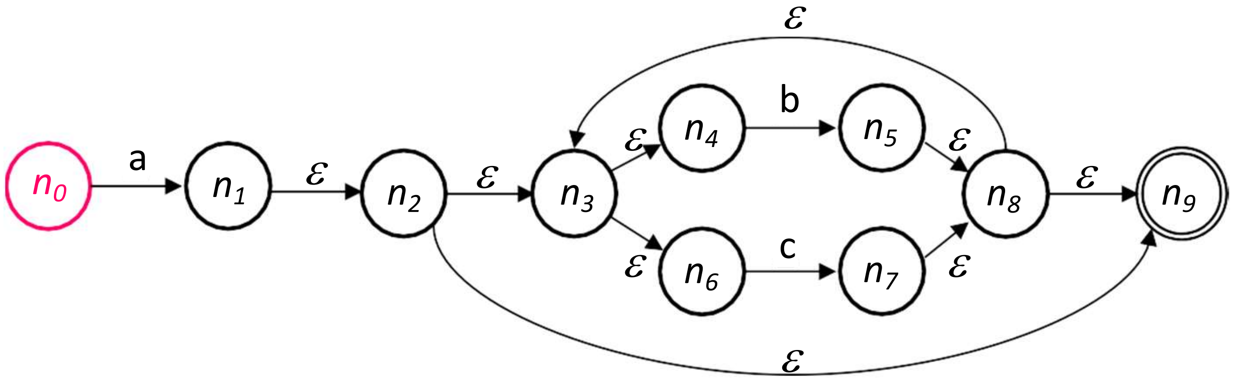
States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0			

NFA → DFA –example 1-



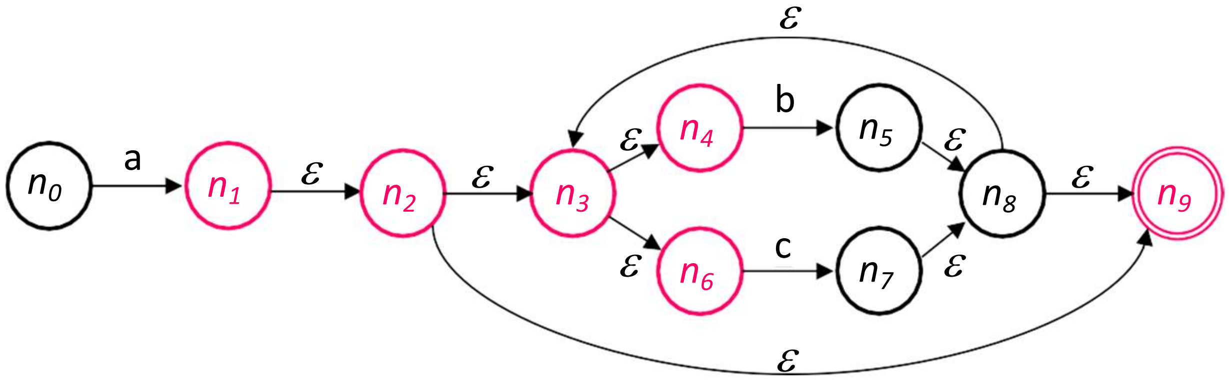
States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\ n_2\ n_3\ n_4\ n_6\ n_9$		

NFA → DFA –example 1-



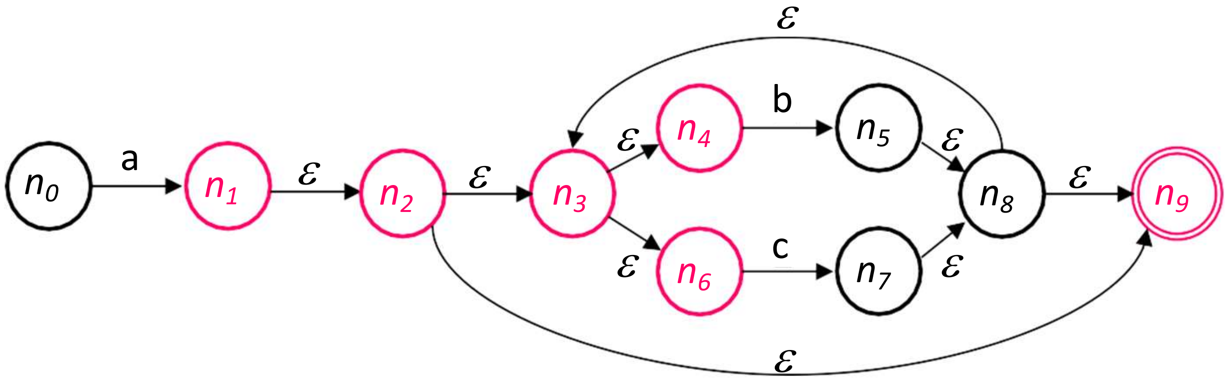
States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\ n_2\ n_3$ $n_4\ n_6\ n_9$	None	None

NFA → DFA –example 1-



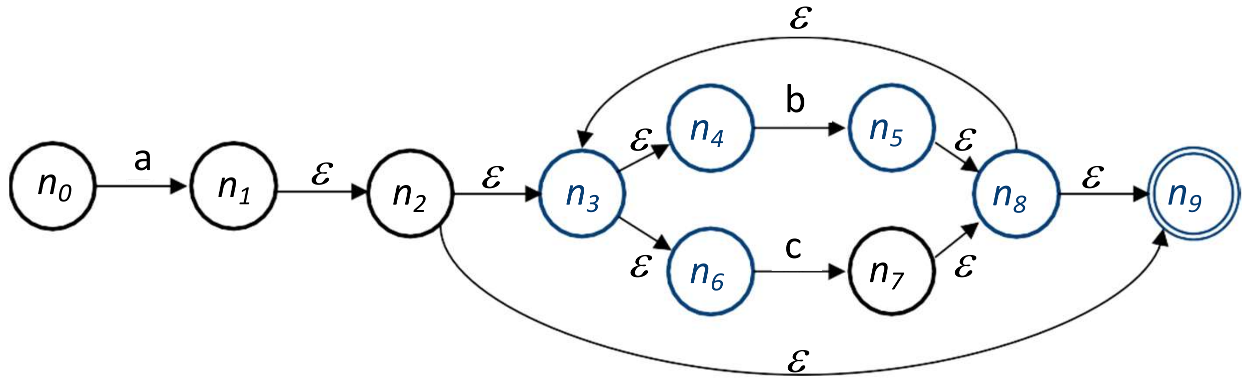
States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\, n_2\, n_3\, n_4\, n_6\, n_9$	None	None
d_1	$n_1\, n_2\, n_3\, n_4\, n_6\, n_9$			

NFA → DFA –example 1-



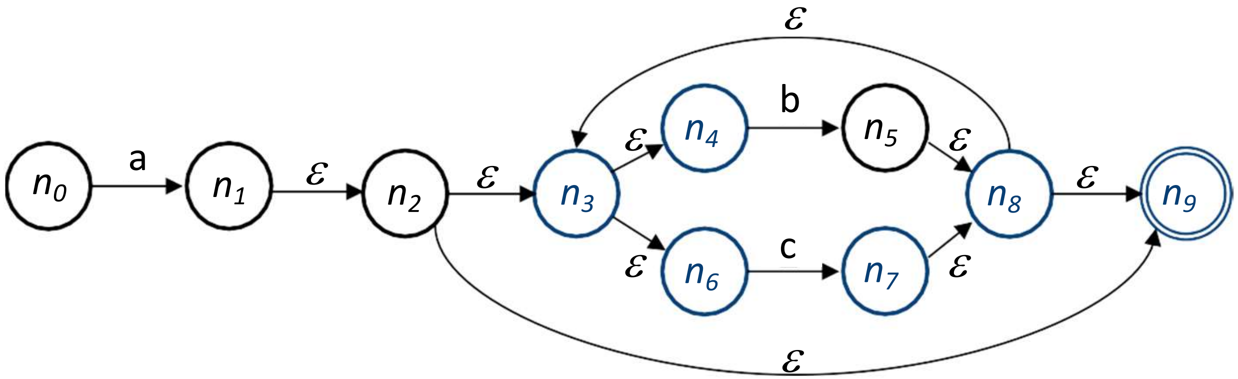
States		***FollowEpsilon* (*Move*(*s,**)**		
DFA	NFA	a	b	c
d_0	n_0	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	None
d_1	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None		

NFA → DFA –example 1-



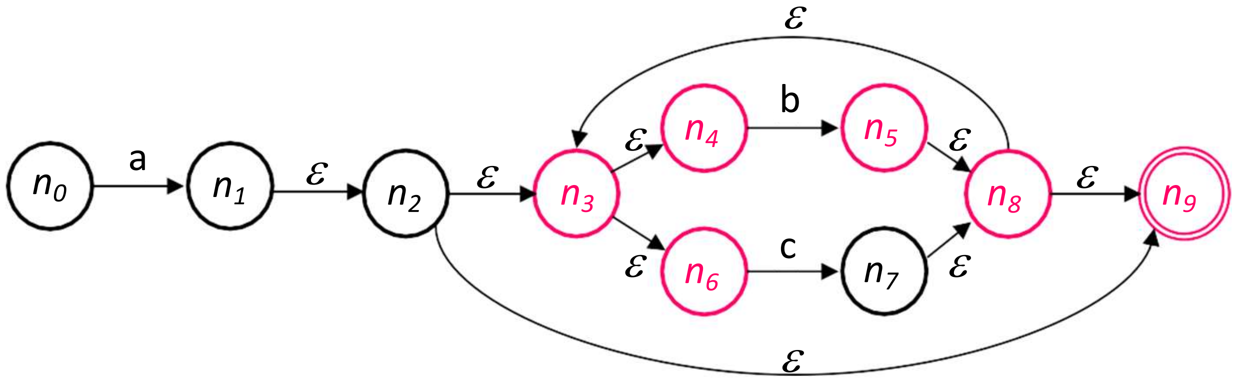
States		***FollowEpsilon* (*Move*(*s,**)**		
DFA	NFA	a	b	c
d_0	n_0	$n_1\ n_2\ n_3$ $n_4\ n_6\ n_9$	None	None
d_1	$n_1\ n_2\ n_3$ $n_4\ n_6\ n_9$	None	$n_5\ n_8\ n_9$ $n_3\ n_4\ n_6$	

