Theory of Computing :

6. Context Free Grammars

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

Outline :

- Regular and Non-regular languages
  - Pumping Lemma
  - Closure Properties
  - Examples
- Context-Free Grammars
  - Reading grammars
  - Creating grammars
- Derivation Trees
- Chomsky Normal Form

Problem Solving and Languages

- **Encoding Problems into a Language..**
  - There are decisional problems that we need to automate the way to solve them
    - *We encode them as a language*
    - *We create the Automaton for the language that we have created.*
    - *To solve an instance of the problem,*
      - *Encode the instance as a word*
      - *The machine will automatically recognize the word in the language or not in the language*

Problem Solving and Languages

- **Encoding Problems into a Language..**
  - Calculator : how to validate it is a valid format ?
  - 1+2=
  - 1/2=
  - +-2+++1=
  - *1+1=
  - -1+1=

Problem Solving and Languages

- **Encoding Problems into a Language..**
  - Calculator : how to validate it is a valid format ?
  - 1+2=
  - 1/2=
  - +-2+++1=
  - *1+1=
  - -1+1=

(+|-|ε) [0-9]⁺ (+ | - | /(-|ε) | x(-|ε)) [0-9]⁺ =

Problem Solving and Languages

- **Encoding Problems into a Language..**
  - Calculator : how to validate it is a valid format ?
  - 1+2=
  - 1/2=
  - +-2+++1=
  - *1+1=
  - -1+1=

This is a regular language

Problem Solving and Languages

- **Encoding Problems into a Language..**
  - Calculator : how to validate it is a valid format ? Scientific or advanced ?
  - (1+2)=
  - (4*(2+3))/2=

Regular Languages

- **Regular Languages.**
    - Is the set of languages which can be represented by a deterministic finite automaton (or NFA or RegEx)

Regular Languages

- **Proving that a Language is Regular.**
  - Pumping Lemma ?

Regular Languages

- **Proving that a Language is Regular.**
  - Either by Creating:
    - DFA
    - NFA
    - RegEx

Pumping Lemma can be never be used to prove that a language is regular

Regular Languages

- **Proving that a Lang**
  - L = { w | w is a p
  - We assume it i
  - We take s=xyz
    - We can take
    - Now, let's try to pump by repeating Y: (Let's assume |y|=k)
      - xy^2z ,→ $1^{P+k}1^P$ (it is in language)
      - xy^3z ,→ $1^{P+k+k}1^P$ (it is also in language)
      - xy^iz ,→ $1^{P+ik}1^P$ (it is also in language for any y and at any i)

Does it mean that the language is regular ?

Regular Languages

- **Proving that a Language is Regular.**
  - L = { w | w is a palindrome }
  - We assume it is regular
  - We take s=xyz = $1^{P}1^{P}$ and |xy| <= P
    - We can take xy must contain only 1s,
    - Now, let's try to pump by repeating Y: (Let's assume |y|=k)
      - xy^2z ,→ $1^{P+k}1^{P}$ (it is in language)
      - xy^3z ,→ $1^{P+k+k}1^{P}$ (it is also in language)
      - xy^iz ,→ $1^{P+ik}1^{P}$ (it is also in language for any y and at any i)

P

- **Prov**
  - ○
  - ○
  - ○

Pumping Lemma can be never be used to prove that a language is regular

If you find a repeating pattern, it is because for another different regular language which is a subset in the first language .

1^* is a subset of palindromes.

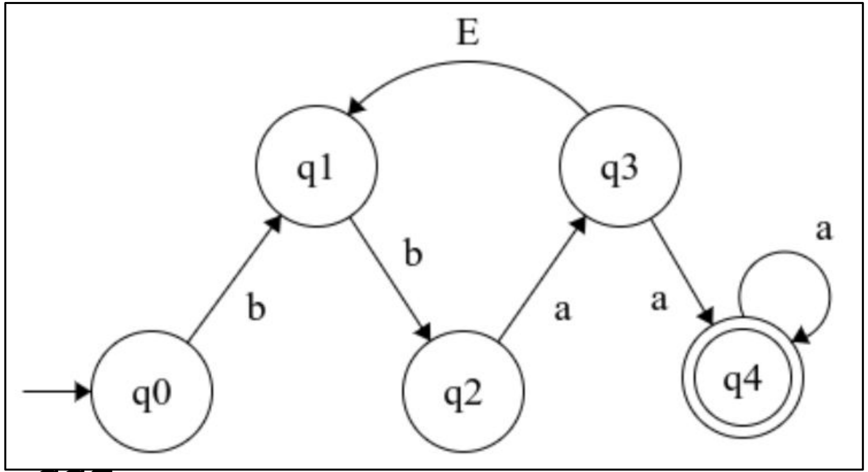
- xy^iz ,→ $1^{P+ik}1^P$ (it is also in language for any y and at any i)

Pumping Lemma

- **Proving that a language is non-regular**
  - Must be infinite
    - Because every finite language is regular since we can construct the regex : word1 | word2 | word3 …. Wordn
  - We use the pumping lemma where we assume that the language is regular
    - There must be a **loop / pumping mechanism** to generate infinite number of words **in the language**
    - We arrive that there is no such mechanism → Contradiction to our assumption → We conclude that the language is non-regular

Pumping Lemma

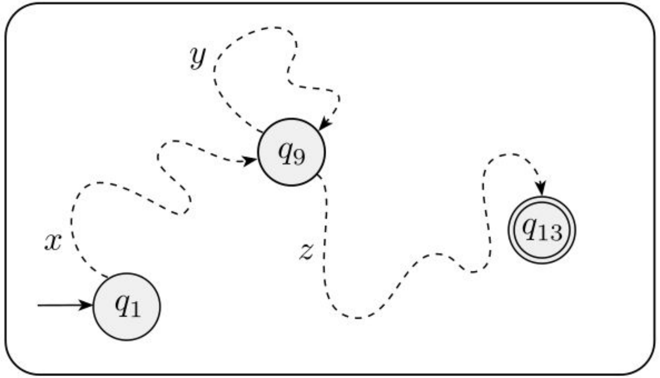
- **Pumping Words**
  - Given a finite state machine of N states (suppose N=5):
    - We can generate:
      - bbaa
      - s=xyz = bba**ba**aa
      - s=xyyz =xy^2z = bba**baba**aa
      - s= xy^3z = bba**bababa**aa
      - s= xy^4z = bba**babababa**aa
      - s= xy^iz =bba**bababababa........**aa**a...**



    - Infinite number + infinite length

Pumping Lemma

- **Proving that a language is non-regular**
  - For a language to be infinite and regular there must be a pumping mechanism
  - And if we take s=xyz, we can pump y to generate more words.
  - ...

