Theory of Computing :

4. Regular Expressions

Professor Imed Bouchrika
National Higher School of Artificial Intelligence
imed.bouchrika@ensia.edu.dz

Outline :

- Revision :
    - Nondeterministic
    - Converting NFA to DFA
    - Minimizing DFA
- Regular Expressions
- Building FA from Regular Expressions
- Regular Expressions for text processing

Nondeterministic Finite Automata

- **Nondeterministic Finite Automata (NFA)**
  - For each state there can be zero, one, two, or more transitions corresponding to a particular symbol.
  - Why :
    - *Because it is easier , compared to the conditions imposed by DFA*
      - *To construct*
      - *To understand*
    - *To simulate scenarios in real life (but computers are deterministic machines)*

Nondeterministic Finite Automata

- **DFA vs NFA :**

	DFA	**NFA**
Transition with the same label from a state	Strictly one	Multiple
Transition with the empty string	Does not exist	It exists
Trap State for missing transitions	Obligatory	Optional

Converting NFA to DFA

- **Algorithm to convert NFA to DFA :**
  - It is called conversion by subset construction :
  - States are represented as sets from the power set : 2^Q
    1. Determine the initial Start State
       - *It is the set containing the original start state union all other states reached from the original state by ε (directly or indirectly)*
    2. Determine the Accept States
       - *Any State set containing at least an original accept state*
    3. For each possible state created from 2^Q, find the possible transitions
       - *If there is missing transition, create a dead state*
    4. Draw the state diagram
    5. Remove any state without **incoming** transitions
  - Another Strategy : **Use the Transition Tables to facilitate the conversion**

Minimization of Finite Automata

- Simpler version of the algorithm in plain English:
    - Group "equivalent" states into a single "region" or set:
    - Equivalent =
        - States equivalent if they lead to the same "region"
        - Region = set of equivalent states.
    - If elements in a grouped set/region are not equivalent:
        - We create a separate "region" for them.
    - We keep splitting regions until we are no able to split = all elements in each region are equivalent.

Regular Expressions

- **Finite Automata**
  - Abstract machines to represent a language
  - It is mostly a graphical representation
  - Is there a different representation that :
    - People can understand ?
    - Machine can compile and interpret ?

Regular Expressions

- **Definition**
  - For this course :
    - A textual representation of regular languages using the three operators : union, concatenation and the star
    - Originated from the work of the mathematician : Stephen Cole Kleene in 1951
  - For text processing :
    - A sequence of characters or operators to represent a particular pattern of strings

Regular Expressions

- **Definition**
    - Regular expressions are usually compact representations and human readable.
    - Regular expressions are abbreviated as : Regex or RE
    - There are variations in the notations depending on the textbook, programming language (Example for the union : **U** vs **+** vs **|**)

Regular Expressions

- **Operation : Union**
  - Notation :
    - |　　U　　+
  - Examples for words:
    - (a U b)
      - a **or b**
  - For languages :
    - {a,b,c} U {1,2,3} = {a,b,c,1,2,3}

Regular Expressions

- **Operation : Concatenation**
  - Notation :
    - ab
    - $a \circ b$
  - Explanation :
    - Attach the first part to the second part.

Regular Expressions

- **Operation : Concatenation**
  - Given languages L_1 and L_2 , we define their concatenation to be the language $L_1 \circ L_2 = \{xy \mid x \in L_1 , y \in L_2 \}$
  - Examples :
    - $\{a,b,c\} \circ \{1,2,3\} = \{a1,a2,a3,b1,b2,b3,c1,c2,c3\}$
    - $L_1 = \{hello\}$ and $L_2 = \{world\}$ then $L_1 \circ L_2 = \{helloworld\}$

Regular Expressions

- **Operation : Star**
  - Notation
    - a^*
  - Example for words :
    - $a^* = \{\boldsymbol{\varepsilon}, a, aa, aaa, aaaa, \ldots\}$
  - For languages :
    - $\{a,b\}^* = \{\boldsymbol{\varepsilon}$, a, b, ab,ba,aab, … all words composed by a and b }

Regular Expressions

- **Operation : Star**
  - More examples:
    - L={ ma , xy , bc }
    - L^* = { ε , ma, xy , maxy, xyma, mabc, bcma, maxymc, ….. }

Regular Expressions

- **Formal Definition**
  - Regular expressions can be in of the following forms:
    - a for some a in the alphabet Σ
    - ε
    - ∅
  - In addition to (provided that R_1 and R_2 are regular expressions:)
    - $R_1 \cup R_2$
    - $R_1 \circ R_2$
    - R_1^*

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $01 \cup 10$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $01 \cup 10$
    - **L= {01,10}**

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $(0 \cup \varepsilon)(1 \cup \varepsilon)$

Regular Expressions

- **Examples :**
  - Σ = { 0 , 1 }
  - What's the language for the following regular expression:
    - (0 ∪ ε)(1 ∪ ε)
    - **L={** ε , 0, 1 , 01 }

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $\Sigma^* 001\Sigma^*$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $\Sigma^* 001\Sigma^*$
    - **L = { w | w contains 001 as a substring}**

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $(0 \cup 1)^* \, 001 \, (0 \cup 1)^*$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $(0 \cup 1)^* 001 (0 \cup 1)^*$
    - It is the same as : $\Sigma^* 001\Sigma^*$
    - **L = { w | w contains 001 as a substring}**

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $0^* \, 1 \, 0^*$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - 1*

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - 1* = {ε, 1, 11,111,111……}

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $(01)^*$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $(01)^* = \{\varepsilon, 01, 0101, 010101, \}$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - (0*| 1*)

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $(0^*|\ 1^*) = \{\varepsilon,0,00,000,0000,....\} \cup \{\varepsilon,1,11,111,1111,...\}$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - (11 | 01)*

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - (11 | 01)* = {ε,11, 01, 1101,0111, 111111,010101,..... Any word composed from 11 and 01 in any way}

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $0^* \, 1 \, 0^*$
    - **L = {w| w contains a single 1}.**

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $(0 \cup 1)^* 1 (00)^* 00$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $(0 \cup 1)^* 1 (00)^* 00$
    - **L = {w| w ends with an even number of zeros}.**

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - What's the language for the following regular expression:
    - $(0 \cup 1)^* 1 (00)^* 00$
    - **L = {w| w ends with an even number of zeros}.**
      - **00 ?**

Regular Expressions

- **Examples :**
  - Σ = { 0 , 1 }
  - What's the language for the following regular expression:
    - (0 U 1)* 1 (00)*00
    - **L = {w| w ends with an even number of zeros}.**
      - **00 ?**
      - (((0|1)*1) | e) (00)* 00

        (e = epsilon)

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language $L = \{ w \in \Sigma^* \mid w$ contains 00 as a substring $\}$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language $L = \{ w \in \Sigma^* \mid w$ contains 00 as a substring $\}$

$$(0 \mid 1)^*00(0 \mid 1)^*$$

11011100101
0000
111110111110011111

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language $L = \{ w \in \Sigma^* \mid w$ contains most a single $0 \}$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language L = { w $\in \Sigma^*$ | w contains most a single 0 }

$$1^*(0 \mid \varepsilon)1^*$$

11110111
111111
0111
0

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language L = $\{ w \in \Sigma^* \mid w$ has only four symbols, $|w|=4 \}$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language $L = \{ w \in \Sigma^* \mid w \text{ has only four symbols, } |w|=4 \}$

(0|1)(0|1)(0|1)(0|1)

0000
1010
1111
1000

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language $L = \{ w \in \Sigma^* \mid w$ has only four symbols, $|w|=4 \}$

$(0|1)(0|1)(0|1)(0|1)$

0000
1010
1111
1000

$(0|1)^4$

0000
1010
1111
1000

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language L = { w $\in \Sigma^*$, w starts and ends with the same symbol}

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language L = { w $\in \Sigma^*$, w starts and ends with the same symbol}
  - **$(0 \Sigma^* 0) \cup (1 \Sigma^* 1)$**

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language L = { w ∈Σ^* , w starts and ends with the same symbol}
  - $(0 \Sigma^* 0) \cup (1 \Sigma^* 1) \cup 0 \cup 1$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language $L = \{ w \in \Sigma^* ,$ w starts and ends with the same symbol and has a length of at least 2$\}$

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language $L = \{ w \in \Sigma^*$, w starts and ends with the same symbol and has a length of at least 2}
  - **$(0 \Sigma^* 0) \cup (1 \Sigma^* 1)$**

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language L = { w $\in \Sigma^*$, w is a string of even length}

Regular Expressions

- **Examples :**
  - $\Sigma = \{ 0 , 1 \}$
  - Language $L = \{ w \in \Sigma^* , w$ is a string of even length$\}$
  - **$(\Sigma\Sigma)*$**
  - **$((1 \cup 0)(1 \cup 0))^*$**

Regular Expressions

- **Regular Languages :**

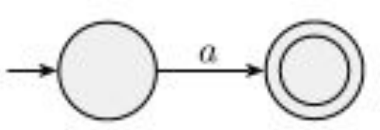
Any language which can be represented by a regular expression, it is a considered a regular language

Regular Expressions

- **Converting to NFA :**
  - R = a for some a $\in \Sigma$. Then L(R) = {a}

Regular Expressions

- **Converting to NFA :**
  - R = a for some a ∈ Σ. Then L(R) = {a}

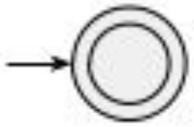


Regular Expressions

- **Converting to NFA :**
  - R = ε. Then L(R) = {ε}

Regular Expressions

- **Converting to NFA :**
  - R = ε. Then L(R) = {ε}

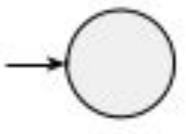


Regular Expressions

- **Converting to NFA :**
  - R = ∅. Then L(R) = ∅

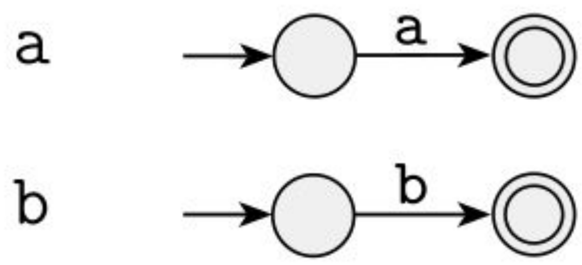
Regular Expressions

- **Converting to NFA :**
  - R = ø. Then L(R) = ø



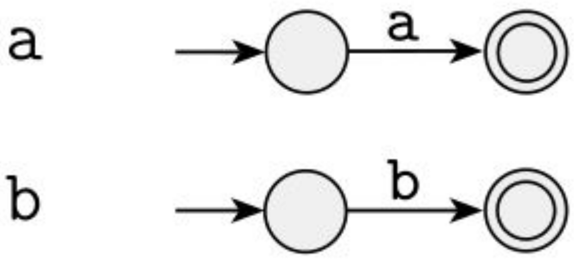
Regular Expressions

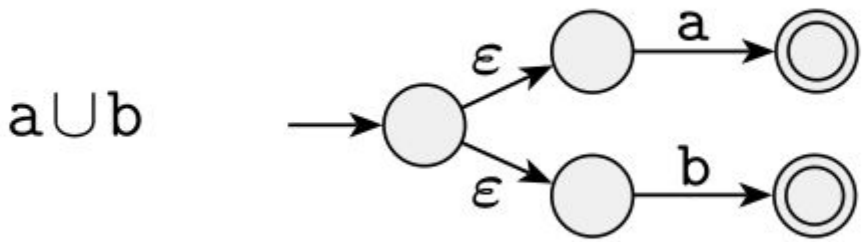
- **Converting to NFA :**
    - Given two simple regular expressions, already represented by their NFAs
    - What's their **union** represented by an NFA **?**