NFA → DFA –example 1-



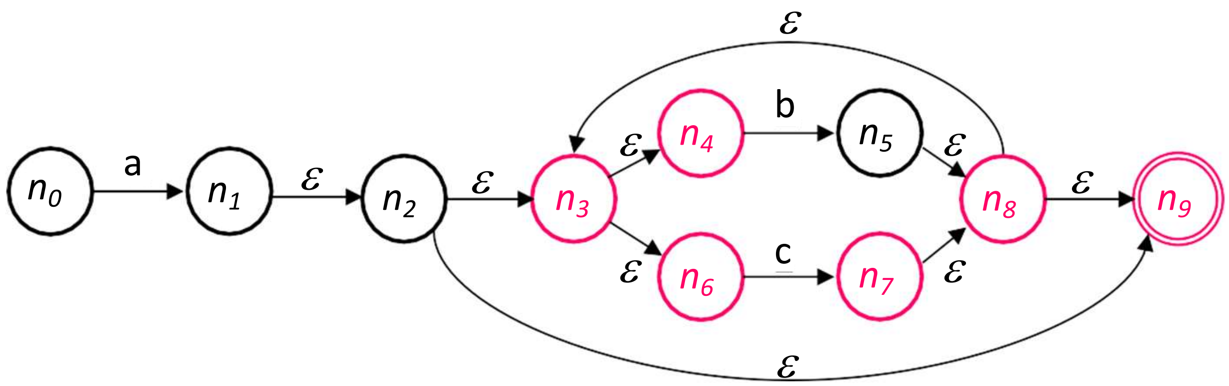
States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	None
d_1	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$

NFA → DFA –example 1-



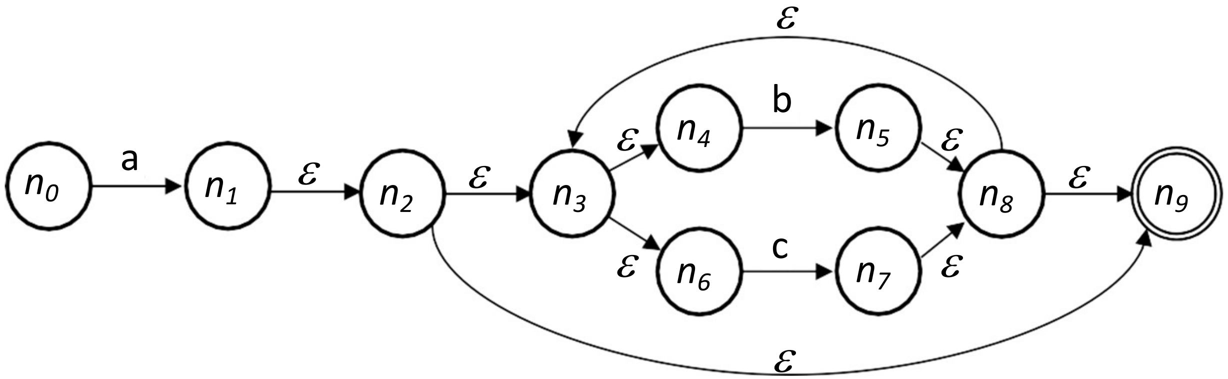
States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	None
d_1	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$
d_2	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$			

NFA → DFA –example 1-



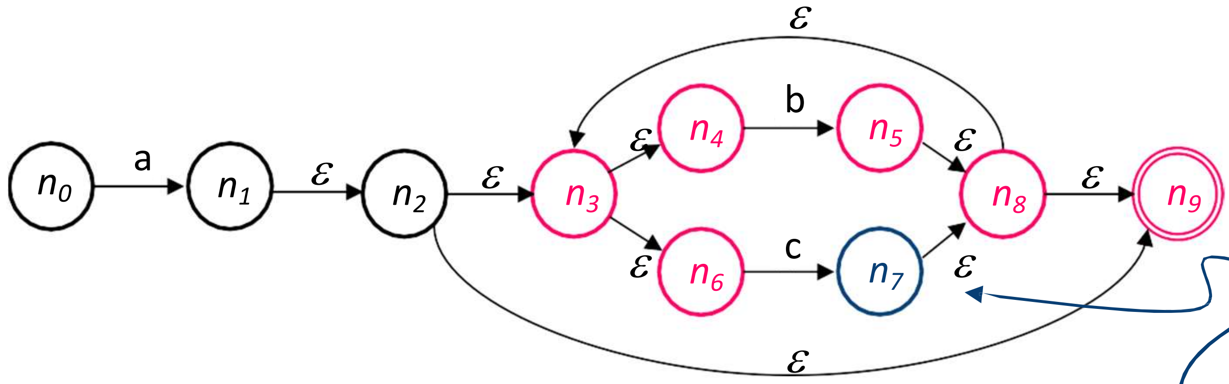
States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	None
d_1	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$
d_2	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$			
d_3	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$			

NFA → DFA –example 1-



States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	None
d_1	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$
d_2	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	None		
d_3	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$	None		

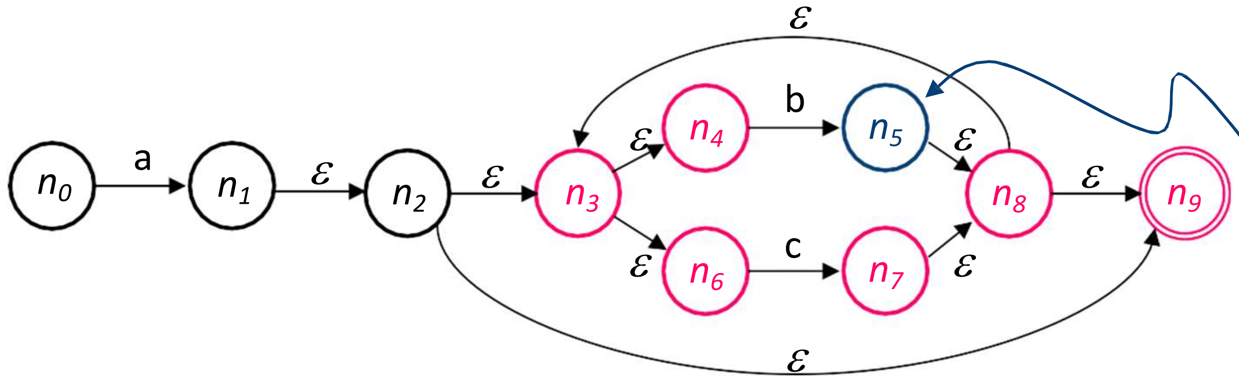
NFA → DFA –example 1-



n_7 is the "nucleous" state of d_3

States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	None
d_1	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$
d_2	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	None	d_2	d_3
d_3	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$	None		

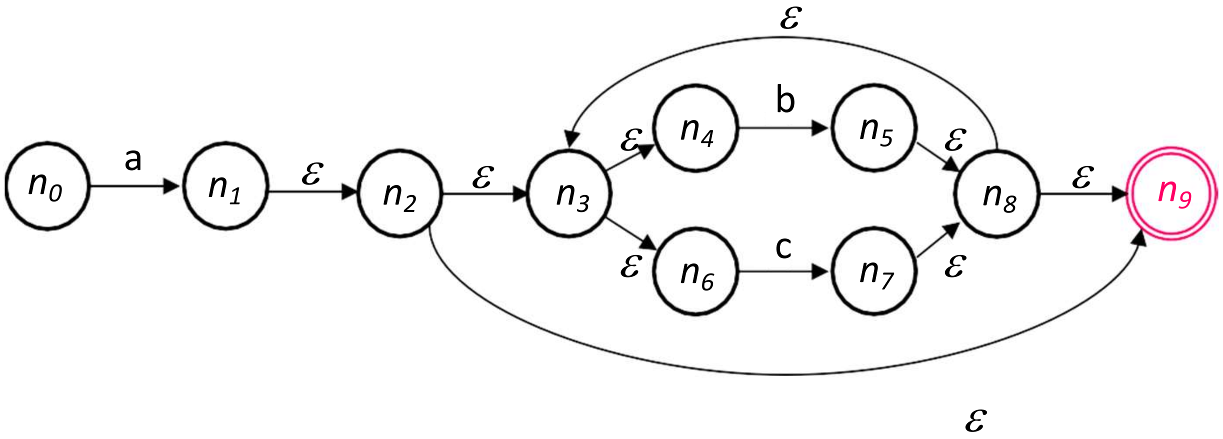
NFA → DFA –example 1-



n_5 is the "nucleous" state of d_2

States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	None
d_1	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$
d_2	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	None	d_2	d_3
d_3	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$	None	d_2	d_3

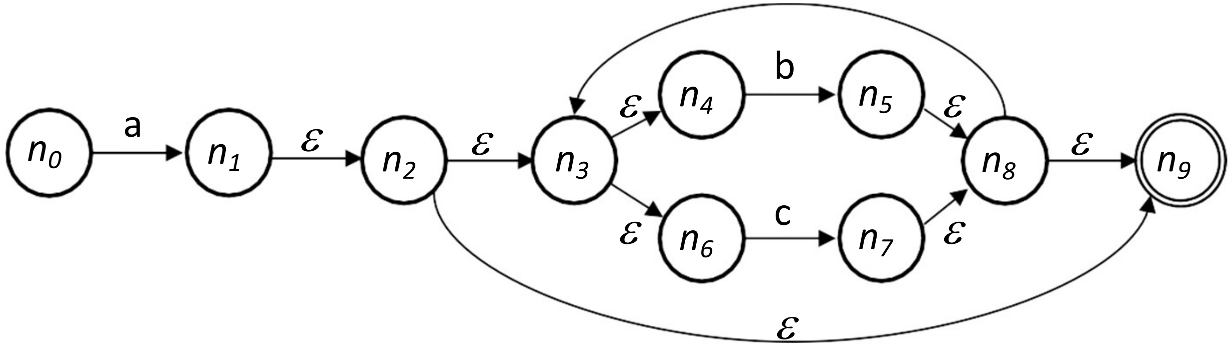
NFA → DFA –example 1-



States		***FollowEpsilon (Move(s,*)***		
DFA	NFA	a	b	c
d_0	n_0	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	None
d_1	$n_1\, n_2\, n_3$ $n_4\, n_6\, n_9$	None	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$
d_2	$n_5\, n_8\, n_9$ $n_3\, n_4\, n_6$	None	d_2	d_3
d_3	$n_7\, n_8\, n_9$ $n_3\, n_4\, n_6$	None	d_2	d_3

Final states (they contain n_9)