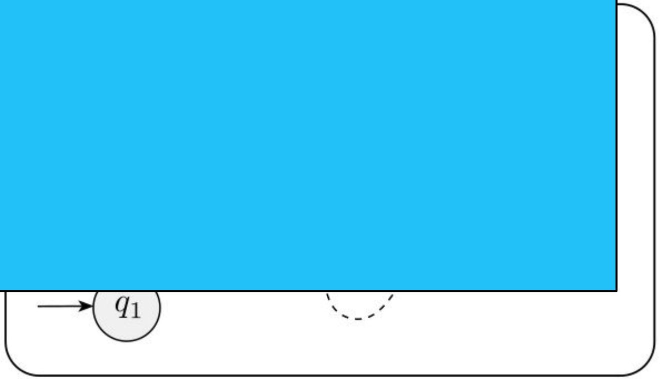


Pumping Lemma

- **Proving that a language is non-regular**
  - For a language to be infinite and regular there must be a pumping n
  - A
  - ...

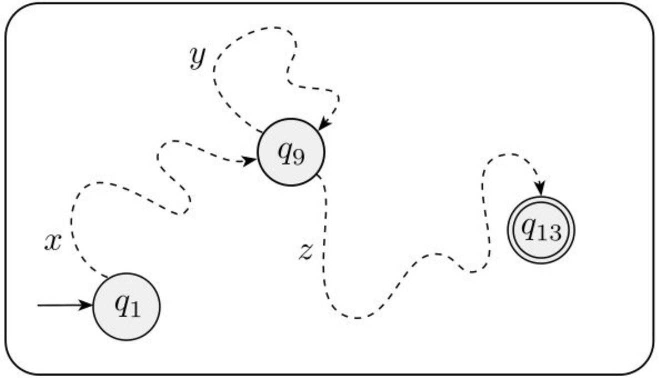
Important, You can never never take a specific value for P

(P=10)



Pumping Lemma

- **Proving that a language is non-regular**
  - Blindly assume that the language $0^{100}11^n$ is not regular (for n>=0)
  - We use the pumping lemma to do so:
    - $s=0^{100}1^p = xyz$,
    - If we assume that P=10, |xy| <=10,
    - xy must be in the first 10 zeros and y Should be zeros.
    - If we pump y, we will increase the number of zeros and exceed **100**
    - Therefore, the words generated are not in the language



Pumping Lemma

- Pr

?

We have used the pumping lemma incorrectly by fixing the value of P to prove regular as non-regular

Yes, there is no pumping mechanism before P=100, but there is after

- Therefore, the words generated are not in the language

Non-Regular Languages Closure Property

- **Using property of Regular Languages:**
  - Closure Property for Non-regular Languages :
    - $D = \{w \mid w = a^m b^n , m \neq n\}$ (Non-regular)
    - $B = \{ a^n b^n : n \geq 0 \}$ (Non-regular)
    - $M = \{ a^n b^{2n} : n \geq 0 \}$ (Non-regular)
    - $D \cup B = a^* b^*$ (Regular)
    - Regular U Regular ⇒ **Must** be regular :
    - Non-regular U Regular ⇒ **We don't know (**$\{ a^n b^n : n \geq 0 \} \cup (a|b)^*$? **)**
    - Non-regular U Non-regular ⇒ **We don't know**
      - *$\{w \mid w = a^m b^n , m \neq n\} \cup \{ a^n b^n : n \geq 0 \} = a^* b^*$ ⇒ Regular*
      - *$\{ a^n b^n : n \geq 0 \} \cup \{ a^n b^{2n} : n \geq 0 \}$ ⇒ Non-Regular*

Quiz Questions

- **Say whether the following languages are regular or non-regular**
  - $L = \{xww^R y \mid x, y \in \Sigma^* \text{ and } |w|, |x|, |y| \geq 1\}$
  - $L = \{xww^R y \mid x, y \in \Sigma^* \text{ and } |w|, |x|, |y| \geq 1\}$

Quiz Questions

- **Say whether the following languages are regular or non-regular**
  - $L = \{xww^R y \mid x, y \in \Sigma^* \text{ and } |w|, |x|, |y| \geq 1\}$
    - $(\Sigma^*00\Sigma^*) \mid (\Sigma^*11\Sigma^*)$ (There is x and y to absorb letters)
  - $L = \{xww^R \mid x, y \in \Sigma^* \text{ and } |w|, |x|, |y| \geq 1\}$
    - No DFA, no RegEx, it is non-regular
      - ⇒ use the pumping lemma:
      - ⇒ $s = 00^P10^P$ $|xy| <= P$ … Pump Y → always more zeros on the left side → word not in the language

In contrast to the first example :

If you pump, you get always words in the language

(though, don't use pumping lemma to prove that a language is regular)

- **Say whether the following languages are regular or non-regular**
  - $L = \{xww^R y \mid x, y \in \Sigma^* \text{ and } |w|, |x|, |y| \geq 1\}$
    - $(\Sigma^*00\Sigma^*) \mid (\Sigma^*11\Sigma^*)$ (There is x and y to absorb letters)
  - $L = \{xww^R \mid x, y \in \Sigma^* \text{ and } |w|, |x|, |y| \geq 1\}$
    - No DFA, no RegEx, it is non-regular
      - ⇒ use the pumping lemma:
      - ⇒ $s = 00^P10^P$ $|xy| \le P$ … Pump Y → always more zeros on the left side → word not in the language

Examples from Non-regular languages

- **Representing a language**
  - Given the following language, are they regular ?
    - L = { w | contains an equal number of 1s and 0s }
      - *Can we create a finite state machine ?*
      - *Can we create a regular expression ?*
        - ***Better to use one of the techniques to prove the language is non regular : Pumping Lemma, fooling sets or closure properties.***

Examples from Non-regular languages

- **Representing a language**
  - Given the following language, are they regular ?
    - L = { w | contains an equal number of 1s and 0s }
      - *The language is non regular*
      - *Mainly due to the limitation of the finite state machine*
    - What's next ?