Regular Expressions

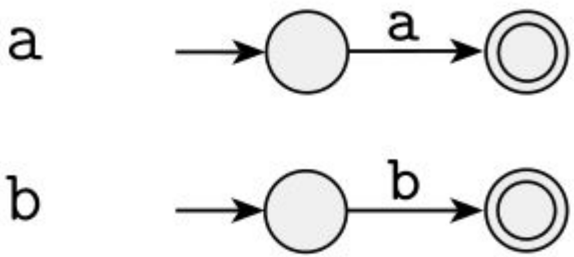
- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - What's their **union** represented by an NFA **?**





Regular Expressions

- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - How to represent their **concatenation** by an NFA **?**

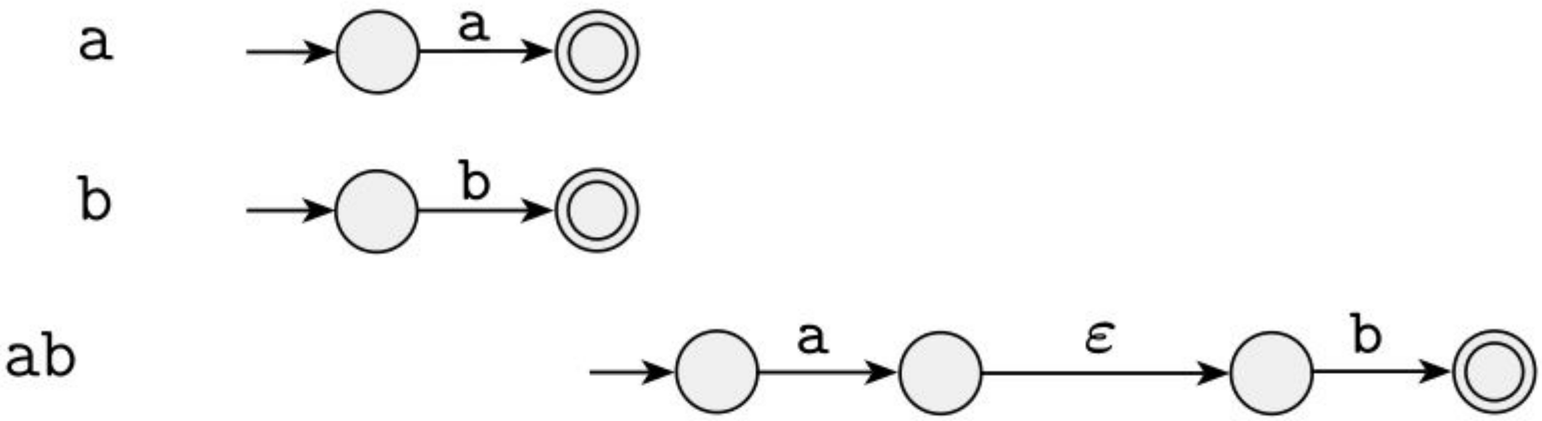


Regular Expressions

- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - How to represent their **concatenation** by an NFA **?**
    - **A B**
      - **Link Accepting States of A to Start state of B with epsilon transition**
      - **Convert all Accepting states of A to non-accepting**

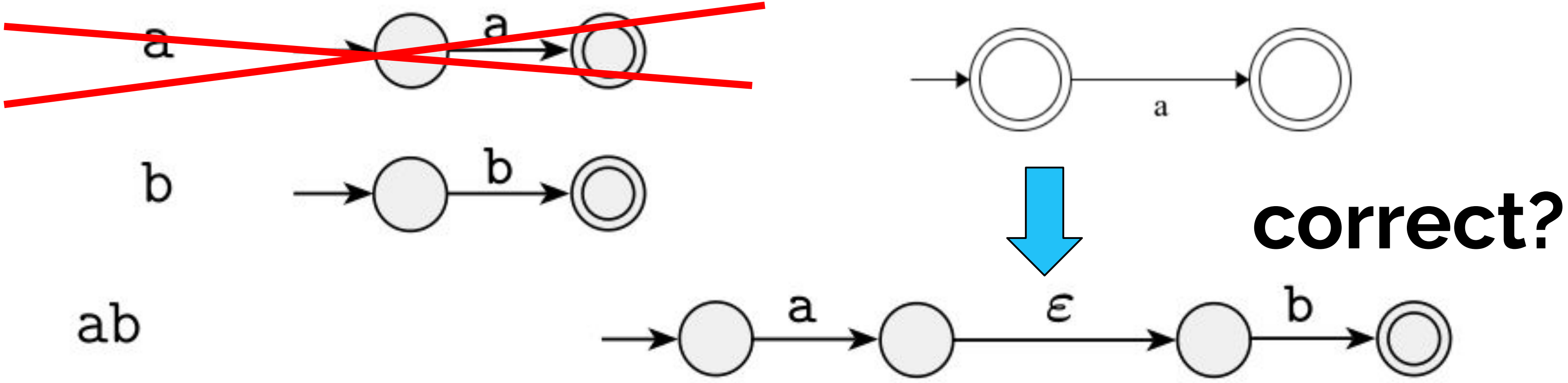
Regular Expressions

- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - How to represent their **concatenation** by an NFA **?**



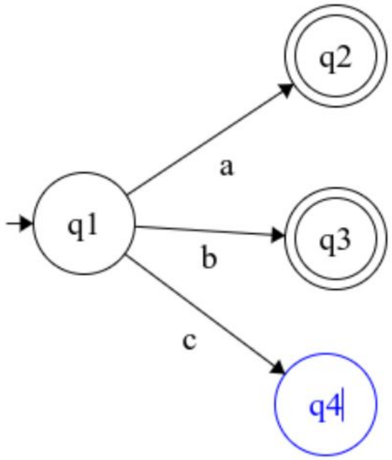
Regular Expressions

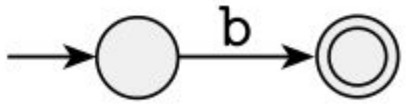
- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - How to represent their **concatenation** by an NFA **?**



Regular Expressions

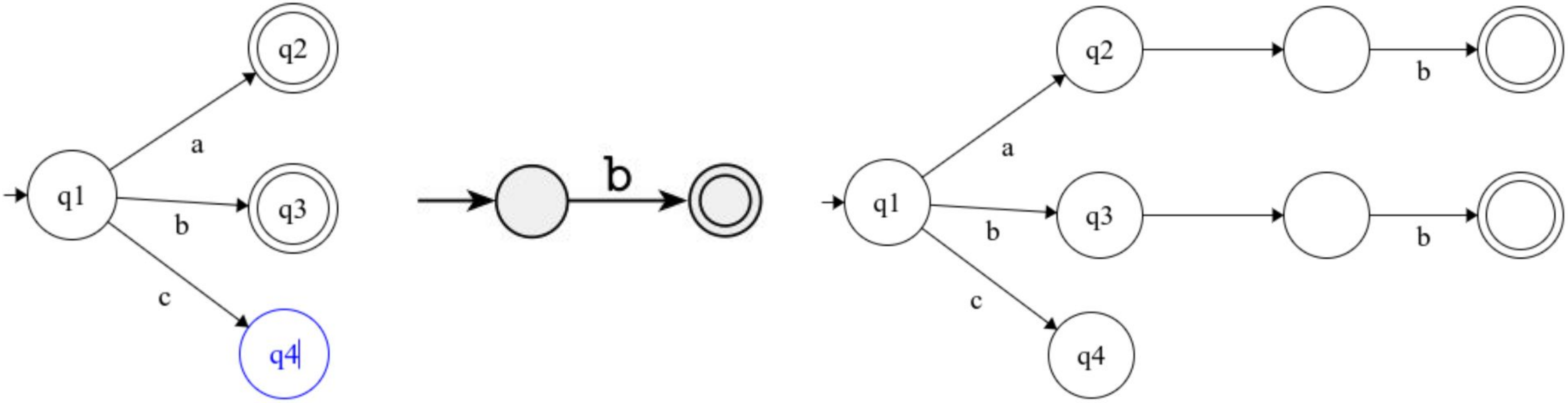
- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - How to represent their **concatenation** by an NFA **?**





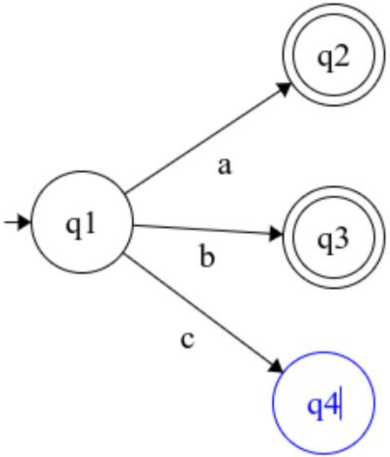
Regular Expressions

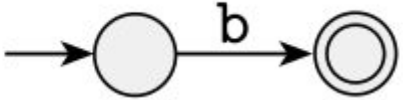
- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - How to represent their **concatenation** by an NFA **?**

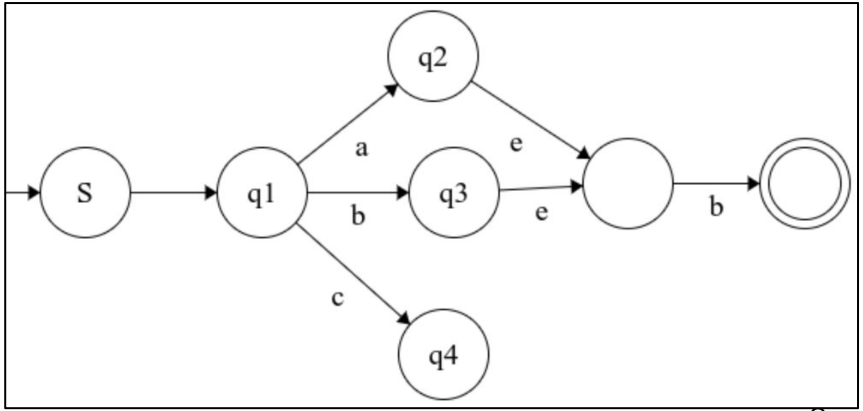


Regular Expressions

- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - How to represent their **concatenation** by an NFA **?**