NFA → DFA –example 1-

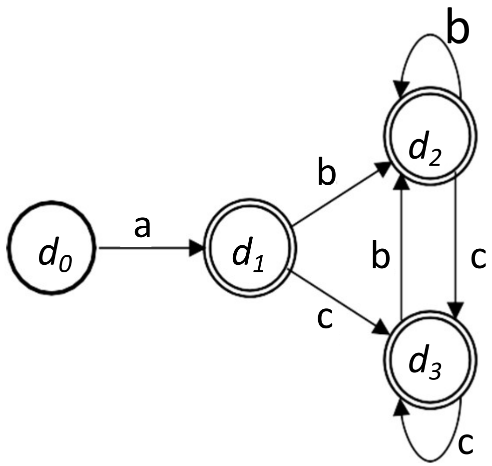


States		***FollowEpsilon* (*Move*(*s,**)**		
DFA	NFA	a	b	c
d_0	n_0	$n_1\,n_2\,n_3$ $n_4\,n_6\,n_9$	None	None
d_1	$n_1\,n_2\,n_3$ $n_4\,n_6\,n_9$	None	$n_5\,n_8\,n_9$ $n_3\,n_4\,n_6$	$n_7\,n_8\,n_9$ $n_3\,n_4\,n_6$
d_2	$n_5\,n_8\,n_9$ $n_3\,n_4\,n_6$	None	d_2	d_3
d_3	$n_7\,n_8\,n_9$ $n_3\,n_4\,n_6$	None	d_2	d_3

Transition Table for the DFA

NFA → DFA –example 1-

The **DFA for** a (b | c)* is

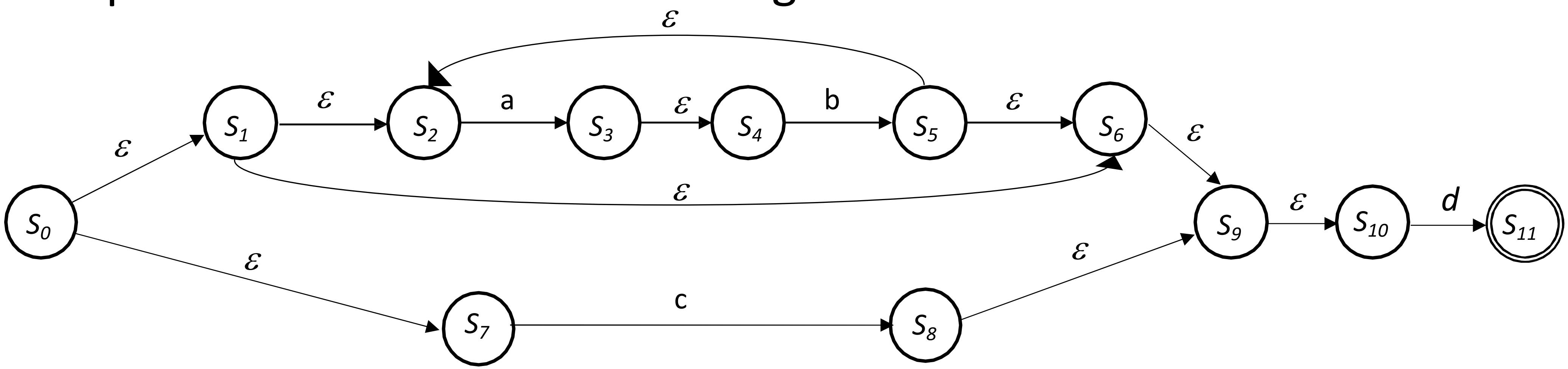


		a	b	c
d_0		d_1	None	None
d_1		None	d_2	d_3
d_2		None	d_2	d_3
d_3		None	d_2	d_3

- Much smaller than the **NFA** (no ε transitions).
- All transitions are deterministic.
- The skeleton is similar to that of the NFA

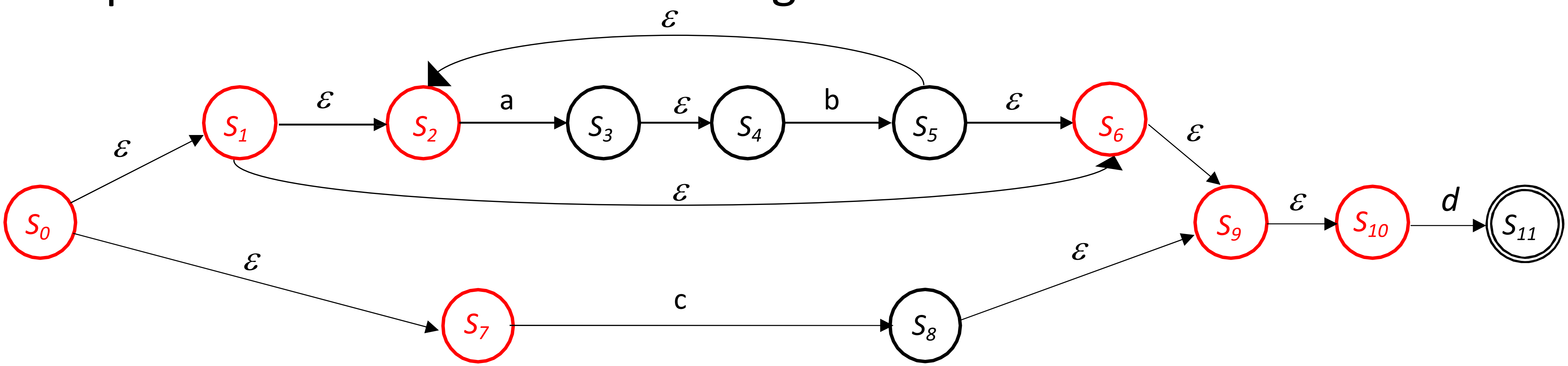
NFA → DFA –example 2-

Compute the DFA from the following NFA:



NFA → DFA –example 2-

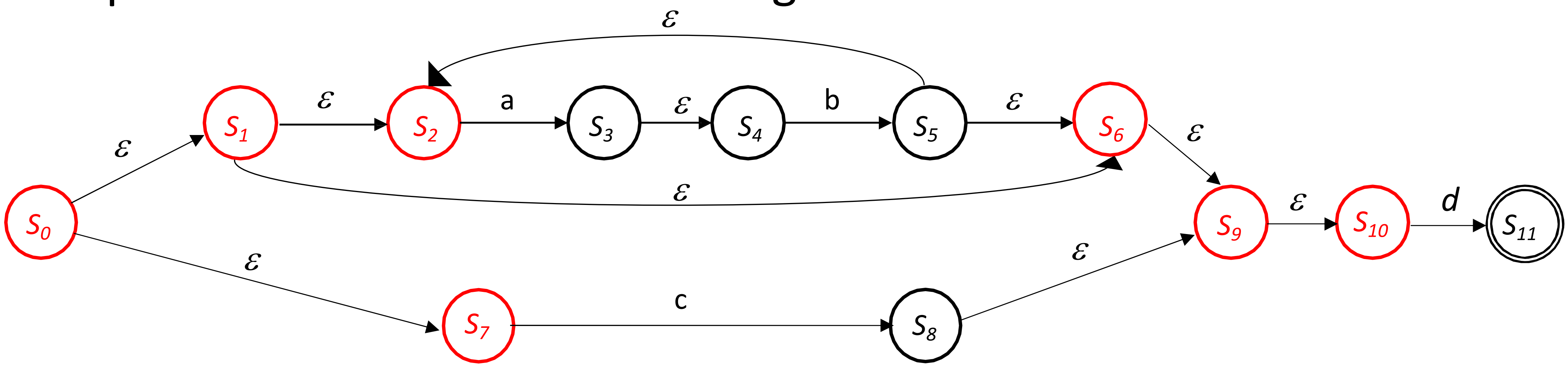
Compute the DFA from the following NFA:



States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0\,S_1\,S_2\,S_3$ $S_7\,S_9\,S_{10}$				

NFA → DFA –example 2-

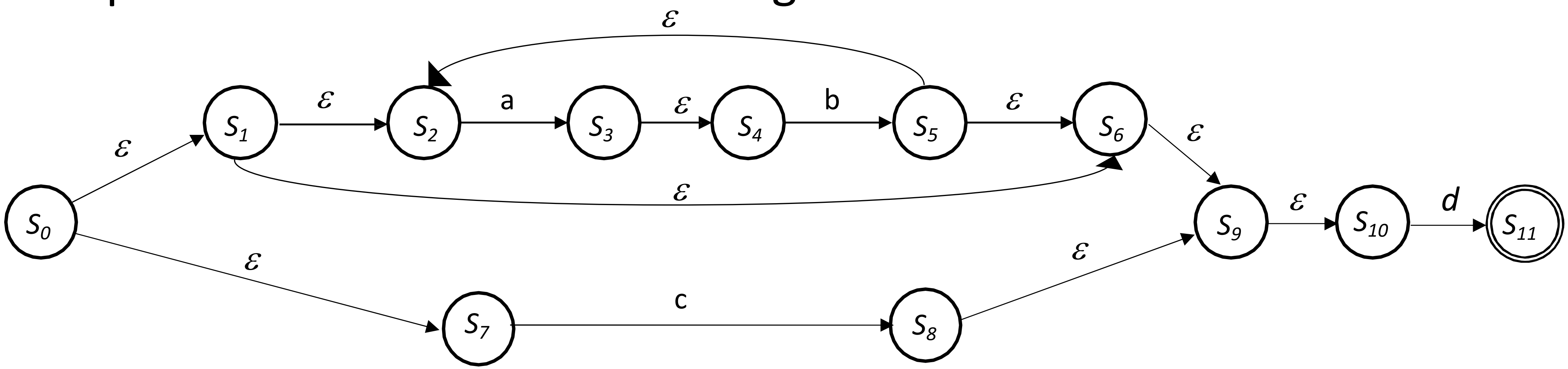
Compute the DFA from the following NFA:



States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0\,S_1\,S_2\,S_3$ $S_7\,S_9\,S_{10}$	$S_3\,S_4$	None	$S_8\,S_9\,S_{10}$	S_{11}