Context-Free Grammar

- **Definitions**
  - Alphabet :
  - Language :
  - Abstract Machine or Computational Model :
  - Grammar :

Context-Free Grammar

- **Definitions**
  - Alphabet :
  - Language :
  - Abstract Machine or Computational Model :
  - Grammar : **is a set of rules for combining symbols from the alphabet to create strings that belong to a specific language.**

Context-Free Grammar

- **Grammar consists of :**
  - A set of **variables** (also called nonterminals),
    - one of which is designated the start variable;
    - It is customary to use **UPPER-CASE** letters for variables
  - a set of terminals (from the alphabet)
  - a list of productions (also called rules or **substitution rules**).

Context-Free Grammar

- **Reading Grammar**
  - $S \rightarrow 0S$
  - $S \rightarrow \varepsilon$
    - **Start variable** : S, it is the only variable
    - **0** is a terminal symbol
    - There are two productions.
      - S produces 0S ⇒ we keep substituting **recursively** until we reach the word we are after.
  - What's the language for this grammar ?

Context-Free Grammar

- **Reading Grammar**
  - $S \rightarrow 0S$
  - $S \rightarrow \varepsilon$
    - What's the language for this grammar ?
    - Let's generate some words:
      - 0 , 00, 000, 0000, 0000, ….

Context-Free Grammar

- **Reading Grammar**
  - $S \rightarrow 0S$
  - $S \rightarrow \varepsilon$
    - What's the language for this grammar ?
    - Let's generate some words:
      - ε, 0 , 00, 000, 0000, 0000, ….

Context-Free Grammar

- **Reading Grammar**
  - $S \rightarrow T\,1\,1\,T$
  - $T \rightarrow 0T \mid 1T \mid \varepsilon$
    - $|$: means **or**
      - What's the language for this grammar ?

Context-Free Grammar

- **Reading Grammar**
  - S → T 1 1 T
  - T → 0T | 1T | ε
    - | : means **or**
      - What's the language for this grammar ?
        - 11
        - 0101011001

Context-Free Grammar

- **Reading Grammar**
    - $S \rightarrow 0S1$
    - $S \rightarrow \varepsilon$
        - What's the language for this grammar ?
            - S is the only variable.
            - The terminals are 0 and 1.
            - There are two production rules.

Context-Free Grammar

- **Reading Grammar**
    - $S \rightarrow 0S1$
    - $S \rightarrow \varepsilon$
        - What's the language for this grammar ?
            - ε
            - 01
            - 0011
            - 000111
            - 00001111

Context-Free Grammar

- **Reading Grammar**
  - $S \rightarrow 0S1$
  - $S \rightarrow \varepsilon$
    - What's the language for this grammar ?
      - **For language $L = \{ 0^n1^n \mid n >= 0 \}$**

Context-Free Grammar

- **Reading Grammar**
  - $S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$
    - What's the language for this grammar ?

Context-Free Grammar

- **Reading Grammar**
  - $S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$
    - What's the language for this grammar ?
      - Let's try to generate some words:
        - a , b, aa , bb, aba, bab, abba, abaaba.....

Context-Free Grammar

- **Reading Grammar**
  - $S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$
    - What's the language for this grammar ?
      - For the language $L = \{ w \mid w = w^R \}$ palindromes over $\{a,b\}$

Context-Free Grammar

- **Deriving a String from a Grammar**
  - The sequence of substitutions to obtain a string is called a **derivation**
  - S → aSa | bSb | a | b | ε
    - Let's drive the word : abababa
    - S ⇒ aSa [*rule S → aSa*]
    - ⇒ a**bSb**a [*rule S → bSb*]
    - ⇒ ab**aSa**ba [*rule S → aSa*]
    - ⇒ aba**b**aba [*rule S → b*]

Context-Free Grammar

- **Deriving a String from a Grammar**
  - The sequence of substitutions or steps to obtain a string is called a **derivation**
    - A **leftmost derivation** is where at each stage one replaces the leftmost **variable**.
    - A **rightmost derivation** is defined similarly

Context-Free Grammar

- **Deriving a String from a Grammar**
  - **Important Notation:**

    language of the grammar is $\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$.

      - It reads : S **yields** or **generates** w (w as a word)

Context-Free Grammar

- **Creating Grammar**

DESIGNING CONTEXT-FREE GRAMMARS

As with the design of finite automata, discussed in Section 1.1 (page 41), the design of context-free grammars requires creativity. Indeed, context-free grammars are even trickier to construct than finite automata because we are more

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 00 *11 *
  - Produce the grammar for it

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - a
  - Produce the grammar for it

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - a
  - Produce the grammar for it
    - $S \rightarrow a$

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 00 *11 *
  - Produce the grammar for it
    - S →

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 00 *11 *
  - Produce the grammar for it
    - $S \rightarrow CD$

C for 00^*
D for 11^*

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 00 *11 *
  - Produce the grammar for it
    - S → CD
    - C → 0 ?
    - D → 1 ?

C for 00^*
D for 11^*

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 00 *11 *
  - Produce the grammar for it
    - S → CD
    - C → 0C | 0
    - D → 1D | 1

Recursion is important

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - (0 | 1)*
  - Produce the grammar for it

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - (0 | 1)*
  - Produce the grammar for it
    - $S \rightarrow 0S \mid 1S \mid \varepsilon$

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - (0 | 1)* 00 (0 | 1)*
  - Produce the grammar for it

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - (0 | 1)* 00 (0 | 1)*
  - Produce the grammar for it
    - S → A 00 A
    - A →

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - $(0 \mid 1)^* \, 00 \, (0 \mid 1)^*$
  - Produce the grammar for it
    - $S \rightarrow A \, 00 \, A$
    - $A \rightarrow 0A \mid 1A \mid \varepsilon$

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - (11 | 00) *11
  - Produce the grammar for it

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - (11 | 00) *11
  - Produce the grammar for it
    - S → TU
    - T → VT | e
    - V → 00 | 11
    - U → 11

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 110(01)* | (01 | 10* 1)* 10.
  - Produce the grammar for it

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 110(01)* | (01 | 10* 1)* 10
  - Produce the grammar for it
    - S → anything | anything

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 110(01)* | (01 | 10* 1)* 10
  - Produce the grammar for it
    - S → A | B

A for 110(01)*
B for (01 | 10* 1)* 10

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 110(01)* | (01 | 10* 1)* 10
  - Produce the grammar for it
    - $S \rightarrow A \mid B$
    - $A \rightarrow 110$?