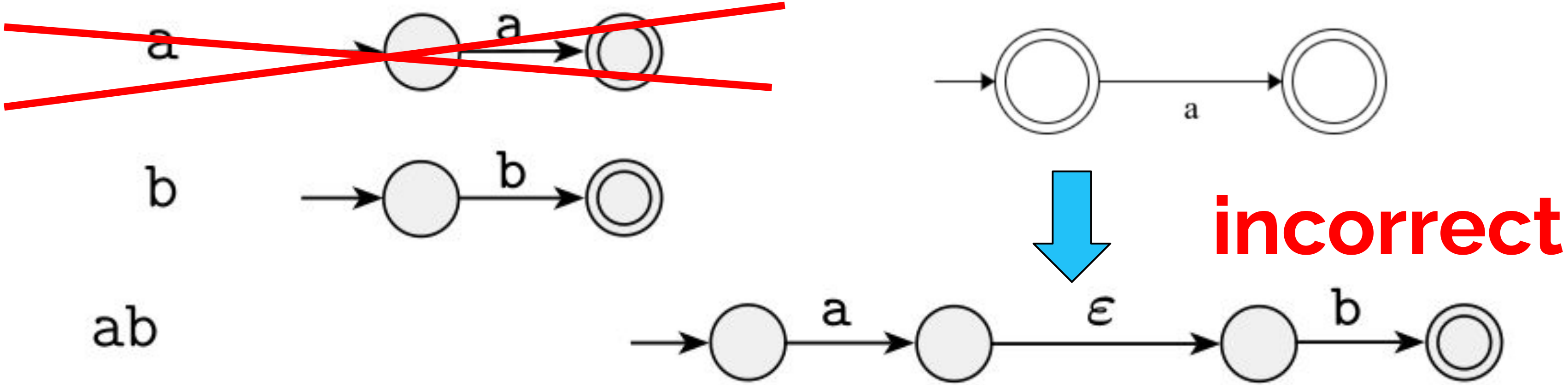






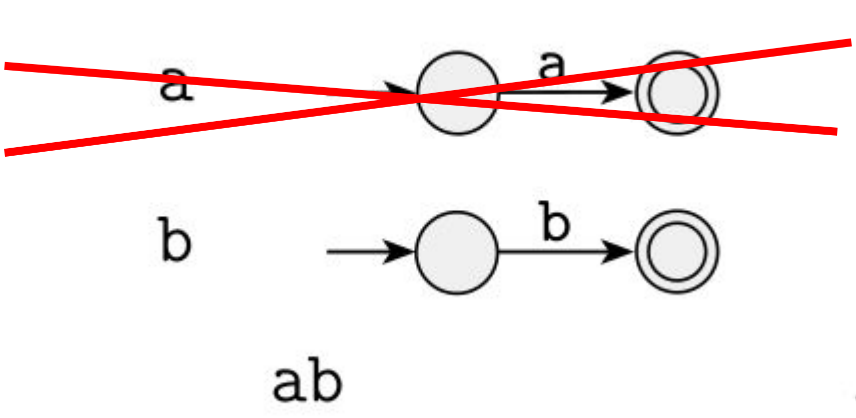
Regular Expressions

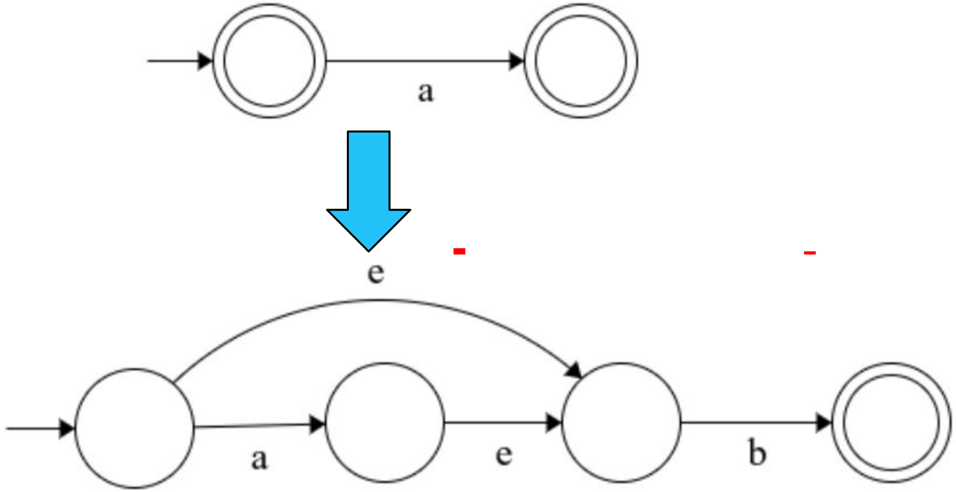
- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - How to represent their **concatenation** by an NFA **?**



Regular Expressions

- **Converting to NFA :**
  - Given two simple regular expressions, already represented by their NFAs
  - How to represent their **concatenation** by an NFA ?

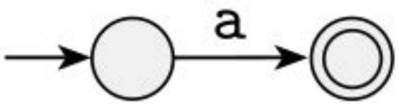




Regular Expressions

- **Converting to NFA :**
  - Given one simple regular expression, already represented by their NFA
  - How to represent the **star** by an NFA **?**
    - **Example : a* = { ε , a, aa,aaa }**

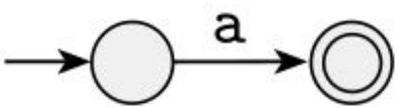
a



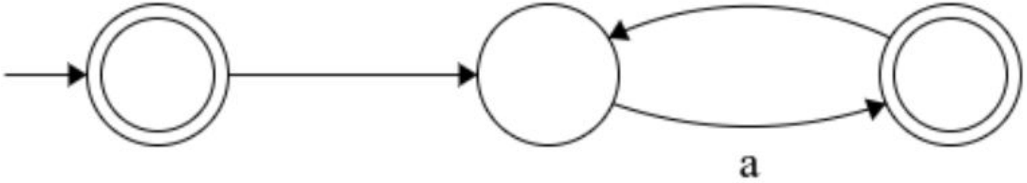
Regular Expressions

- **Converting to NFA :**
  - Given one simple regular expression, already represented by their NFA
  - How to represent the **star** by an NFA **?**
    - **Example : a^* = { ε , a, aa,aaa }**

a



a^*

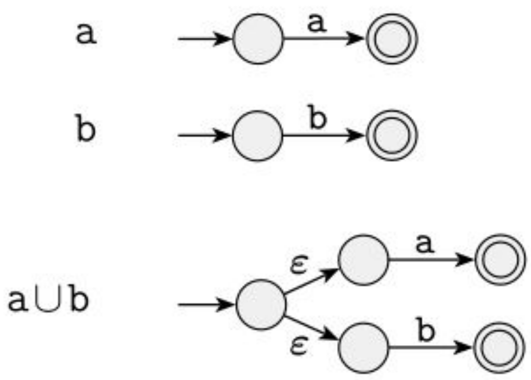


Regular Expressions

- **Converting to NFA :**
  - Given one simple regular expression, already represented by their NFA
  - How to represent the **star** by an NFA **?**
    - **A***
      - **Make a new start state as accepting**
        - **Link it with the previous start state with epsilon transition**
      - **Link all accepting states to the original start state with epsilon transition**

Regular Expressions

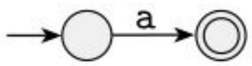
- **Converting to NFA :**
  - Given one simple regular expression, already represented by their NFA
  - How to represent the **star** by an NFA **?**
    - **(b U a)* ?**



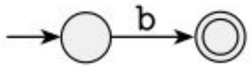
Regular Expressions

- **Converting to NFA :**
  - Given one simple regular expression, already represented by their NFA
  - How to represent the **star** by an NFA **?**
    - **(b U a)* ?**

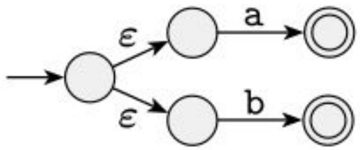
a



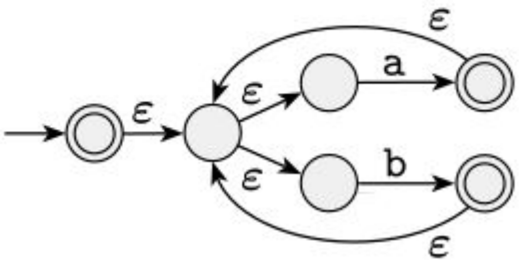
b



a∪b



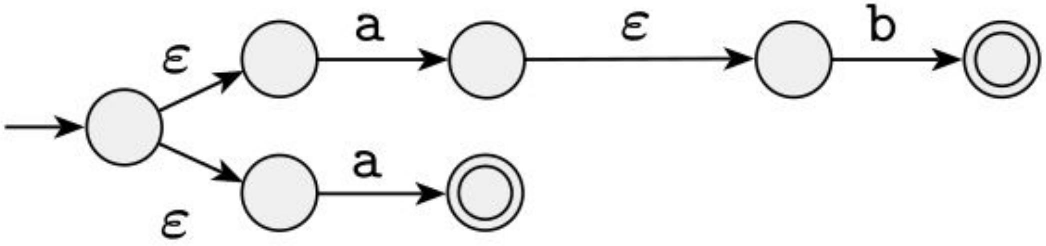
(a∪b)*



Regular Expressions

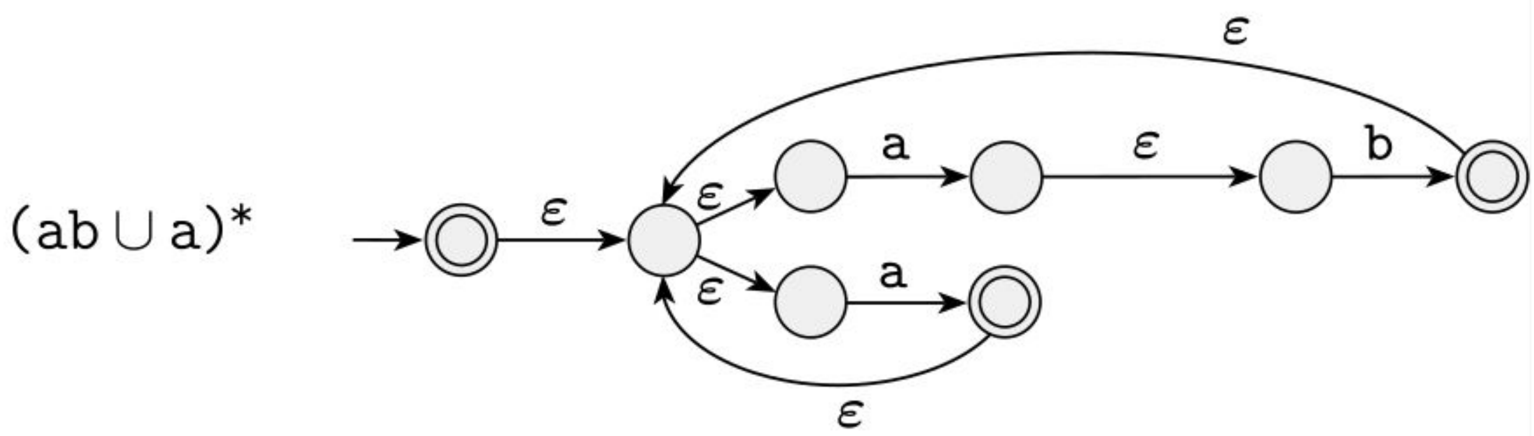
- **Converting to NFA :**
  - Given one simple regular expression, already represented by their NFA
  - How to represent the **star** by an NFA **?**
    - **(ab U a)* ?**

ab ∪ a



Regular Expressions

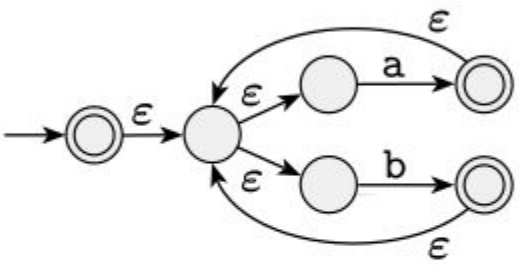
- **Converting to NFA :**
  - Given one simple regular expression, already represented by their NFA
  - How to represent the **star** by an NFA **?**
    - **(ab U a)* ?**



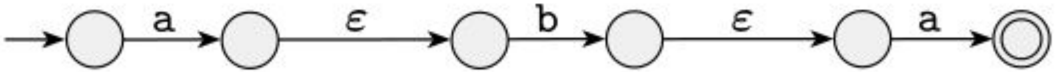
Regular Expressions

- **Converting to NFA :**
  - Given one simple regular expression, already represented by their NFA
  - How to represent the following expression:
    - **(a U b)* aba ?**

(a∪b)*



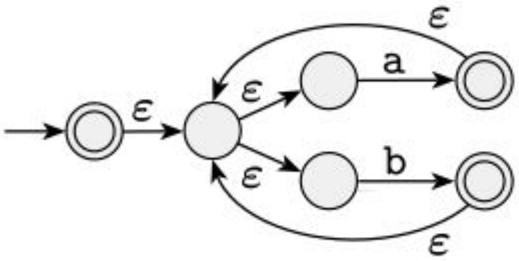
aba



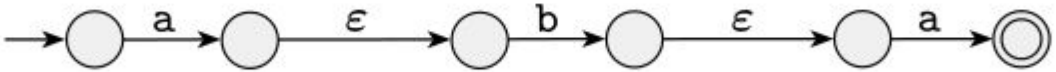
Regular Expressions

- **Converting to NFA :**
  - Given one simple regular express
  - How to represent the following ex
    - **(a U b)* aba ?**

(a∪b)*



aba

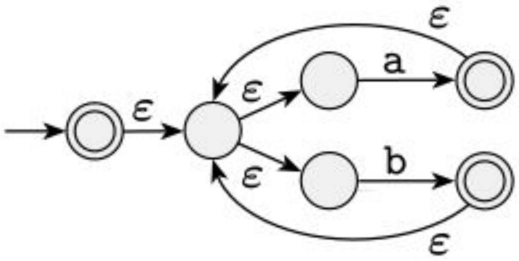


The state diagram for **aba** seems to have redundant epsilon transitions because :

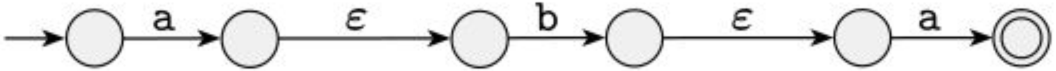
We are following the simple and **safe** rules for concatenating NFAs of the language {a} and {b} and {a}.
(Imagine you are given complex NFAs to concatenate, you must follow the rules)

Of course, you can later optimize and remove redundant transitions provided you are certain.