NFA → DFA –example 2-

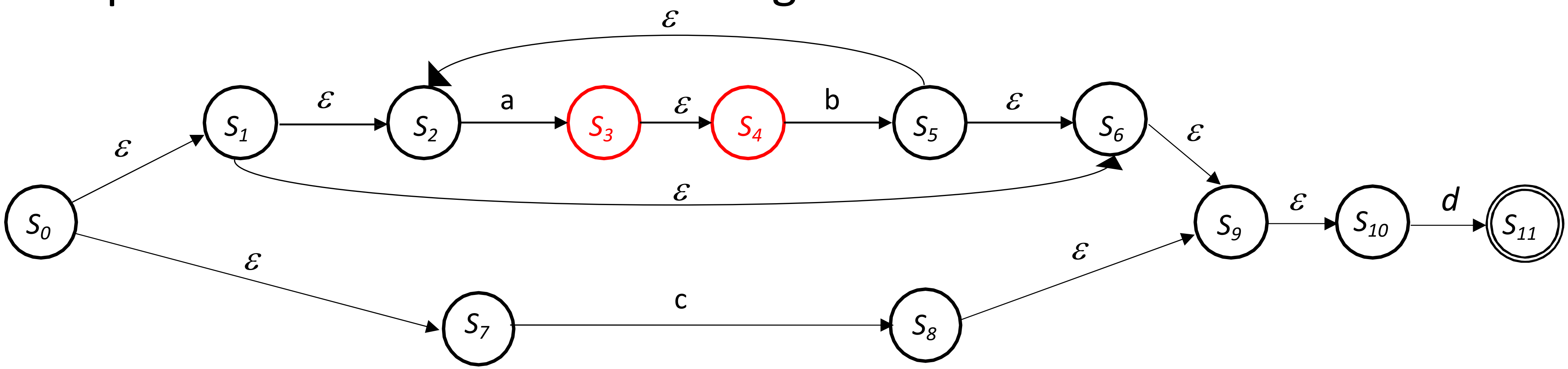
Compute the DFA from the following NFA:



States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0 S_1 S_2 S_3 S_7 S_9 S_{10}$	$S_3 S_4$	None	$S_8 S_9 S_{10}$	S_{11}
d_1	$S_3 S_4$				
d_2	$S_8 S_9 S_{10}$				
d_3	S_{11}				

NFA → DFA –example 2-

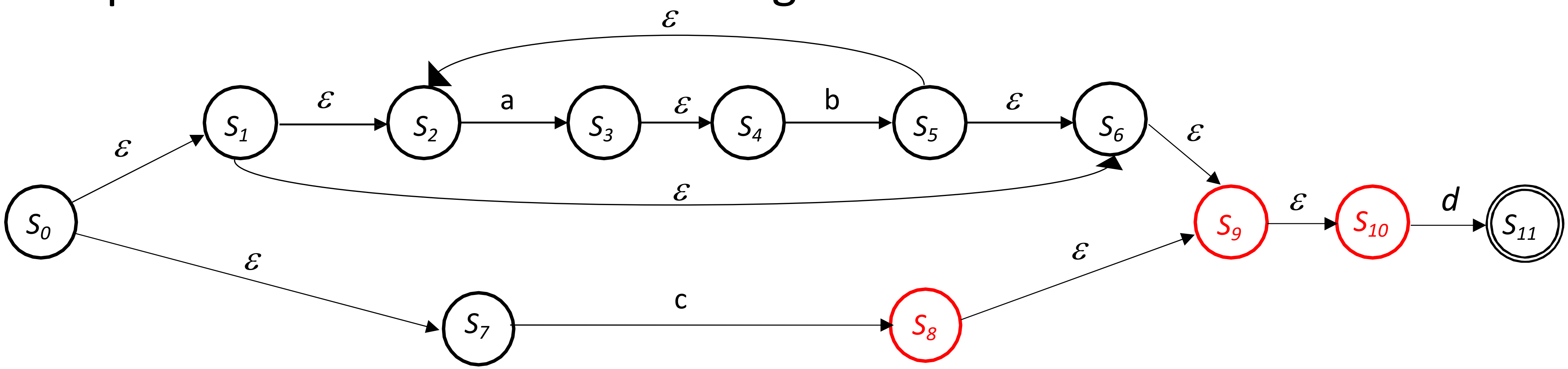
Compute the DFA from the following NFA:



States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0\,S_1\,S_2\,S_3\,S_7\,S_9\,S_{10}$	$S_3\,S_4$	None	$S_8\,S_9\,S_{10}$	S_{11}
d_1	$S_3\,S_4$	None	$S_2\,S_5\,S_6\,S_9\,S_{10}$	None	None
d_2	$S_8\,S_9\,S_{10}$				
d_3	S_{11}				

NFA → DFA –example 2-

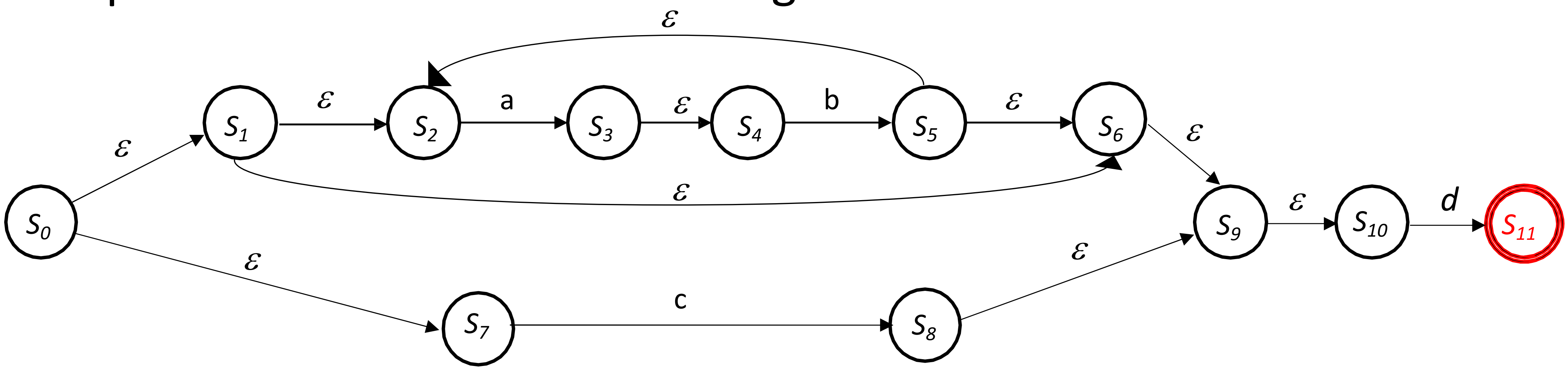
Compute the DFA from the following NFA:



States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0 S_1 S_2 S_3$ $S_7 S_9 S_{10}$	$S_3 S_4$	None	$S_8 S_9 S_{10}$	S_{11}
d_1	$S_3 S_4$	None	$S_2 S_5 S_6$ $S_9 S_{10}$	None	None
d_2	$S_8 S_9 S_{10}$	None	None	None	S_{11}
d_3	S_{11}				

NFA → DFA –example 2-

Compute the DFA from the following NFA:

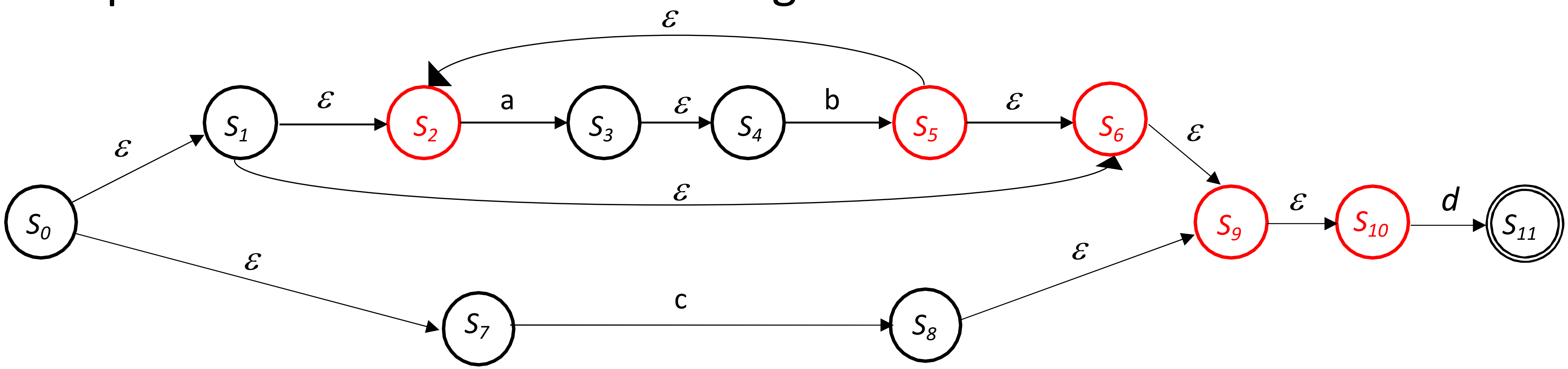


States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0 S_1 S_2 S_3$ $S_7 S_9 S_{10}$	$S_3 S_4$	None	$S_8 S_9 S_{10}$	S_{11}
d_1	$S_3 S_4$	None	$S_2 S_5 S_6$ $S_9 S_{10}$	None	None
d_2	$S_8 S_9 S_{10}$	None	None	None	S_{11}
d_3	S_{11}	None	None	None	None

d_3 is the final state, as it contains S_{11}

NFA → DFA –example 2-

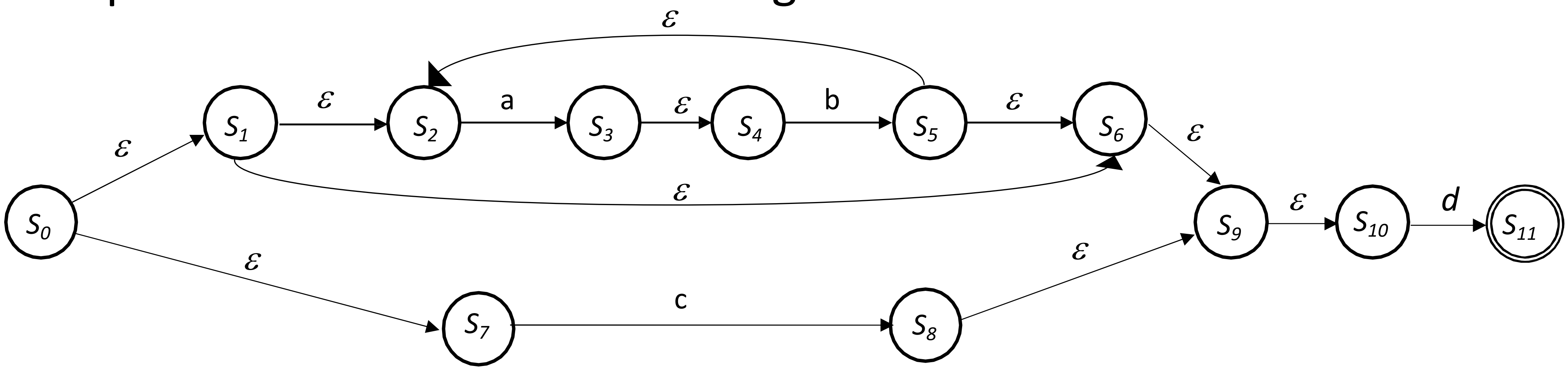
Compute the DFA from the following NFA:



States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0\,S_1\,S_2\,S_3$ $S_7\,S_9\,S_{10}$	$S_3\,S_4$	None	$S_8\,S_9\,S_{10}$	S_{11}
d_1	$S_3\,S_4$	None	$S_2\,S_5\,S_6$ $S_9\,S_{10}$	None	None
d_2	$S_8\,S_9\,S_{10}$	None	None	None	S_{11}
d_3	S_{11}	None	None	None	None
d_4	$S_2\,S_5\,S_6$ $S_9\,S_{10}$	d_1	None	None	d_3

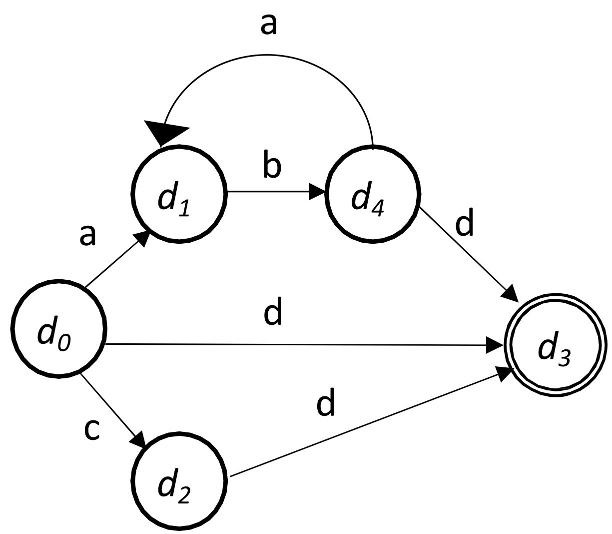
NFA → DFA –example 2-

Compute the DFA from the following NFA:



States		***FollowEpsilon* (*Move*(*s,***)			
DFA		a	b	c	d
d_0		$S_3\,S_4$	None	$S_8\,S_9\,S_{10}$	S_{11}
d_1		None	$S_2\,S_5\,S_6$ $S_9\,S_{10}$	None	None
d_2		None	None	None	S_{11}
d_3		None	None	None	None
d_4		d_1	None	None	d_3

((ab)*| c)d



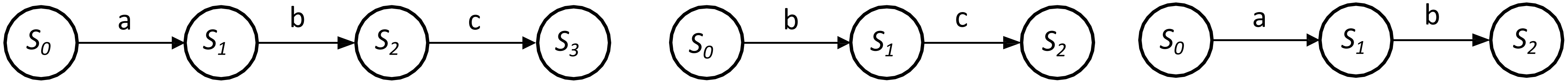
RE → DFA –example-

Compute the DFA from the regular expression abc | bc | ab:

RE → DFA –example-

<u>Compute the DFA from the regular expression abc | bc | ab:</u>

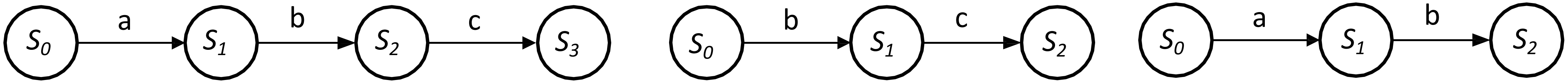
We start by computing the NFA. As there are no parenthesis and no Kleene stars, we perform first the concatenations:



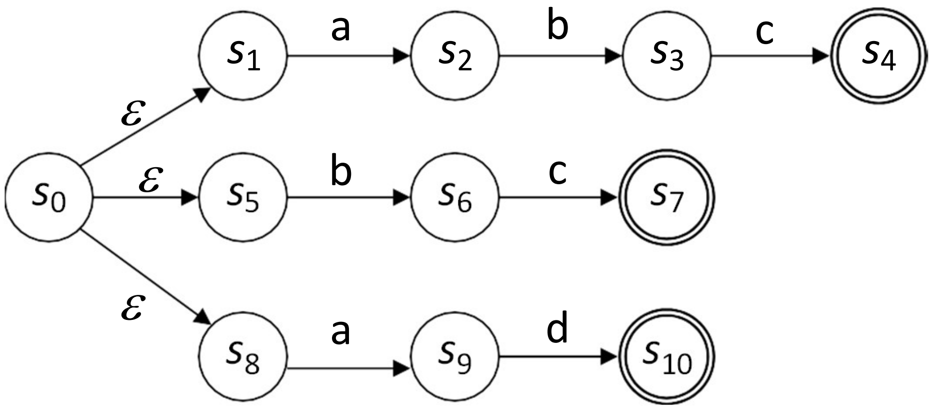
RE → DFA –example-

Compute the DFA from the regular expression abc | bc | ab:

We start by computing the NFA. As there are no parenthesis and no Kleene stars, we perform first the concatenations:



Finally, we perform the alternations

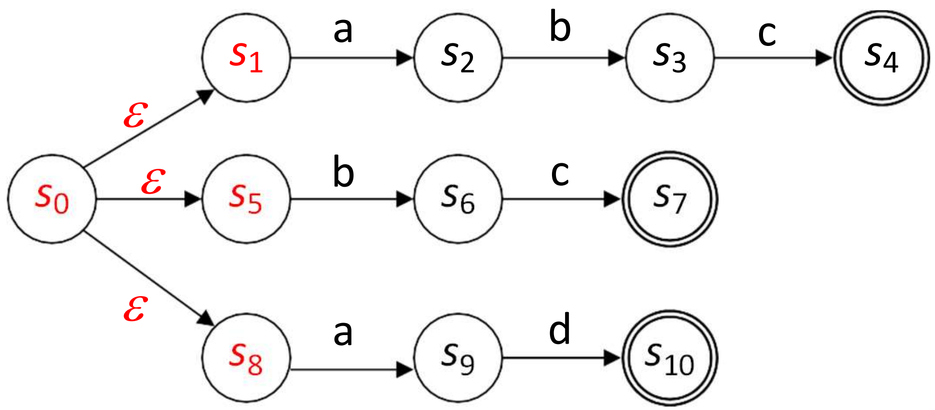


Note: I do not add the ε transitions at the –adding s_4, s_7 , s_{10} as final states- due to there is nothing else to add to the NFA.

RE → DFA –example-

<u>Compute the DFA from the regular expression abc | bc | ab:</u>

Let's compute now the DFA.

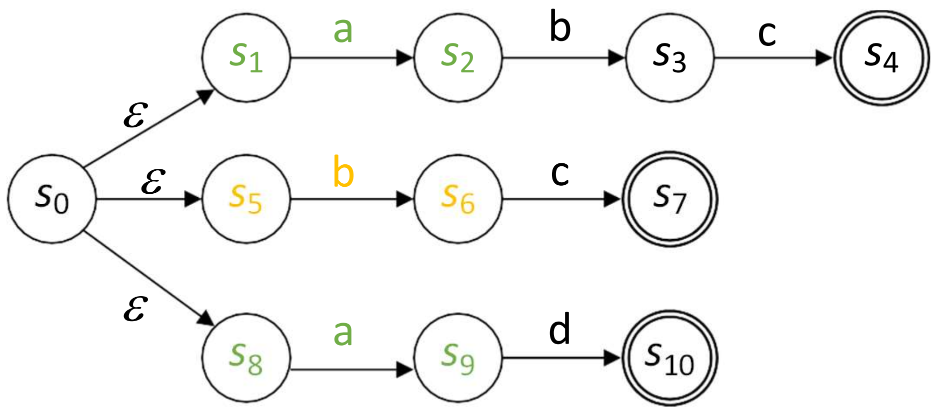


States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	*$S_0 S_1 S_5 S_8$*				

RE → DFA –example-

<u>Compute the DFA from the regular expression abc | bc | ab:</u>

Let's compute now the DFA.

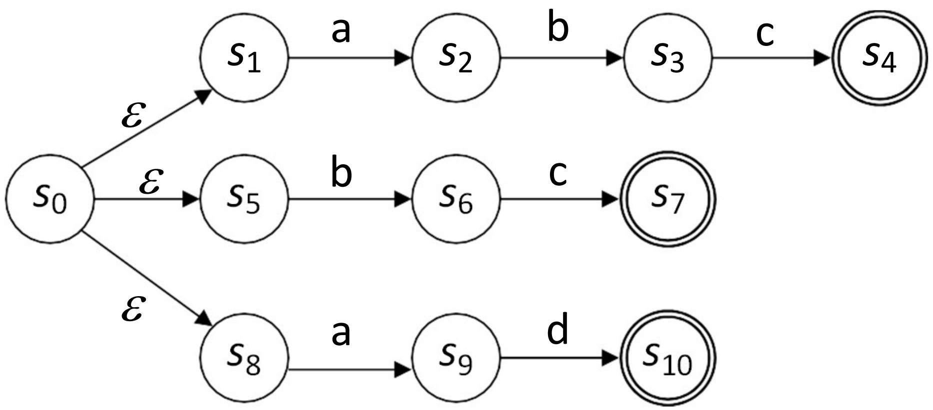


States		***FollowEpsilon* (*Move*(*s,**)**			
DFA	NFA	a	b	c	d
d_0	$S_0\, S_1\, S_5\, S_8$	$S_2 S_9$	S_6	None	None

RE → DFA –example-

Compute the DFA from the regular expression abc | bc | ab:

Let's compute now the DFA.