How to represent (01)*

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 110(01)* | (01 | 10* 1)* 10
  - Produce the grammar for it
    - S → A | B
    - A → 110 **C**
    - **C → ε | 01C**

How to represent (01)*

Next : B ?

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 110(01)* | (01 | 10* 1)* 10
  - Produce the grammar for it
    - S → A | B
    - A → 110 C
    - C → ε | 01C
    - **B → D 10**
    - **D →01D | E1 | ε**

How to represent (01)*

Next : B ?

Context-Free Grammar

- **Creating Grammar**
  - Given the following Regular Expression :
    - 110(01)* | (01 | 10* 1)* 10
  - Produce the grammar for it
    - S → A | B
    - A → 110 C
    - C → ε | 01C
    - B → D 10
    - D →01D | E1D | ε
    - E → 10E | ε

Context-Free Grammar

- **Creating Grammar**
  - Given the following language:
    - $L = \{w \mid n_a(w) = n_b(w)\}$
  - Produce the grammar for it

Context-Free Grammar

- **Creating Grammar**
  - Given the following language:
    - $L = \{w \mid n_a(w) = n_b(w)\}$
  - Produce the grammar for it
    - $S \rightarrow$

Context-Free Grammar

- **Creating Grammar**
  - Given the following language:
    - $L = \{w \mid n_0(w) = n_1(w)\}$
  - Produce the grammar for it
    - $S \rightarrow \varepsilon$

Start always with the obvious cases

Context-Free Grammar

- **Creating Grammar**
  - Given the following language:
    - $L = \{w \mid n_0(w) = n_1(w)\}$
  - Produce the grammar for it
    - $S \rightarrow \varepsilon \mid S0S1S \mid S1S0S$

S is like the * operator

Context-Free Grammar

- **Creating Grammar**
  - Given the following language:
    - $L = \{w \mid n_0(w) = n_1(w)\}$
  - Produce the grammar for it
    - $S \rightarrow \varepsilon \mid S0S1S \mid S1S0S$
  - Let's test :
    - Acceptance : 0011, 1100, 1001
    - Rejection : 0001, 110

Test your grammar for Acceptance and Rejection

Context-Free Grammar

- **Creating Grammar**
  - Given the following language:
    - $L = \{w \mid n_0(w) = n_1(w)\}$
  - Produce the grammar for it
    - $S \rightarrow \varepsilon \mid S\mathbf{0}S\mathbf{1}S \mid S\mathbf{1}S\mathbf{0}S$
  - What about the following grammars ? do they represent the language ?
    - $S \rightarrow 0S1S \mid 1S0S \mid \varepsilon$
    - $S \rightarrow 0S1 \mid 1S0 \mid SS \mid \varepsilon$

Context-Free Grammar

- **Creating Grammar**
  - Given the following language:
    - { ε }
  - Produce the grammar for it
    -

Context-Free Grammar

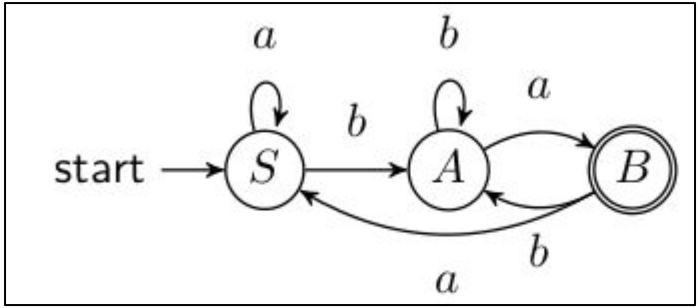
- **Creating Grammar**
  - Given the following language:
    - Alternating 0 and 1
  - Produce the grammar for it
    -

Context-Free Grammar

- **Creating Grammar**
  - Given the following language:
    - { } (Phi)
  - Produce the grammar for it
    -

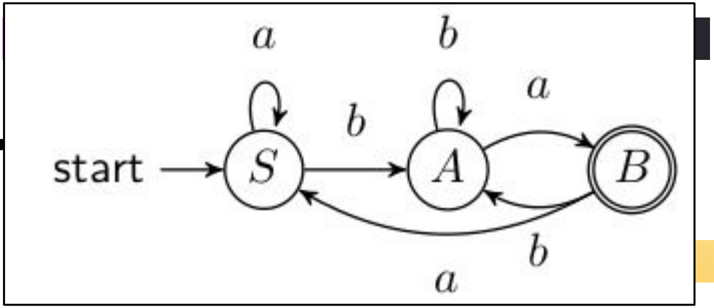
Context-Free Grammar

- **Creating Grammar**
  - Given the following language represented by a DFA :



  - Produce the grammar for it

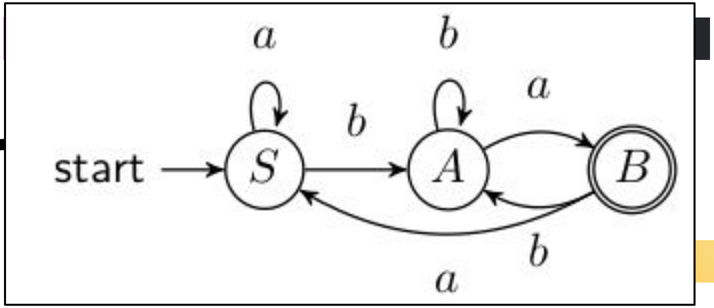
Context-Free Grammar



- **Creating Grammar**
  - Given the following language represented by a DFA :
  - Produce the grammar for it
    - S → a**S** | b**A**