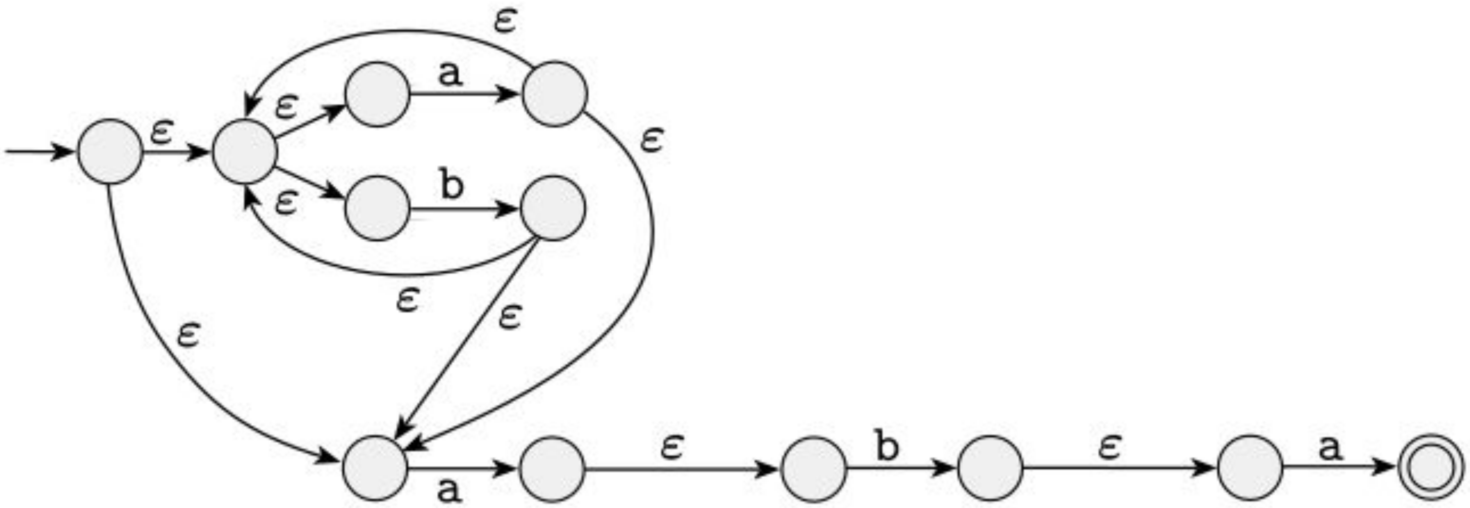
(a∪b)*



aba



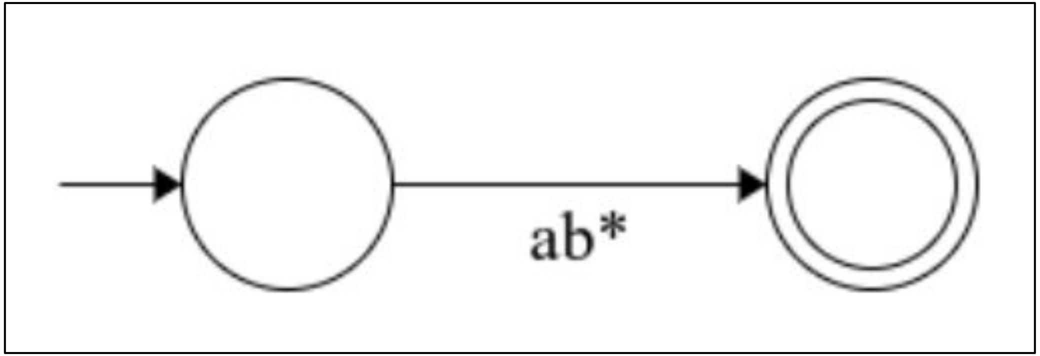
(a∪b)*aba



Regular Expressions

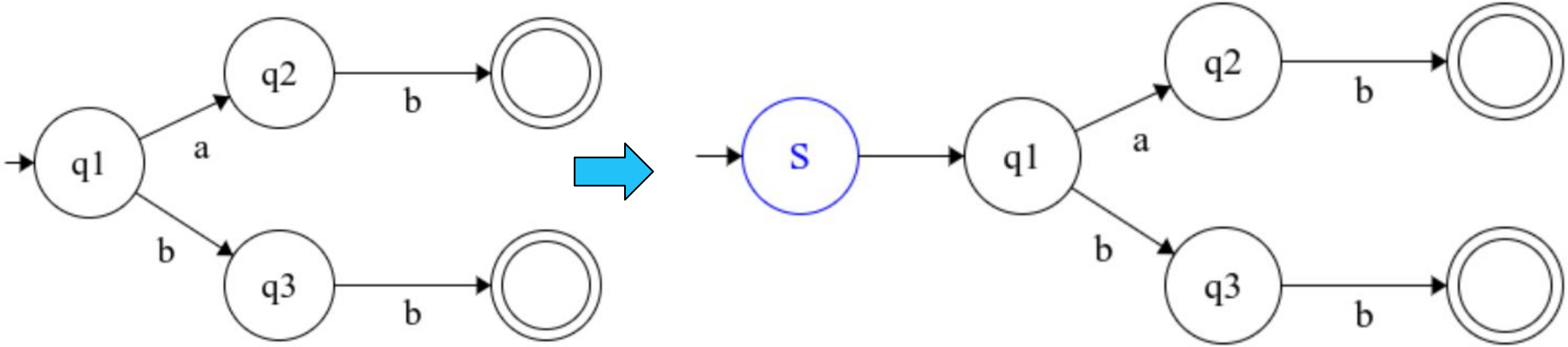
- **Converting NFA to Regular Expressions :**
  - If a language is regular, then it is described by a regular expression.
  - New type of finite automata called:

    “Generalized nondeterministic finite automaton : GNFA ”
    - **Transition can be expressed by Regular Expression**



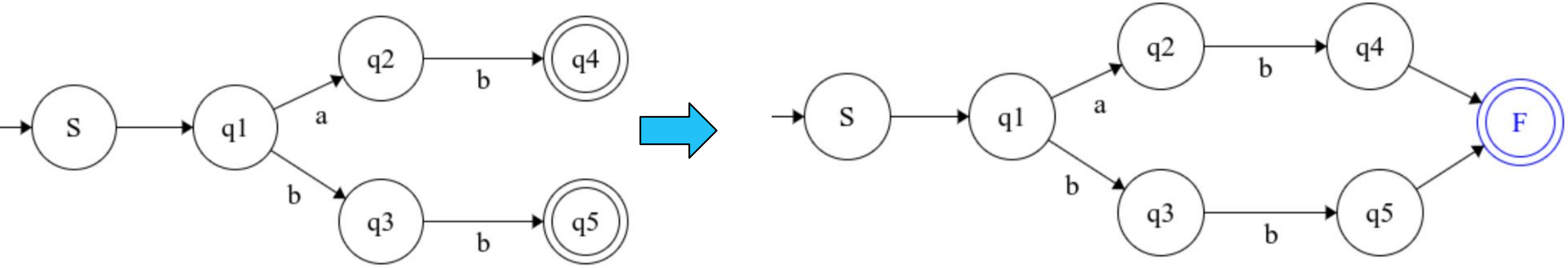
Regular Expressions

- **Converting NFA to Regular Expressions :**
  - **Step 1:**
    - Create a new start state and link it to the previous original start state with epsilon



Regular Expressions

- **Converting NFA to Regular Expressions :**
  - **Step 2:**
    - Create a new **single accept** state, link it to the previous original accept states with epsilon and turn the original accept states as **non-accept state**

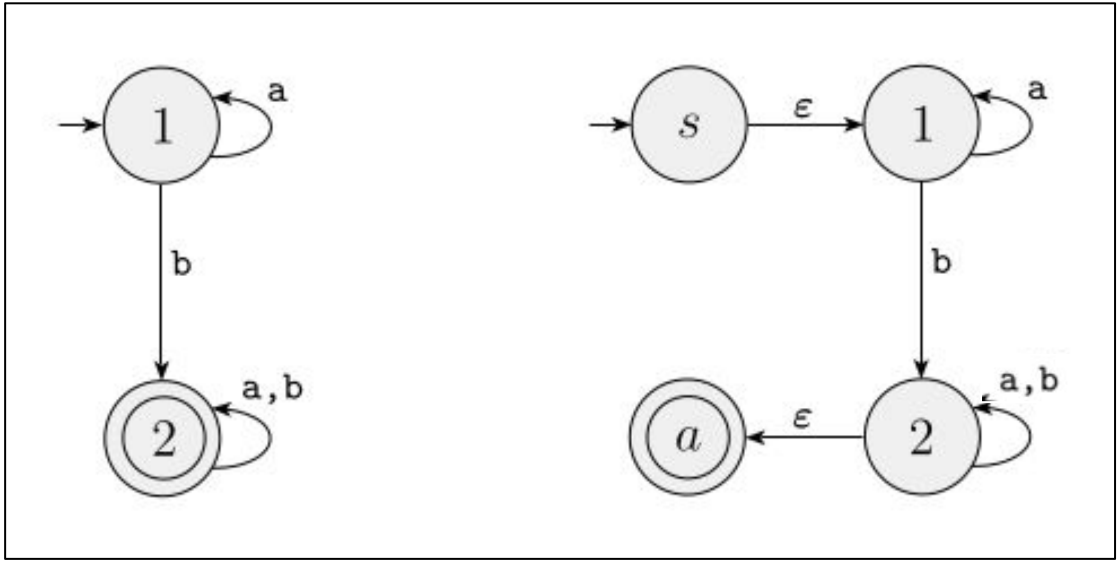


Regular Expressions

- **Converting NFA to Regular Expressions :**
  - **Step 3:**
    - Eliminate states and their transitions by observing possible patterns that you can make its regular expressions.