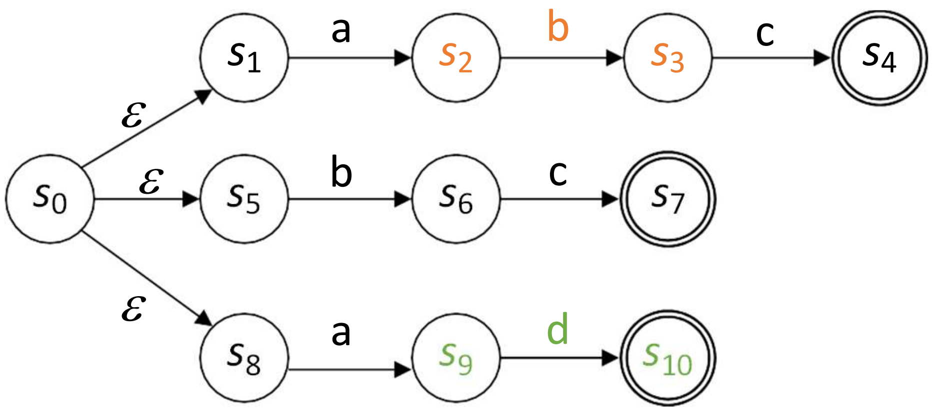


States		*FollowEpsilon (Move(s,*)*			
DFA	NFA	a	b	c	d
d_0	$S_0 S_1 S_5 S_8$	S_2S_9	S_6	None	None
d_1	S_2S_9				
d_2	S_6				

RE → DFA –example-

Compute the DFA from the regular expression abc | bc | ab:

Let's compute now the DFA.

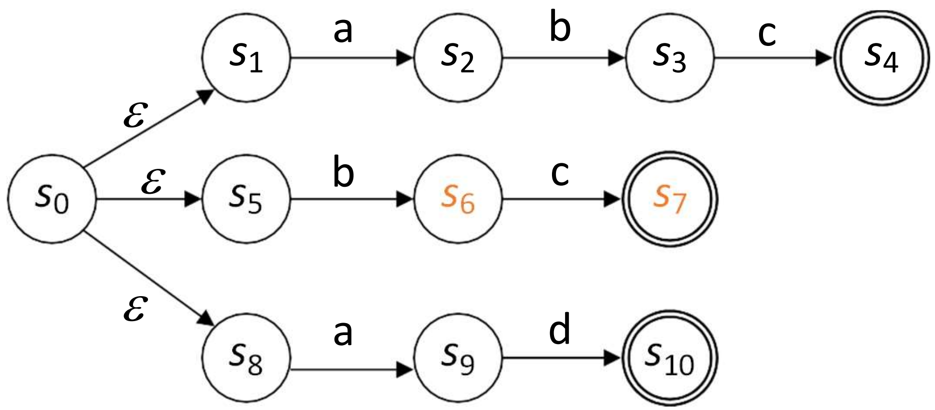


States		***FollowEpsilon* (*Move*(*s,**)**			
DFA	NFA	a	b	c	d
d_0	$S_0 S_1 S_5 S_8$	$S_2 S_9$	S_6	None	None
d_1	$S_2 S_9$	None	S_3	None	S_{10}
d_2	S_6				

RE → DFA –example-

Compute the DFA from the regular expression abc | bc | ab:

Let's compute now the DFA.

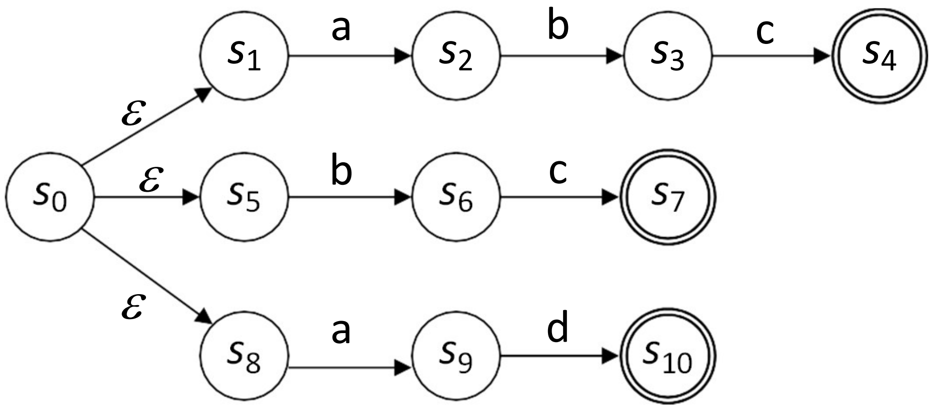


States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0 S_1 S_5 S_8$	S_2S_9	S_6	None	None
d_1	S_2S_9	None	S_3	None	S_{10}
d_2	S_6	None	None	S_7	None

RE → DFA –example-

Compute the DFA from the regular expression abc | bc | ab:

Let's compute now the DFA.

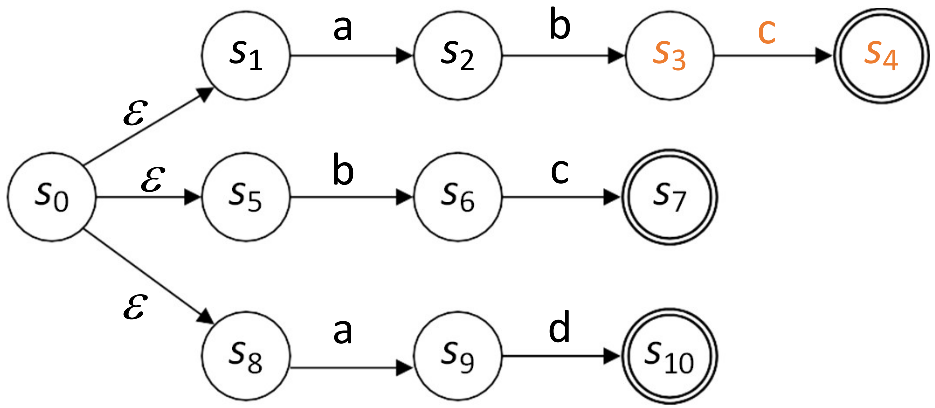


States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0 S_1 S_5 S_8$	$S_2 S_9$	S_6	None	None
d_1	$S_2 S_9$	None	S_3	None	S_{10}
d_2	S_6	None	None	S_7	None
d_3	S_3				
d_4	S_{10}				

RE → DFA –example-

<u>Compute the DFA from the regular expression abc | bc | ab:</u>

Let's compute now the DFA.

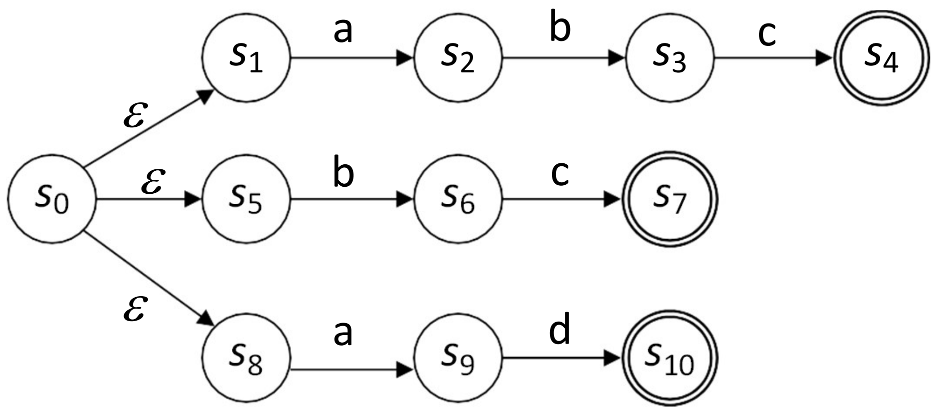


States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0 S_1 S_5 S_8$	$S_2 S_9$	S_6	None	None
d_1	$S_2 S_9$	None	S_3	None	S_{10}
d_2	S_6	None	None	S_7	None
d_3	S_3	None	None	S_4	None
d_4	S_{10}				

RE → DFA –example-

<u>Compute the DFA from the regular expression abc | bc | ab:</u>

Let's compute now the DFA.

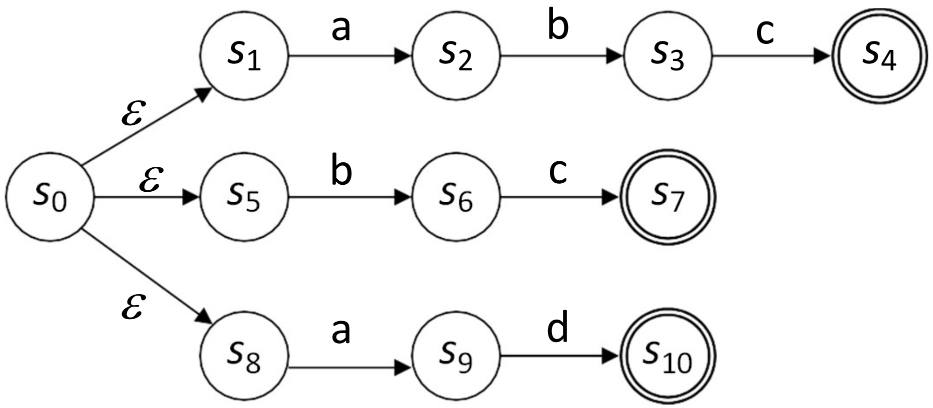


States		***FollowEpsilon (Move(s,*)***			
DFA	NFA	a	b	c	d
d_0	$S_0 S_1 S_5 S_8$	$S_2 S_9$	S_6	None	None
d_1	$S_2 S_9$	None	S_3	None	S_{10}
d_2	S_6	None	None	S_7	None
d_3	S_3	None	None	S_4	None
d_4	S_{10}	None	None	None	None

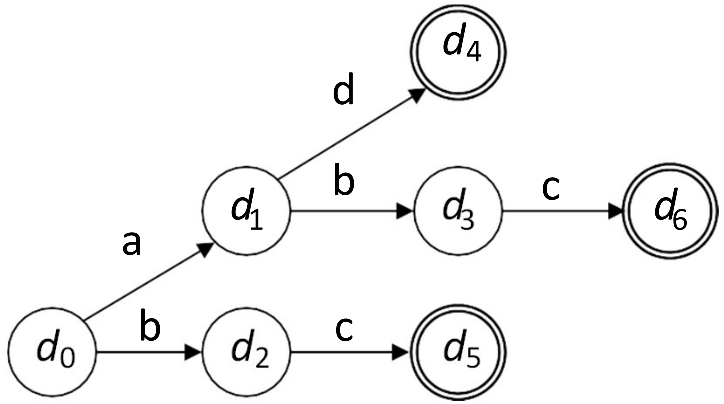
RE → DFA –example-

Compute the DFA from the regular expression abc | bc | ab:

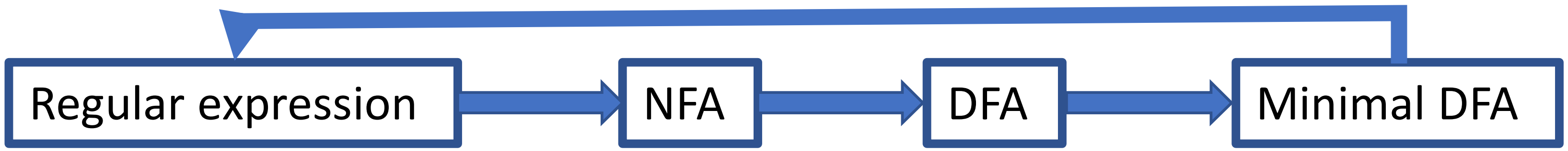
Let’s compute now the DFA.