Use
- **states as Variables**
- **Transition as terminal symbols**

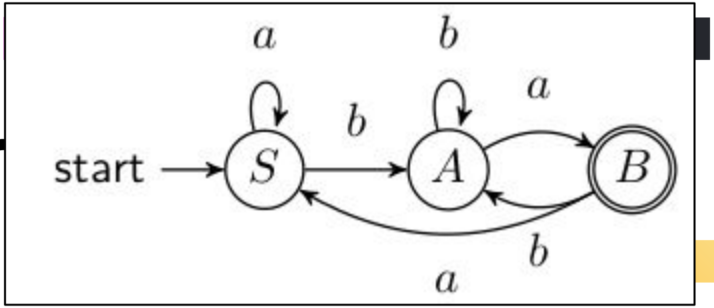
Context-Free Grammar



- **Creating Grammar**
  - Given the following language represented by a DFA :
  - Produce the grammar for it
    - S → a**S** | b**A**
    - **A → bA | aB**

Use
- **states as Variables**
- **Transition as terminal symbols**

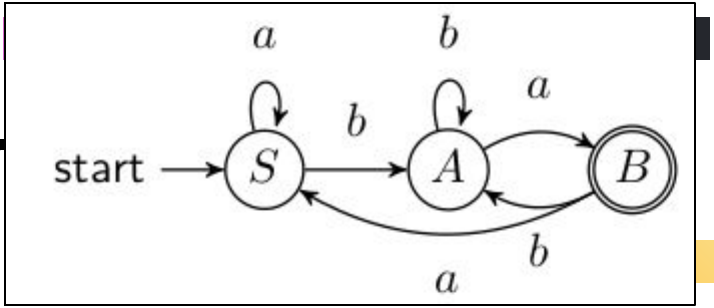
Context-Free Grammar



- **Creating Grammar**
  - Given the following language represented by a DFA :
  - Produce the grammar for it
    - S → a**S** | b**A**
    - **A → bA | aB**
    - **B → bA | aS | ε**

<u>Note :</u> only one terminal in addition to one variable.

Context-Free Grammar



- **Creating Grammar**
  - Given the following language represented by a DFA :
  - Produce the grammar for it
    - S → a**S** | b**A**
    - **A → bA | aB**
    - **B → bA | aS | ε**

<u>Why Epsilon ?</u>

Context-Free Grammar

- **Creating Grammar**
  - Given the following language :

    *All binary strings with an **even** number of **a***
  - Produce the grammar for it

Context-Free Grammar

- **Creating Grammar**
  - Given the following language :

    *All binary strings with an **even** number of **a***
  - Produce the grammar for it
    - $S \rightarrow aXaS \mid \varepsilon$

Context-Free Grammar

- **Creating Grammar**
  - Given the following language represented by a DFA :

    *All binary strings with an **even** number of **a***
  - Produce the grammar for it
    - S → aXaS | ε
    - X → bX | ε

Context-Free Grammar

- **Creating Grammar**
  - Given the following language represented by a DFA :

    *All binary strings with an **even** number of **a***
  - Produce the grammar for it
    - $S \rightarrow aXaS \mid \varepsilon$
    - $X \rightarrow bX \mid \varepsilon$

- **What about the word: baa**

Context-Free Grammar

- **Creating Grammar**
  - Given the following language represented by a DFA :

    *All binary strings with an **even** number of **a***
  - Produce the grammar for it
    - $S \rightarrow aXaS \mid \varepsilon \mid$ **bS**
    - $X \rightarrow bX \mid \varepsilon$

- **What about the word: baa**

Context-Free Grammar

- **Creating Grammar**
  - Given the following language represented by a DFA :

    All binary strings with an ***even*** *number of* ***a***
  - Produce the grammar for it
    - S → aXaS | ε | **bS**
    - X → bX | ε
  - Or:
    - S → bS | aT | ε
    - T → bT | aS

Context-Free Grammar

- **Creating Grammar**
  - Given the following language :

    All binary strings with both an even number of a ***and*** *an even number of b.*
  - Produce the grammar for it
    - $S \rightarrow \varepsilon$?

Context-Free Grammar

- **Creating Grammar**
  - Given the following language :

    All binary strings with both an even number of a ***and*** *an even number of b.*
  - Produce the grammar for it
    - $S \rightarrow \varepsilon \mid aX \mid bY$

Context-Free Grammar

- **Creating Grammar**
  - Given the following language :

    All binary strings with both an even number of a ***and*** *an even number of b.*
  - Produce the grammar for it
    - $S \rightarrow \varepsilon \mid aX \mid bY$
    - $X \rightarrow aS \mid bZ$
    - $Y \rightarrow bS \mid aZ$
    - $Z \rightarrow aY \mid bX$

Context-Free Grammar

- **Regular Grammar :**
  - used to describe regular language.
  - Is a special type of context-free grammar
  - It can be strictly either :
    - right-linear grammar
    - Left-linear grammar.
  - The only form for the grammar that production rules can take (right-linear grammar) (= few restrictions)
    - $A \rightarrow a$
    - $A \rightarrow aB$
    - $A \rightarrow \varepsilon$

Derivation Tree

- **Construction of a Tree**
  - Root
    - is the start variable,
  - all internal nodes:
    - are labeled with variables
    - The children of an internal node are labeled from left to right with the right-hand side of the production rule used.
  - all leaves :
    - are labeled with terminals including the empty string.