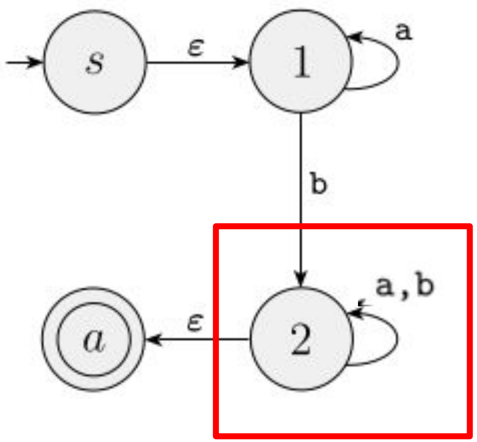
Regular Expressions

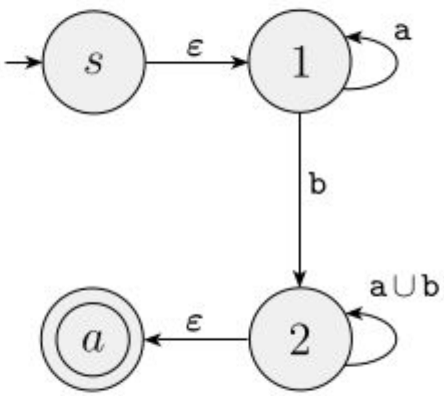
- **Converting NFA to Regular Expressions :**
  - **Example :**
    - **Start State**
    - **End Accepting State**



Regular Expressions

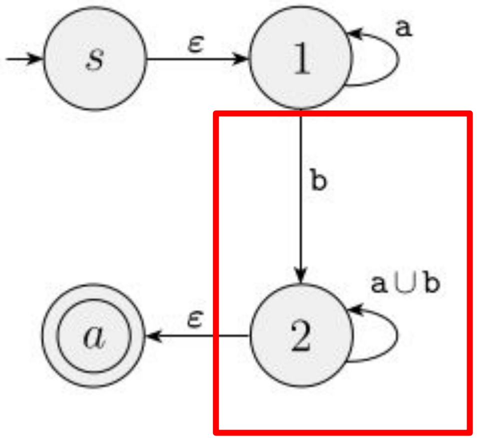
- **Converting NFA to Regular Expressions :**
  - **Example :**

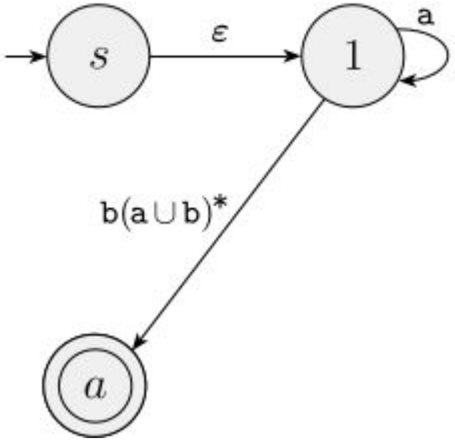




Regular Expressions

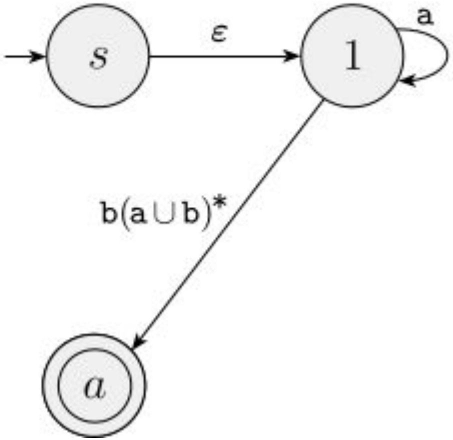
- **Converting NFA to Regular Expressions :**
  - **Example :**

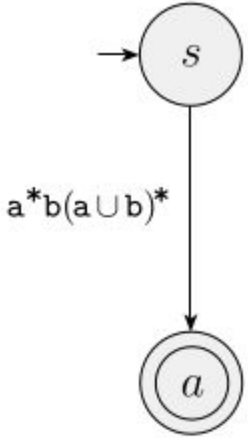




Regular Expressions

- **Converting NFA to Regular Expressions :**
  - **Example :**





Regular Expressions

- **RegEx on Linux or programming Languages :**
  - | : Union
  - ? : Zero or more
  - . : Any single character
  - + : One or more characters
  - ^ : Start of the line
  - $: End of the line
  - \s : Space
  - a{1,3} : The letter a : one or three times.
  - [a-z] : any letter from a to z
  - [0-9] : any digit

Regular Expressions

- **RegEx on Linux or programming Languages :**
  - | : Union
  - ? : Zero or more
  - . : Any single character
  - + : One or more characters
  - ^ : Start of the line
  - $: End of the line
  - \s : Space
  - a{1,3} : The letter a : one or three times.
  - [a-z] : any letter from a to z
  - [0-9] : any digit

Even though ! is the negation for RegEx

For this course, forget that we have the negation.

Regular Expressions

- **Useful websites to learn more regular expressions:**
  - https://regexlearn.com/learn
  - https://regexone.com/
  - https://regexr.com/

Questions

- How can we develop a system for a password strength checker ?
    - At least 8 chars
    - At least 3 special characters (#, ? , @ …)
    - At least one number
    - At least one capital letter

Questions

- Is formulating Regular Expressions easier than NFAs and DFAs ?

Questions

- Is formulating Regular Expressions easier than NFAs and DFAs ?
  - Formulate the RegEx for the language which includes all words with the exception of the word **a** :

    ({a,b}* - a)

Questions

- Is formulating Regular Expressions easier than NFAs and DFAs ?
    - Formulate the RegEx for the language which includes all words with the exception of the word **a** :

      You can think really hard to come up with the RegEx, but it is a complex task.

Questions

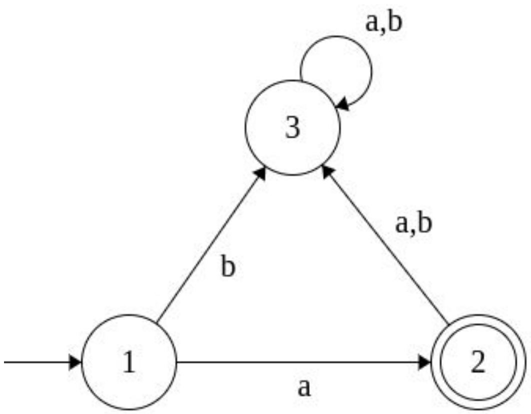
- Is formulating Regular Expressions easier than NFAs and DFAs ?
  - Formulate the RegEx for the language which includes all words with the exception of the word **a** **:**

    Better solution:

    - **Create the DFA for the easy language (language = {a})**
    - **Infer the DFA for the complement of the easy language**
    - **Convert to RegEx**

Questions

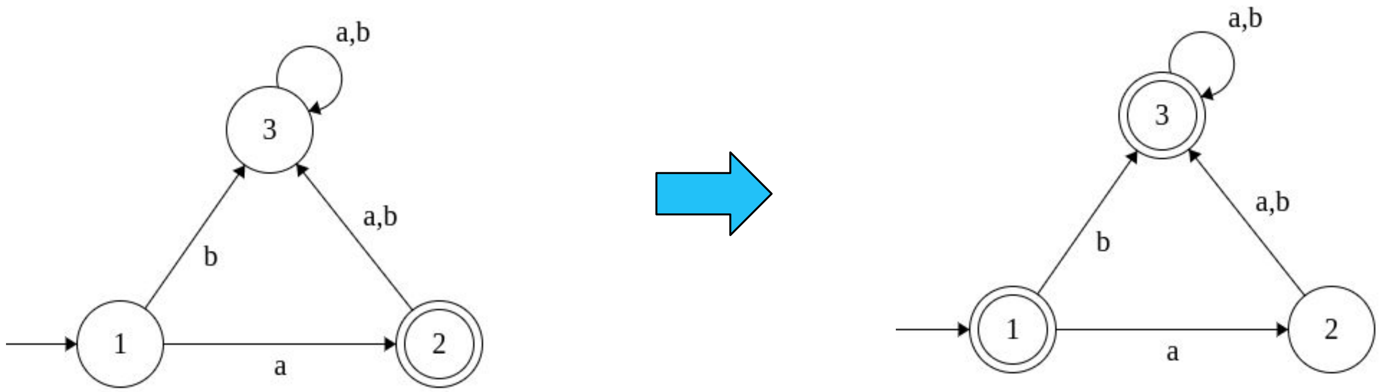
- Is formulating Regular Expressions easier than NFAs and DFAs ?
  - **1. Create the DFA for the easy language (language = {a})**



Questions

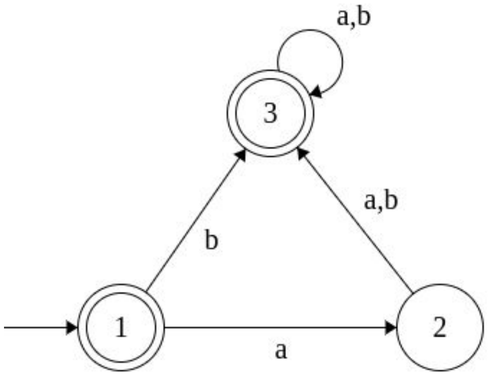
- Is formulating Regular Expressions easier than NFAs and DFAs ?
  - **2. Infer the DFA for the complement of the easy language**

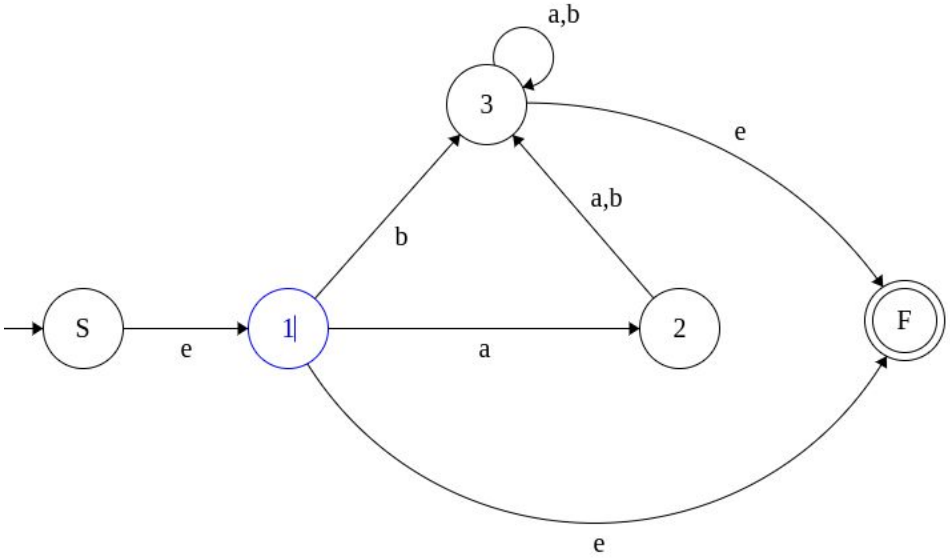
    {a} , its complement All words - {a}