States		*FollowEpsilon (Move(s,*)*			
DFA		a	b	c	d
d_0		d_1	d_2	None	None
d_1		None	d_3	None	d_4
d_2		None	None	d_5	None
d_3		None	None	d_6	None
d_4		None	None	None	None



Planing



RE → NFA *(Thompson construction)*

Build a **NFA** for each term in the **RE**

Combine them following the patterns marked by the operators.

NFA → DFA *(Subset construction)*

Build a **DFA** that simulates the **NFA**

DFA → Minimal **DFA**

Brzozowski algorithm's

DFA → RE

Join all the paths from s_0 to a final state

DFA → *minimal* DFA: Brzozowski

Intuition:

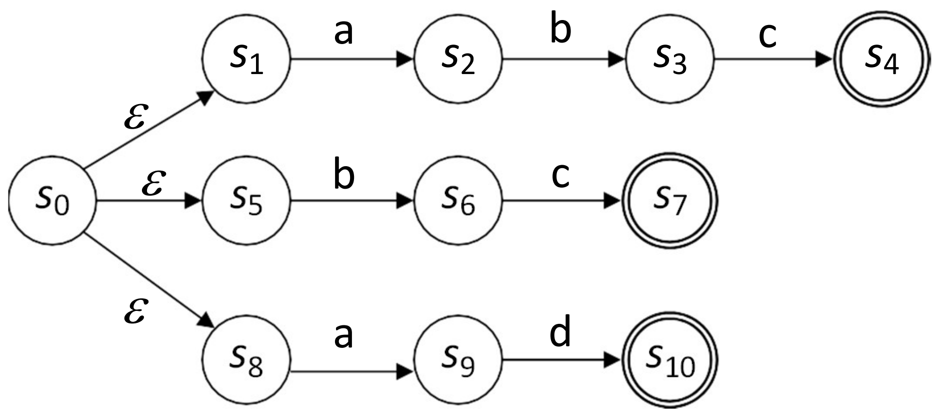
- The subset construction method joins the prefixes that appear in the **NFA.**

DFA → *minimal* DFA: Brzozowski

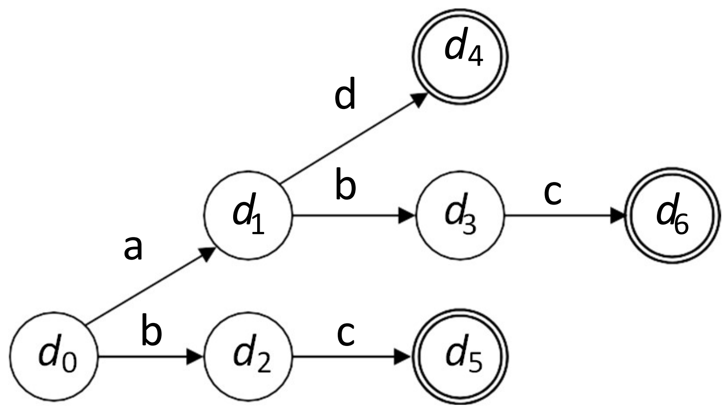
Intuition:

- The subset construction method joins the prefixes that appear in the **NFA.**

As an example, for the regular expression abc | bc |bd:



NFA



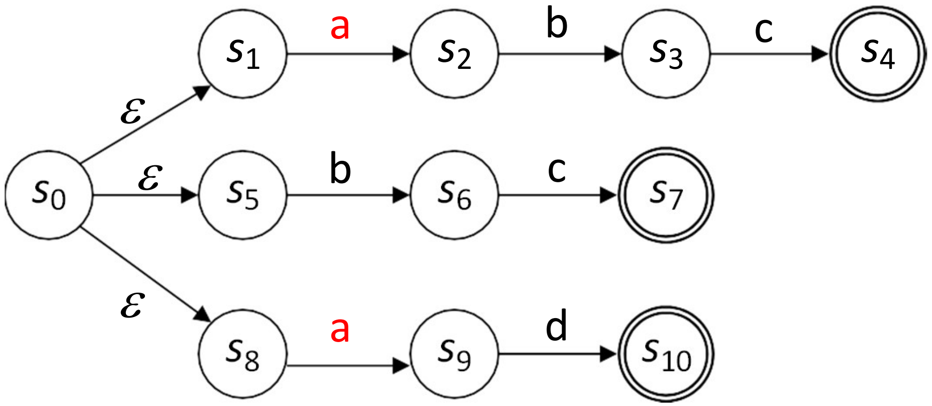
DFA

DFA → *minimal* DFA: Brzozowski

Intuition:

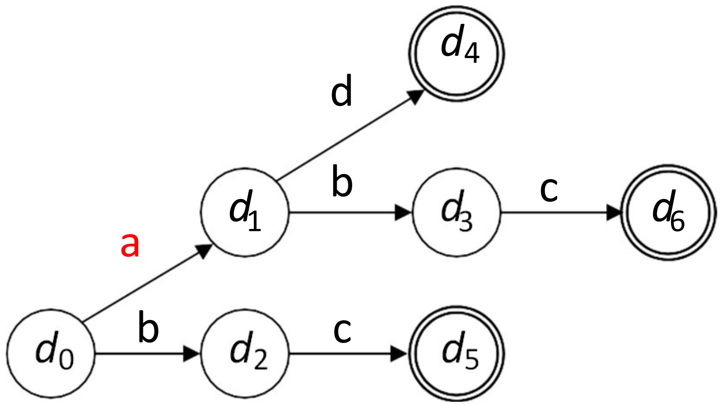
- The subset construction method joins the prefixes that appear in the **NFA.**

As an example, for the regular expression abc | bc |bd:



NFA

In the NFA, Thompson's construction leave ε transitions between simple characters.



DFA

The subset construction method deletes ε transitions and join the patch for aa.
However, it leaves the duplicates tailes intact (bc in this case).

DFA → *minimal* DFA: Brzozowski

<u>Idea</u>:

To use the Subset Construction twice.

DFA → *minimal* DFA: Brzozowski

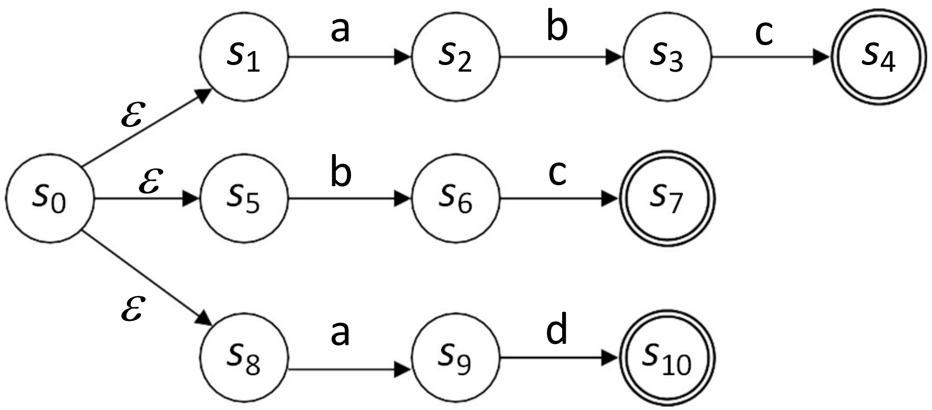
Idea:

To use the Subset Construction twice.

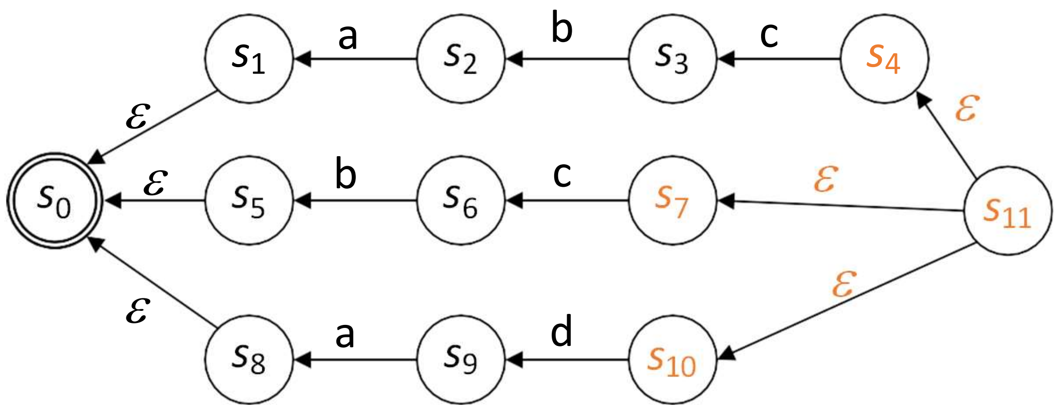
Given an NFA N we define:

– Reverse(N) is the NFA built by:

1) Puting the initial state as final.
2) Adding a new initial state with a ε *transition* to each previous final state.
3) Rotating the rest of edges.



N



Reverse(N)

DFA → *minimal* DFA: Brzozowski

Idea:

To use the Subset Construction twice.

Given an NFA N we define:

– Reverse(N) is the NFA built by:

1) Puting the initial state as final.
2) Adding a new initial state with a ε *transition* to each previous final state.
3) Rotating the rest of edges.

– Subset(N) is the DFA resulting from applying the subset construction to N

- Reachable(N) is the result to delete in N all the states that can not be reached from the initial state.

DFA → *minimal* DFA: Brzozowski

<u>Idea</u>:

To use the Subset Construction twice.