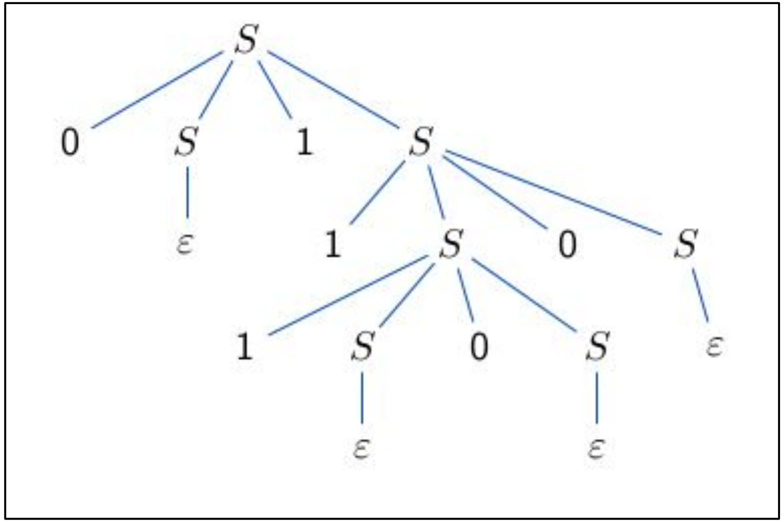
Derivation Tree

- **Construction of a Tree : Example**
  - Grammar : $S \rightarrow 0S1S \mid 1S0S \mid \varepsilon$
  - Derivation for : **011100**
  - Derivation Process :

    $S \Rightarrow 0S1S \Rightarrow 01S \Rightarrow 011S0S$

    $\Rightarrow 0111S0S0S \Rightarrow 01110S0S$

    $\Rightarrow 011100S \Rightarrow 011100$

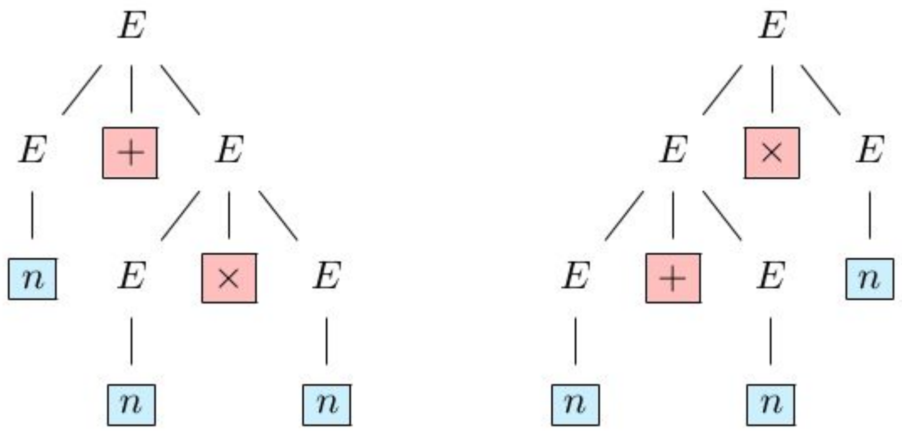


Ambiguous Grammar

- **Definition**
  - A grammar is unambiguous if there is a **unique** leftmost derivation for **each** string in the language.
    - Equivalently, for each string there is a **unique derivation tree.**
  - Inversely : **Ambiguous Grammar** is defined as the grammar where there is a given word (even a single word) that can have more than one derivation tree.

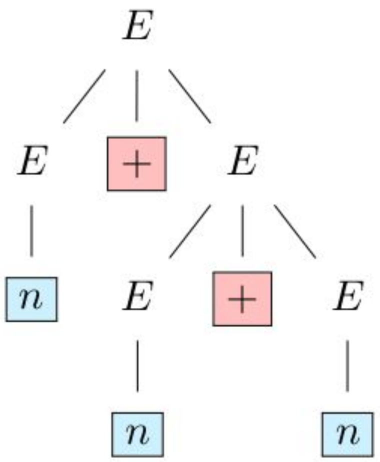
Ambiguous Grammar

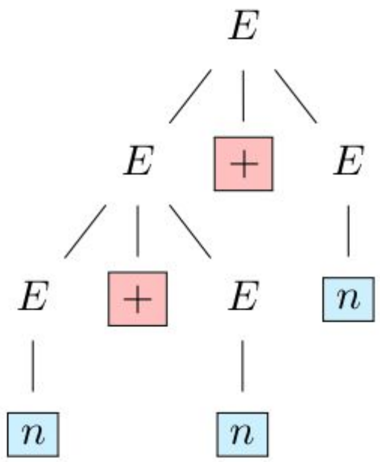
- **Example : E →E+E | E×E | (E) | n**
  - Word : n + n x n
    - LMD 1: E ⇒ E + E ⇒ n + E ⇒ n + E × E ⇒ n + n × E ⇒n+n×n
    - LMD 2: E ⇒ E × E ⇒ E + E × E ⇒ n + E × E ⇒ n + n × E ⇒n+n×n



Ambiguous Grammar

- **Example : E →E+E | E×E | (E) | n**
  - Word : n + n + n
    - LMD 1: E ⇒ E + E ⇒ n + E ⇒ n + E + E ⇒ n + n + E ⇒ n+n+n
    - LMD 2: E ⇒ E + E ⇒ E + E + E ⇒ n + E + E ⇒ n + n + E ⇒n+n+n





Ambiguous Grammar

- **C++ Program** :
  - Is it ambiguous ?
  - What's the output ?

```cpp
#include <iostream>
using namespace std;

int main()
{
    if (true)
        if (false)
            ;
    else
        cout << "Hi!";

    return 0;
}
```

Ambiguous Grammar

- **C++ Program** :
  - Is it ambiguous ?
  - What's the output ?
    - The output is : **Hi !**

```
#include <iostream>
using namespace std;

int main()
{
    if (true)
        if (false)
            ;
    else
        cout << "Hi!";

    return 0;
}
```

Ambiguous Grammar

- **C++ Program** :
  - Is it ambiguous ?
  - What's the output ?
    - The output is : **Hi !**
    - **? else ? belong to which if ?**

```
#include <iostream>
using namespace std;

int main()
{
    if (true)
        if (false)
            ;
    else
        cout << "Hi!";

    return 0;
}
```

Chomsky Normal Form

- **Definition**
  - A context-free grammar is said to be in **Chomsky normal form** (CNF) if every production is of one of these three types:
    - $S \rightarrow BC$
    - $S \rightarrow a$
    - $S \rightarrow \varepsilon$
- **Why :**
  - To avoid the ambiguity problem during parsing

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 1 : Start nonterminal must not appear on RHS
  - Step 2 : Remove ε productions
  - Step 3: Remove unit productions
  - Step 4 : Convert to CNF

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 1 : Start nonterminal must not appear on RHS
    - Example :
      - **S** → ASA | aB
      - A→B | **S**
      - B→b | ε