Questions

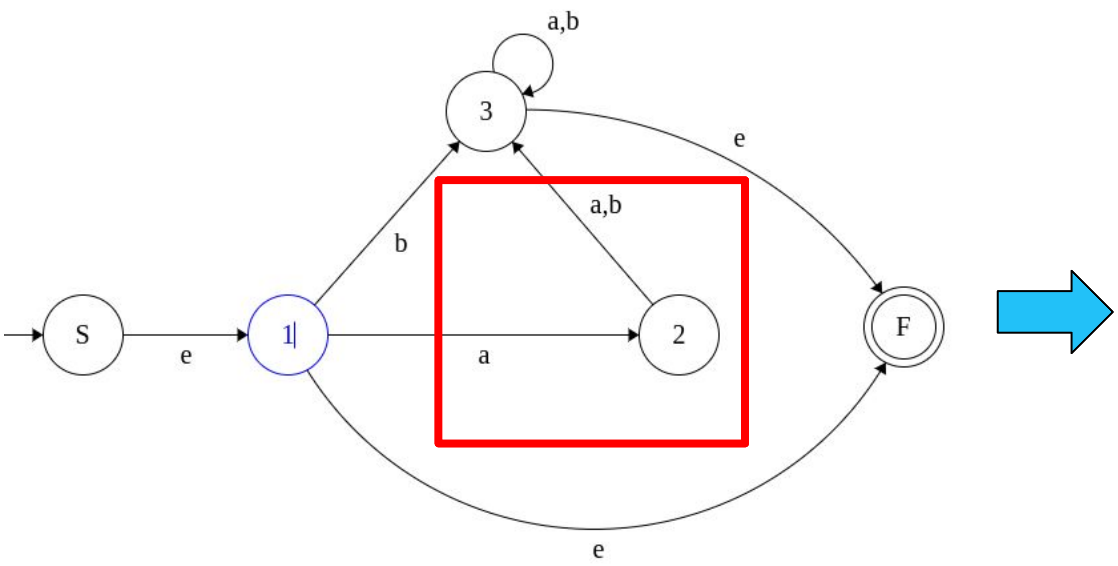
- Is formulating Regular Expressions easier than NFAs and DFAs ?
  - **3. Convert to RegEx**





Questions

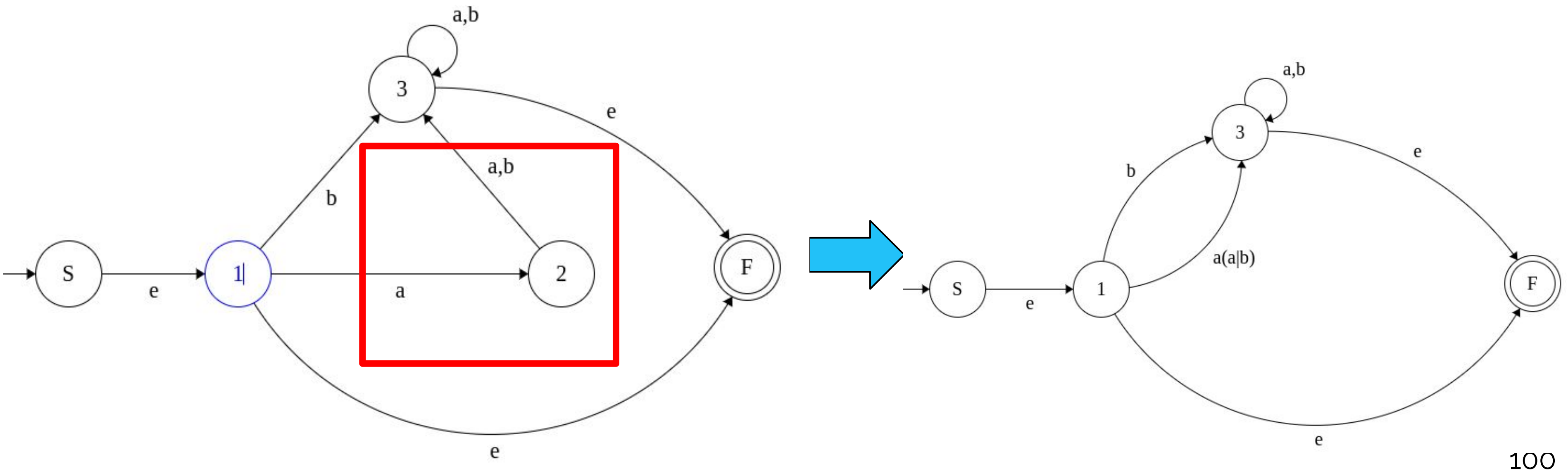
- Is formulating Regular Expressions easier than NFAs and DFAs ?



e : Epsilon

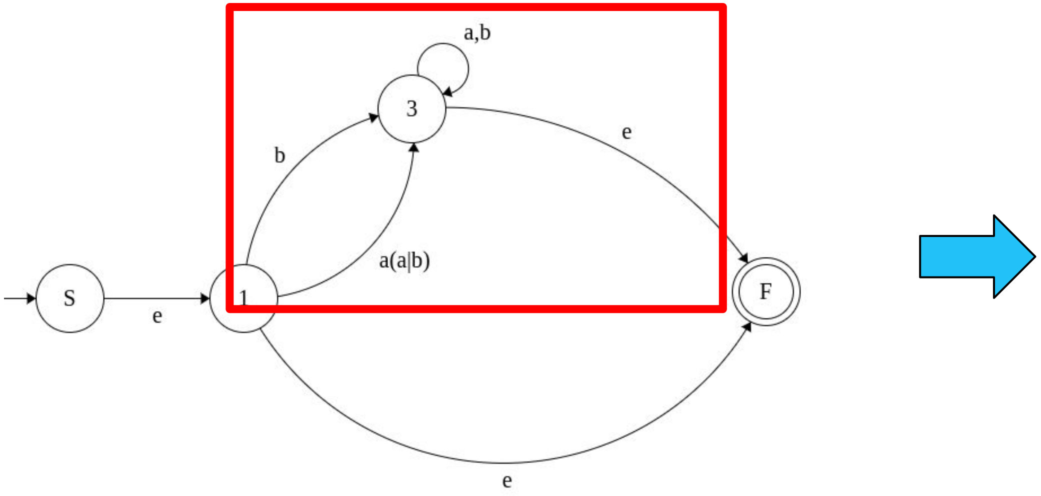
Questions

- Is formulating Regular Expressions easier than NFAs and DFAs ?



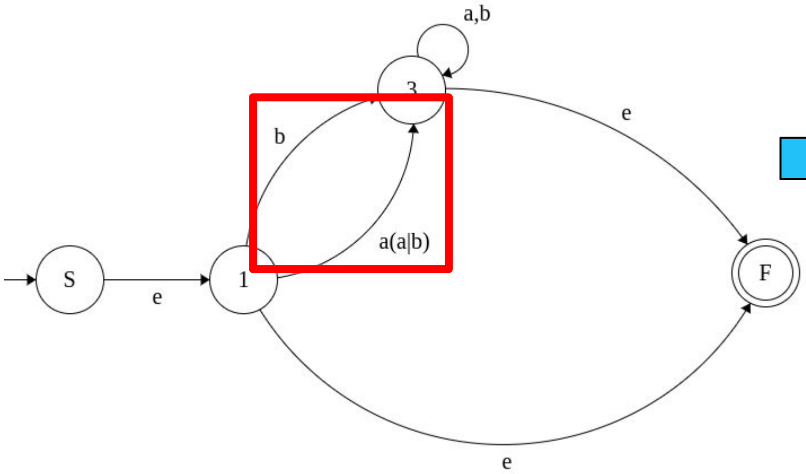
Questions

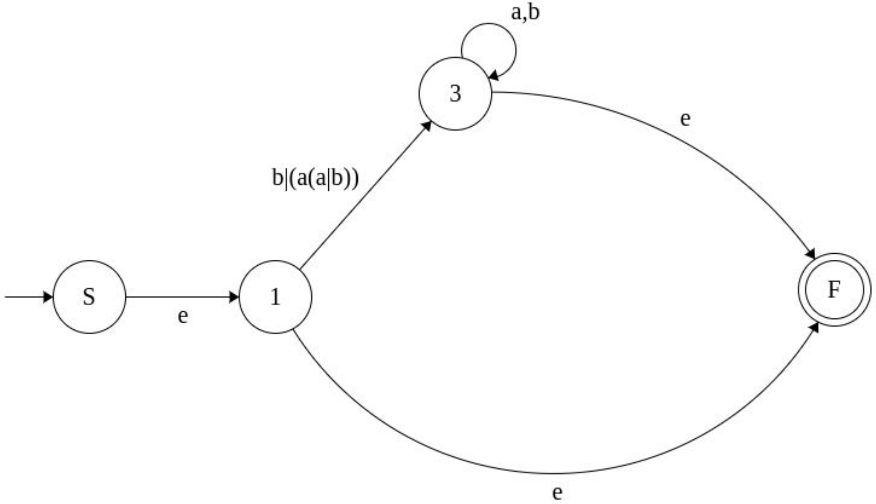
- Is formulating Regular Expressions easier than NFAs and DFAs ?
  - **3. Convert to RegEx**



Questions

- Is formulating Regular Expressions easier than NFAs and DFAs ?
  - **3. Convert to RegEx**

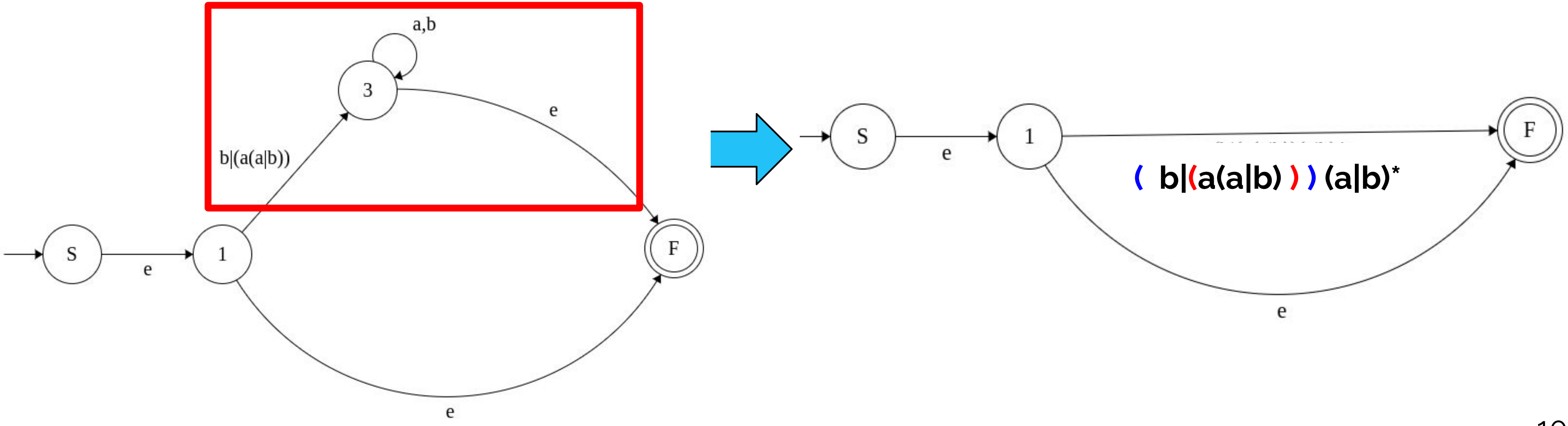




Questions

For Concatenation with Epsilon, we can drop Epsilon

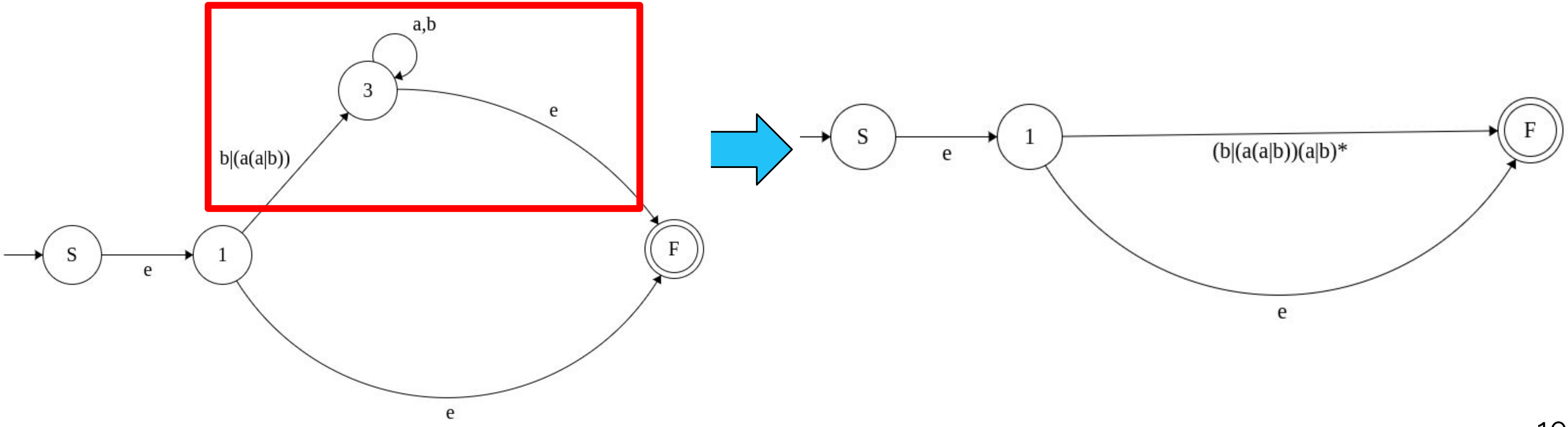
- Is formulating Regular Expressions easie
  - **3. Convert to RegEx**