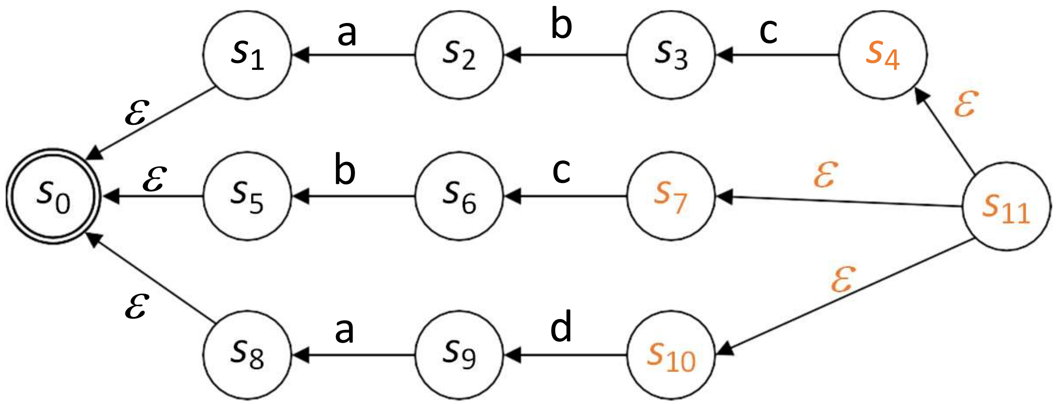
<u>The minimal DFA for the NFA N is obtained as follows:</u>

Reachable(Subset(Reverse(Reachable(Subset(Reverse(N))))))

DFA → *minimal* DFA –example-

Step 1

Apply subset construction to *Reverse(NFA)* in order to join the sufixes of the original NFA

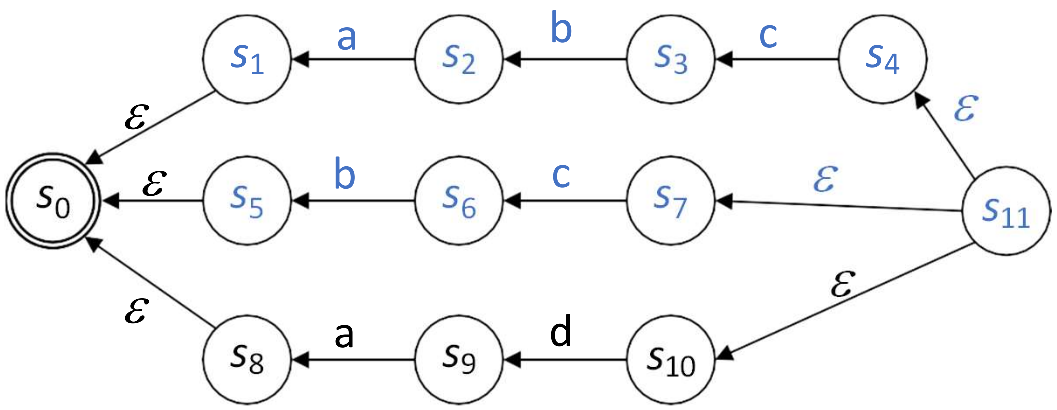


Reverse(N)

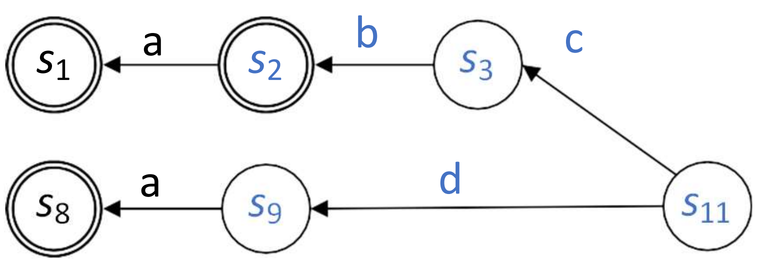
DFA → *minimal* DFA –example--

Step 1

Apply subset construction to *Reverse(NFA)* in order to join the sufixes of the original NFA



Reverse(N)

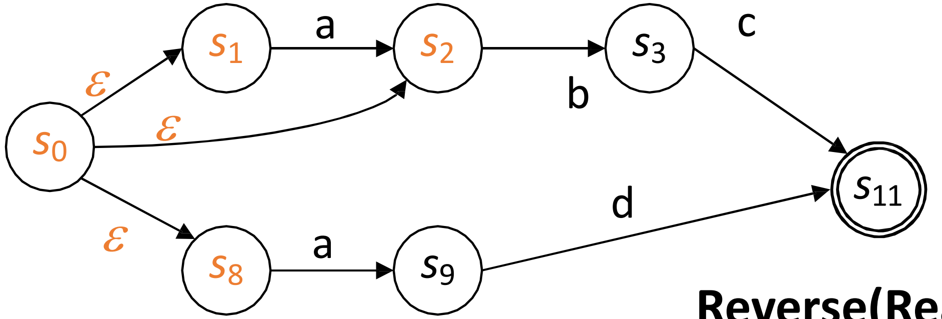


Reachable(Subset(Reverse(N)))

DFA → *minimal* DFA –example--

Step 2

We apply again Reverse(·), and we use the subset construction to join the prefixes of the original NFA

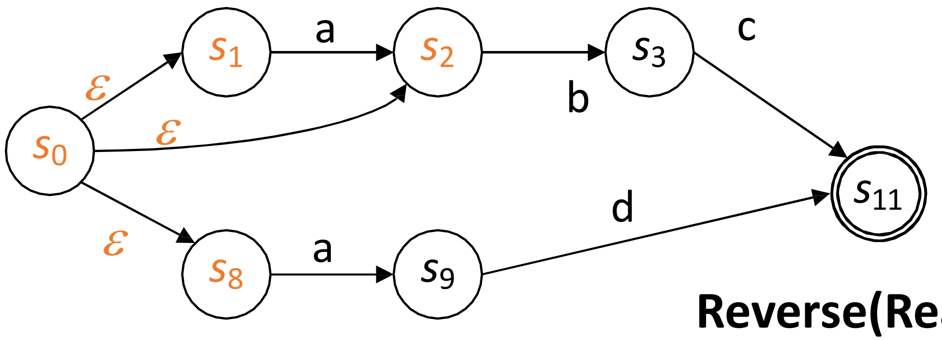


Reverse(Reachable(Subset(Reverse(N)))

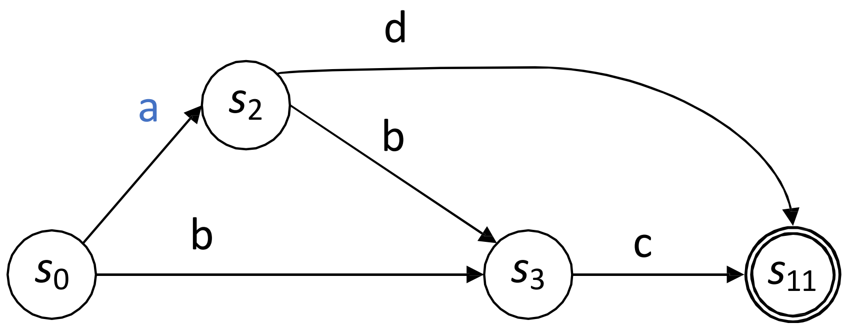
DFA → *minimal* DFA –example--

Step 2

We apply again Reverse(·), and we use the subset construction to join the prefixes of the original NFA

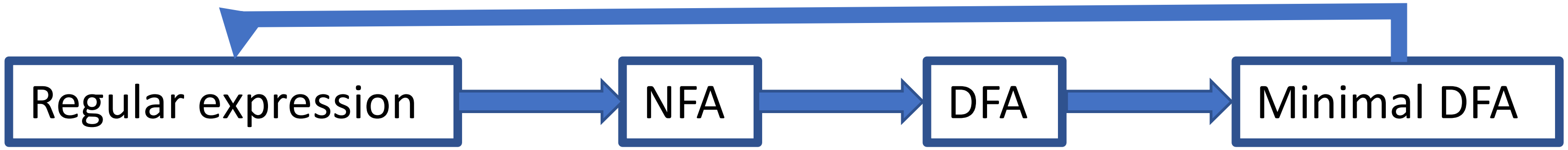


Reverse(Reachable(Subset(Reverse(N)))



Minimal DFA: Reachable(Subset (Reverse(Reachable(Subset(Reverse(N)))))

Planing



RE → NFA *(Thompson construction)*

Build a **NFA** for each term in the **RE**

Combine them following the patterns marked by the operators.

NFA → DFA *(Subset construction)*

Build a **DFA** that simulates the **NFA**

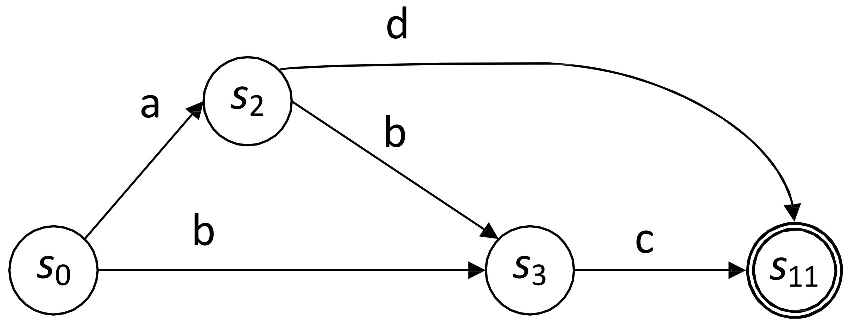
DFA → Minimal **DFA**

Brzozowski algorithm's

DFA → RE

Join all the paths from s_0 to a final state

DFA → *RE*



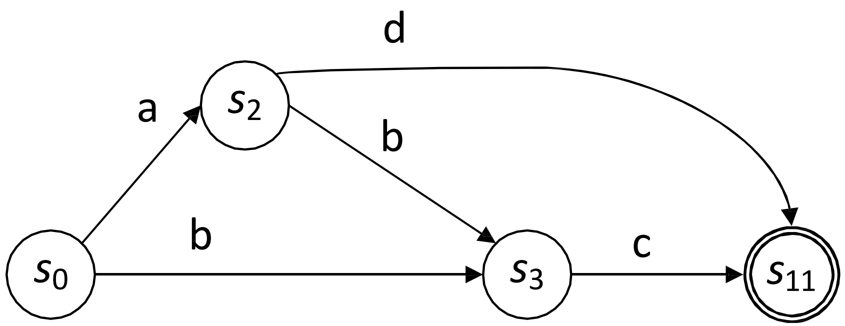
Possible paths from initial state to final state:

ad

abc

bc

DFA → *RE*



Possible paths from initial state to final state:

ad

abc

bc

Then the regular expression is:

ad | abc | bc