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 1 : Start nonterminal must not appear on RHS
    - Example :
      - **S** → ASA | aB
      - A→B | **S**
      - B→b | ε
    - It would be :
      - $\mathbf{S_0}$ → **S**
      - S → ASA | aB
      - A →B |S
      - B → b | ε

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 2 : Remove ε productions
    - Example :
      - $S_0 \rightarrow S$
      - $S \rightarrow ASA \mid aB$
      - $A \rightarrow B \mid S$
      - $\mathbf{B \rightarrow b \mid \varepsilon}$

We need to remove :

$\mathbf{B \rightarrow b \mid \varepsilon}$

Chomsky Normal Form

- **Rules to convert CFG to CNF**
    - Step 2 : Remove ε productions
        - Example :
            - $S_0 \rightarrow S$
            - $S \rightarrow ASA \mid a\mathbf{B}$
            - $A \rightarrow \mathbf{B} \mid S$
            - $\mathbf{B \rightarrow b \mid \varepsilon}$

We need to remove :

$\mathbf{B \rightarrow b \mid \varepsilon}$

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 2 : Remove ε productions
    - Example :
      - $S_0 \rightarrow S$
      - S → ASA | a**B**
      - A→ **B** | S
      - **B→b | ε**
    - It would be :
      - $S_0 \rightarrow S$
      - **S → ASA | a B | a**
      - A→ **B** | S | **ε**
      - **B→ b**

We need to remove ε for A which came from B

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 2 : Remove ε productions
    - Example :
      - $S_0 \rightarrow S$
      - $S \rightarrow ASA \mid a\mathbf{B}$
      - $A \rightarrow \mathbf{B} \mid S$
      - $\mathbf{B} \rightarrow \mathbf{b} \mid \boldsymbol{\varepsilon}$
    - It would be :
      - $S_0 \rightarrow S$
      - $S \rightarrow \mathbf{ASA} \mid a\ B \mid \mathbf{a}$
      - $A \rightarrow B \mid S \mid \boldsymbol{\varepsilon}$
      - $B \rightarrow b$

ASA can be when considering ε : AS or SA or ASA

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 2 : Remove ε productions
    - Example :
      - $S_0 \rightarrow S$
      - $S \rightarrow ASA \mid a\mathbf{B}$
      - $A \rightarrow \mathbf{B} \mid S$
      - $\mathbf{B \rightarrow b \mid \varepsilon}$
    - It would be :
      - $S_0 \rightarrow S$
      - $S \rightarrow \mathbf{ASA \mid AS \mid SA \mid S} \mid a\,B \mid \mathbf{a}$
      - $A \rightarrow B \mid S$
      - $B \rightarrow b$

ASA can be when considering ε : AS or SA or ASA

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 3 : Remove Unit Productions in the form of **A**
    - Example :
      - $\mathbf{S_0 \rightarrow S}$
      - $S \rightarrow ASA \mid AS \mid SA \mid S \mid a\,B \mid a$
      - $A \rightarrow B \mid S$
      - $B \rightarrow b$

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 3 : Remove Unit Productions
    - Example :
      - $S_0 \to S$
      - $S \to ASA \mid AS \mid SA \mid S \mid aB \mid a$
      - $\mathbf{A} \to \mathbf{B} \mid S$
      - $B \to b$
    - It would be :
      - $S_0 \to S$
      - $S \to ASA \mid AS \mid SA \mid S \mid aB \mid a$
      - $A \to \mathbf{b} \mid S$
      - $B \to b$

Any other Unit production in the form $A \to Y$

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 3 : Remove Unit Productions
    - Example :
      - $\mathbf{S_0 \rightarrow S}$
      - $S \rightarrow ASA \mid AS \mid SA \mid \mathbf{S} \mid aB \mid a$
      - $A \rightarrow B \mid S$
      - $B \rightarrow b$
    - It would be :
      - $S_0 \rightarrow S$
      - $S \rightarrow ASA \mid AS \mid SA \mid aB \mid a$
      - $A \rightarrow \mathbf{b} \mid S$
      - $B \rightarrow b$

We remove S → S : Anything to do ? do nothing

Chomsky Normal Form

Rules to convert CFG to CNF

- Step 3 : Remove Unit Productions
  - Example :
    - $S_0 \rightarrow S$
    - $S \rightarrow ASA \mid AS \mid SA \mid S \mid a\,B \mid a$
    - $A \rightarrow B \mid \mathbf{S}$
    - $B \rightarrow b$
  - It would be :
    - $S_0 \rightarrow S$
    - $S \rightarrow ASA \mid AS \mid SA \mid a\,B \mid a$
    - $A \rightarrow b \mid \mathbf{ASA \mid AS \mid SA \mid a\,B \mid a}$
    - $B \rightarrow b$

We remove A → S

Chomsky Normal Form

Rules to convert CFG to CNF

- Step 3 : Remove Unit Productions
  - Example :
    - $\mathbf{S_0 \rightarrow S}$
    - $S \rightarrow ASA \mid AS \mid SA \mid S \mid a B \mid a$
    - $A \rightarrow B \mid \mathbf{S}$
    - $B \rightarrow b$
  - It would be :
    - $\mathbf{S_0 \rightarrow ASA \mid SA \mid AS \mid aB \mid a}$
    - $S \rightarrow ASA \mid AS \mid SA \mid a B \mid a$
    - $A \rightarrow b \mid \mathbf{ASA \mid AS \mid SA \mid a B \mid a}$
    - $B \rightarrow b$

We remove $S_0 \rightarrow S$

Chomsky Normal Form

Rules to convert CFG to CNF

- Step 4 : Converting by removing cases of ABC
  - Example :
    - $S_0 \rightarrow$ ASA | SA | AS | aB | a
    - $S \rightarrow$ ASA | AS | SA | S | a B | a
    - $A \rightarrow$ b | ASA | AS | SA | a B | a
    - $B \rightarrow$ b
  - It would be :
    - $\mathbf{S_0 \rightarrow AA_1}$ **| SA | AS | aB | a**
    - $\mathbf{A_1 \rightarrow SA}$
    - $S \rightarrow \mathbf{AA_1}$ | AS | SA | S | a B | a
    - $A \rightarrow$ b | $\mathbf{AA_1}$ **| AS | SA | a B | a**
    - $B \rightarrow$ b