Questions

- Is formulating Regular Expressions easie
    - **3. Convert to RegEx**

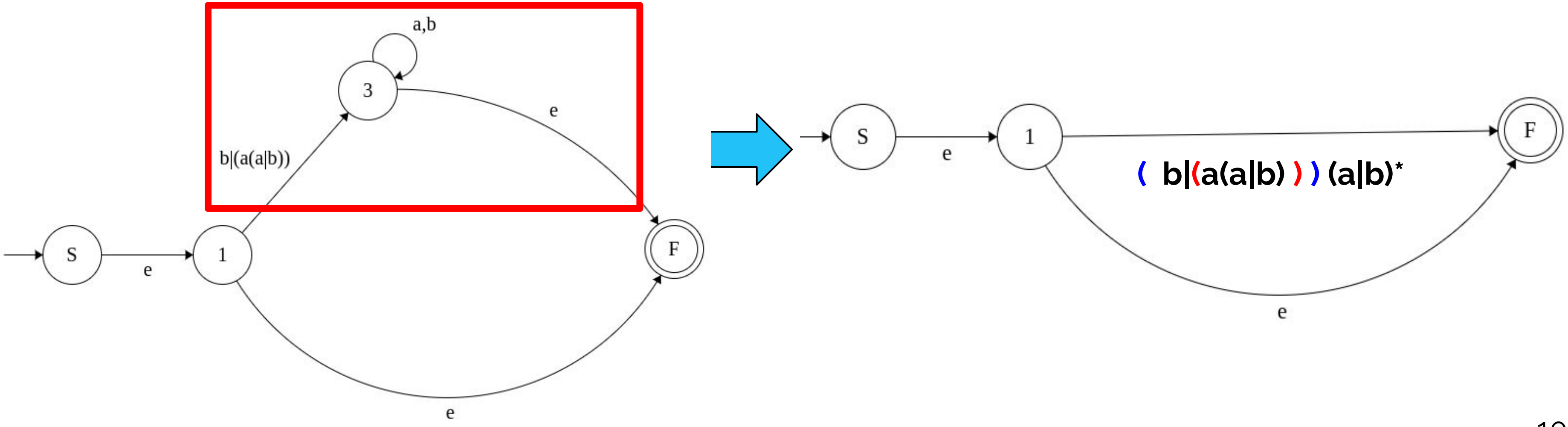
The use of well-colored parentheses is a must to have a readable syntax



Questions

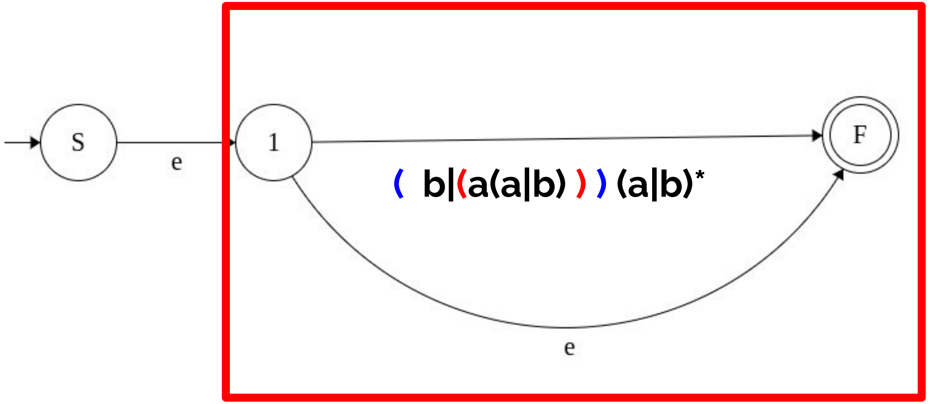
The use of well-colored parentheses is a must to have a readable syntax

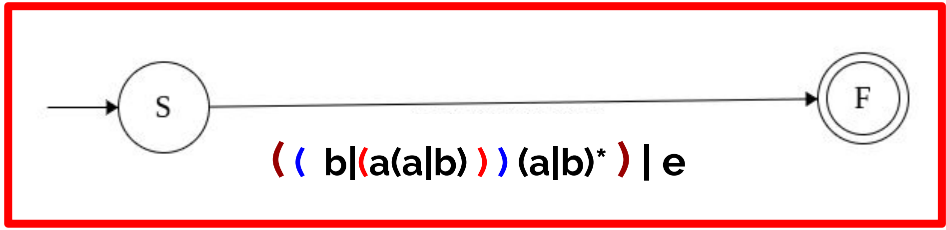
- Is formulating Regular Expressions easie
  - **3. Convert to RegEx**



Questions

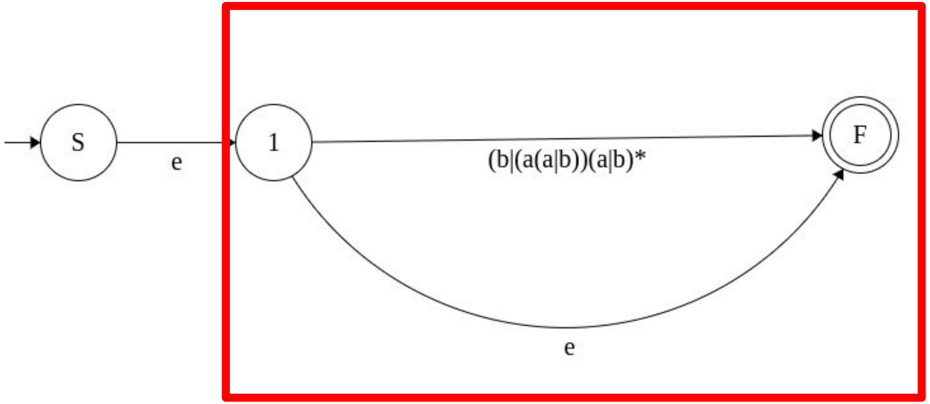
- Is formulating Regular Expressions easier than NFAs and DFAs ?
    - **3. Convert to RegEx**

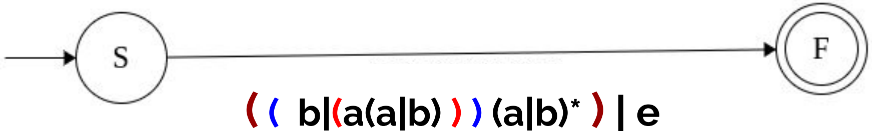




Questions

- Is formulating Regular Expressions easier than NFAs and DFAs ?
    - **3. Convert to RegEx : ((b|(a(a|b))) (a|b)*) | e**





Questions

- Is formulating Regular Expressions easier than NFAs and DFAs ?
  - **3. Convert to RegEx : ((b|(a(a|b))) (a|b)*) | e**
    - **Verify :**
      - **ε** in the language + accepted by RegEx
      - **a :** not in the language + not accepted by regex
      - **baaaaa .. :** in the language + accepted.
      - **aa** : ok
      - **baaaa : ??**

Questions

- Is formulating Regular Expressions easier than NFAs and DFAs ?
  - Formulate the RegEx for the language which includes all words with the exception of the words **a and b ?**

Questions by Students

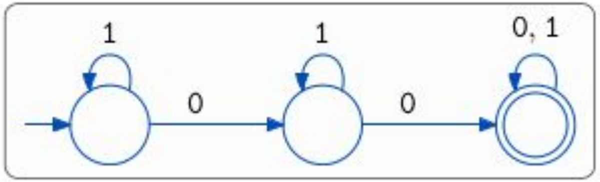
- For the concatenation:
    - How to concatenate :{a} with ∅

Question asked by Djamel BOUGHEDDOU , 2023/2024

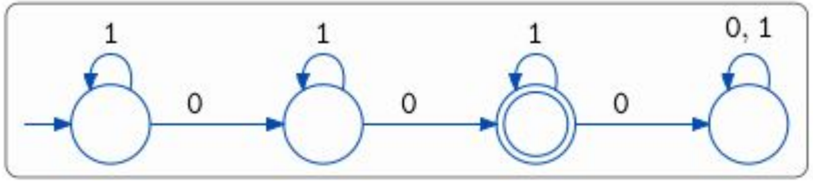
Solutions : TD 2

- **Ex 1 : Do also the RegEx on the fly for these exercises :**

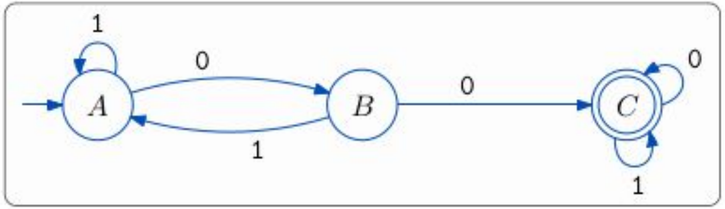
A. All strings containing at least two 0s



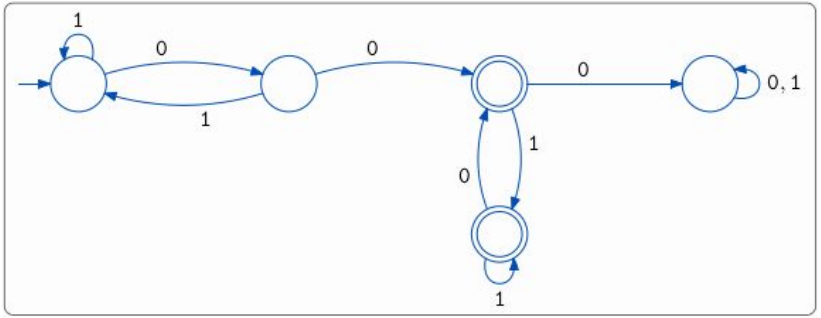
B. All strings containing exactly two 0s



C. All strings containing 00 as substring

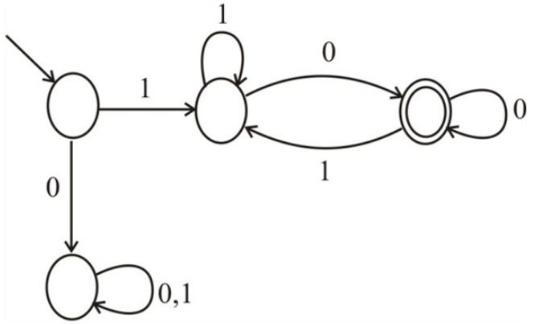


D. All strings containing 00 as substring exactly once

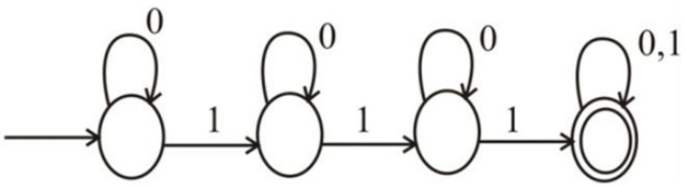


- **Ex 2 (TD 2)**

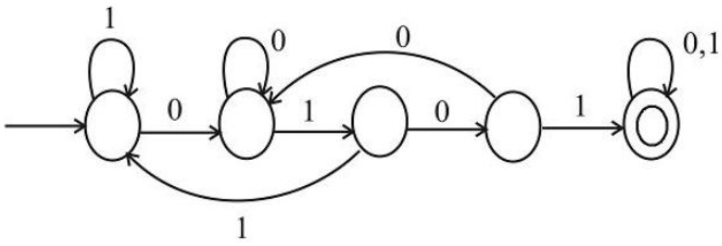
{w| w begins with a 1 and ends with a 0}



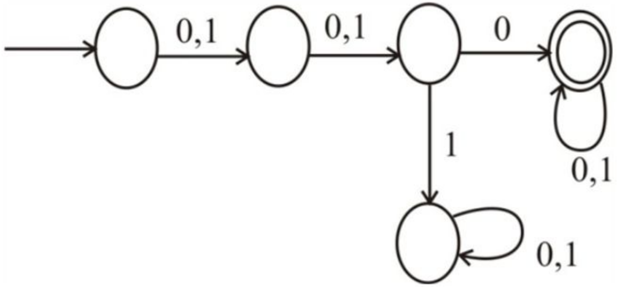
{w| w contains at least three 1s}



{w| w contains the substring 0101 (i.e., w = x0101y for some x and y)}



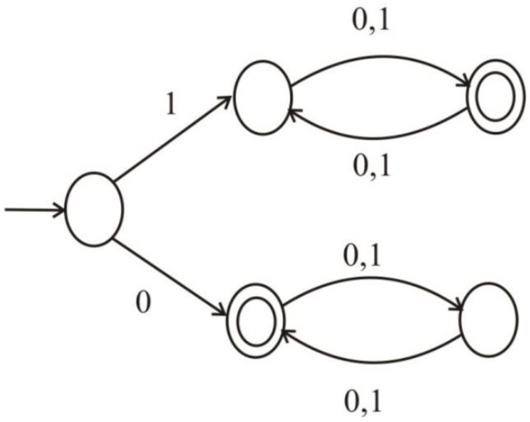
{w| w has length at least 3 and its third symbol is a 0}

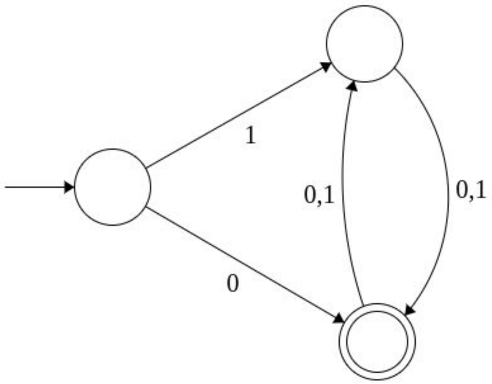


Solutions : TD 2

- **Ex 2**

`{w| w starts with 0 and has odd length, or starts with 1 and has even length}`

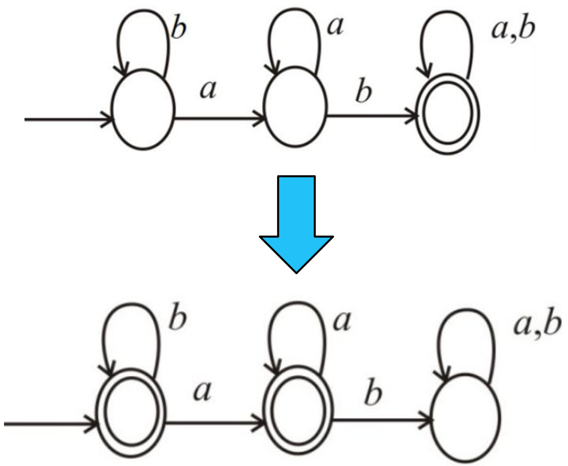




- # Ex 3 (TD 2)

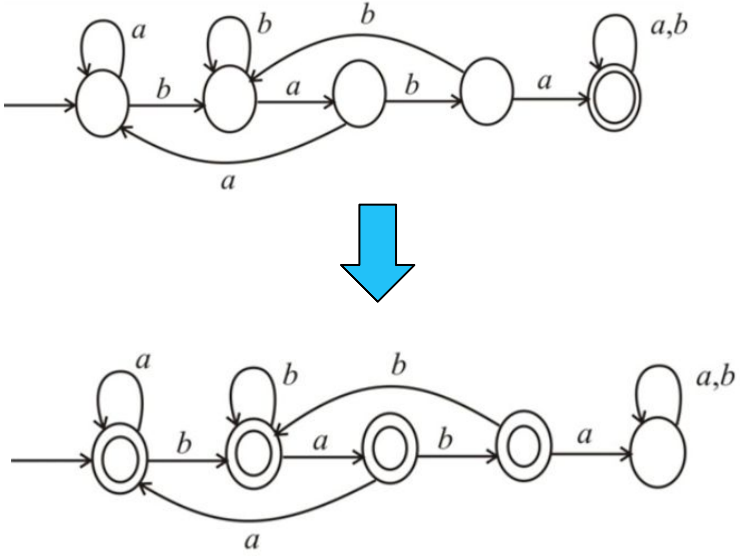
{w| w does not contain the substring ab}

Complement : {w| w contains the substring ab}



{w| w does not contain the substring baba}

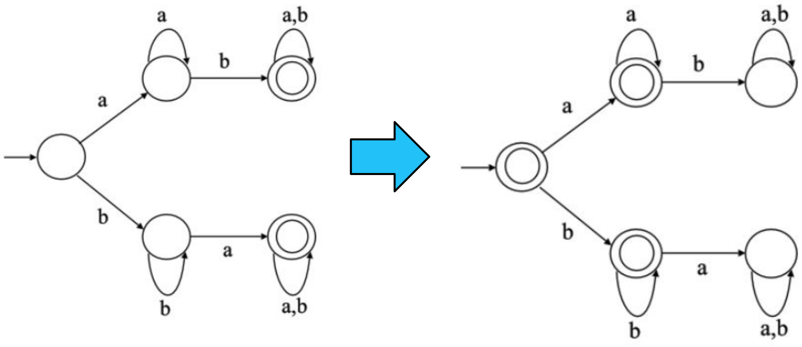
Complement : {w| w contains the substring baba}



- **Ex 3 (TD 2)**

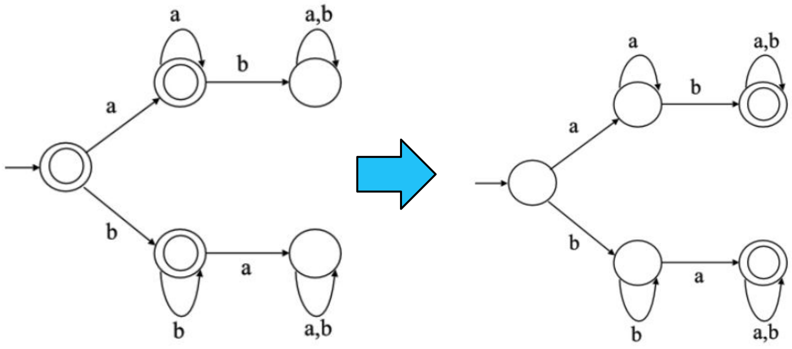
{w| w contains neither the substrings ab nor ba}

Complement : {w| w contains either the substrings ab or ba}



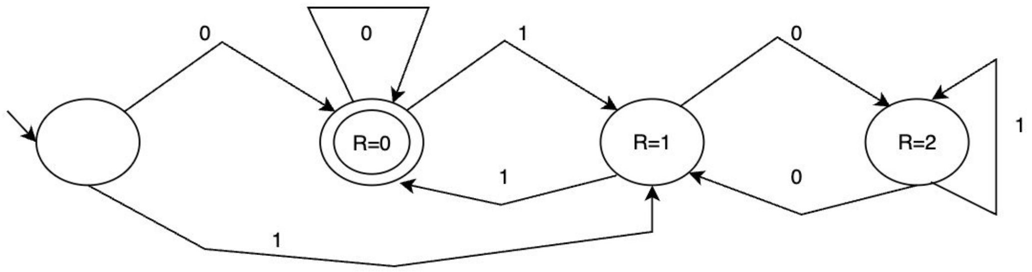
{w| w is any string not in a∗ ∪ b∗ }

Complement : {w| w is any string in a∗ ∪ b∗ }



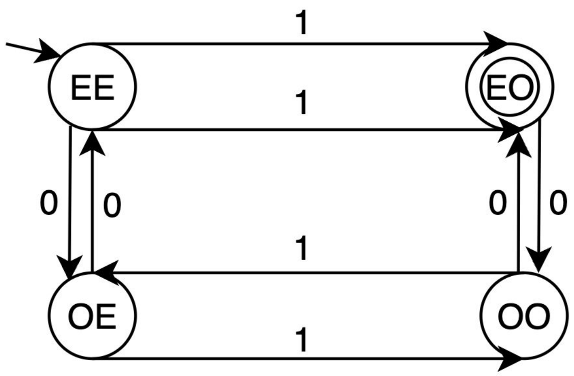
Solutions : Assignment 1

- **Ex 1.a**
  - L = {w | w is a binary string that is multiple of 3}. (i.e.the binary number when converted to decimal, it is multiple of 3).



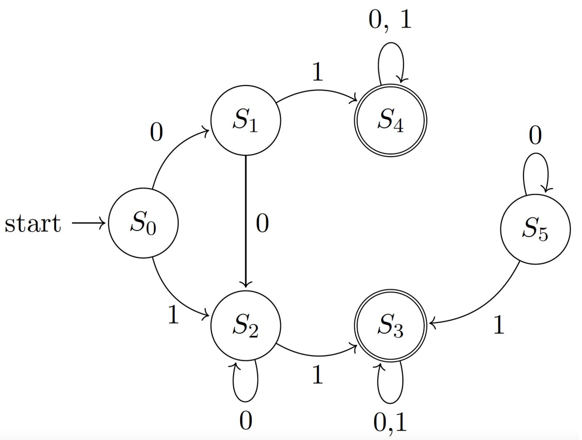
Solutions : Assignment 1

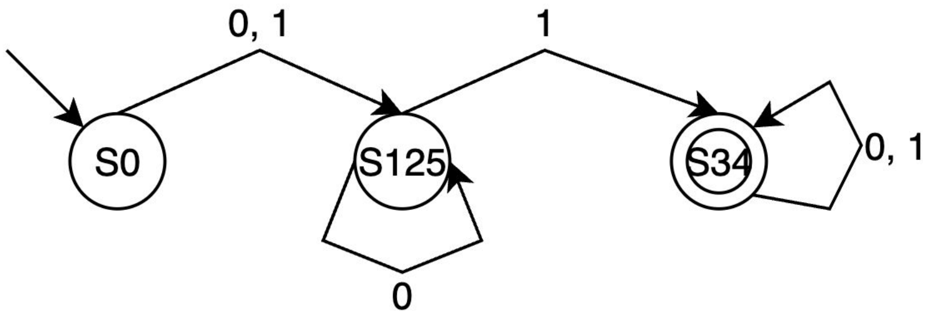
- **Ex 1.b**
  - `L = {w | w contains an even number of zeroes` **`and`** `an odd number of ones}.`



Solutions : Assignment 1

- **Ex 2**
  - Minimization of DFA





Midterm Skeleton