Converting ASA $\rightarrow$ $\mathbf{AA_1}$

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 4 : Converting by removing cases of ABC
    - Example :
      - $\mathbf{S_0 \rightarrow AA_1 \mid SA \mid AS \mid aB \mid a}$
      - $\mathbf{A_1 \rightarrow SA}$
      - $S \rightarrow \mathbf{AA_1} \mid AS \mid SA \mid S \mid a\,B \mid a$
      - $A \rightarrow b \mid \mathbf{AA_1 \mid AS \mid SA \mid a\,B \mid a}$
      - $B \rightarrow b$

$S \rightarrow aB$ is not accepted by CNF

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 4 : Converting by removing cases of ABC
    - Example :
      - $\mathbf{S_0 \rightarrow AA_1 \mid SA \mid AS \mid aB \mid a}$
      - $\mathbf{A_1 \rightarrow SA}$
      - $S \rightarrow \mathbf{AA_1} \mid AS \mid SA \mid S \mid a\,B \mid a$
      - $A \rightarrow b \mid \mathbf{AA_1 \mid AS \mid SA \mid a\,B \mid a}$
      - $B \rightarrow b$

$S \rightarrow aB$ is not accepted by CNF

$S \rightarrow aB$ can be converted to : $S \rightarrow A_2B \quad A_2 \rightarrow a$

Chomsky Normal Form

- **Rules to convert CFG to CNF**
  - Step 4 : Converting by removing cases of ABC
    - Example :
      - $S_0 \rightarrow AA_1 \mid SA \mid AS \mid \mathbf{aB} \mid a$
      - $A_1 \rightarrow SA$
      - $S \rightarrow AA_1 \mid AS \mid SA \mid S \mid \mathbf{a\,B} \mid a$
      - $A \rightarrow b \mid AA_1 \mid AS \mid SA \mid \mathbf{a\,B} \mid a$
      - $B \rightarrow b$
    - It would be :
      - $S_0 \rightarrow AA_1 \mid SA \mid AS \mid \mathbf{A_2}B \mid a$
      - $A_1 \rightarrow SA$
      - $S \rightarrow AA_1 \mid AS \mid SA \mid \mathbf{A_2B} \mid a$
      - $A \rightarrow b \mid AA_1 \mid AS \mid SA \mid \mathbf{A_2B} \mid a$
      - $B \rightarrow b$
      - $\mathbf{A_2 \rightarrow a}$

Questions

- Why it is called “Context-Free”
  -

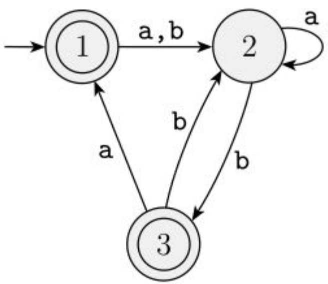
TD 4 RegEx- Solutions

- **Exercise 1**
  - The language of all strings containing exactly two a's.
    $b^*ab^*ab^*$
  - The language of all strings containing at least two a's.
    $\Sigma^*a\Sigma^*a\Sigma^*$
  - The language of all strings that do not end with ab.
    ((a|b)*(a|bb)) | b | e
  - The language of all strings that begin or end with aa or bb.
    (aa|bb)Σ* | Σ*(aa|bb)
  - The language of all strings in which every a is followed immediately by bb.
    (b|abb)* OR b*(abb)*b*
  - The language of all strings containing both bb and aba as substrings.
    Σ*
  - The language of all strings in which the number of a's is even.
    Σ*
  - The language of all strings not containing the substring aa.
    b*(abb*)* (ε|a) OR (ε|a)(a|ba)*) | ((b|ab)*(ε|a))
  - The language of all strings containing no more than one occurrence of the string aa. (The string aaa should be viewed as containing two occurrences of aa.)

TD 4 RegEx- Solutions

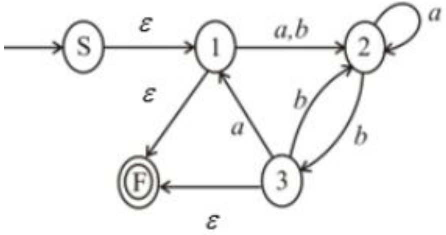
- **Exercise 2**
  - (0 ∪ 1)* 000(0 ∪ 1)*
  - (((00)* (11)) ∪ 01)*
  - ∅*
  - (0 ∪ 1+)0+ 1+

TD 4 RegEx- Solutions

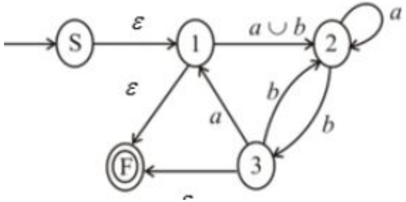


- **Exercise 3**

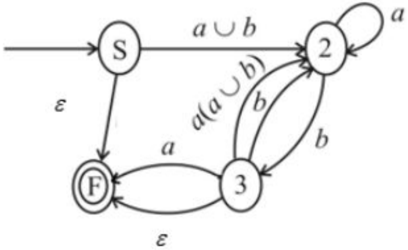
1. Make new Start State S and new Final State F



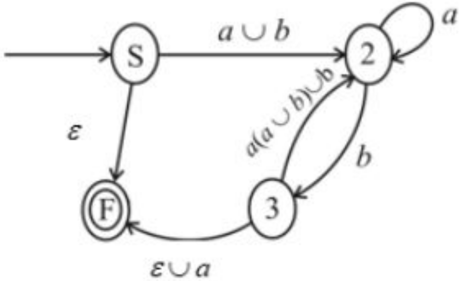
2. From 1 and 2, we can write them as **a | b**



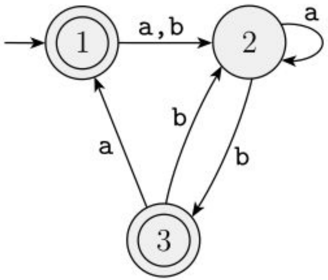
3. We drop the state 1 and compensate for the missing transitions



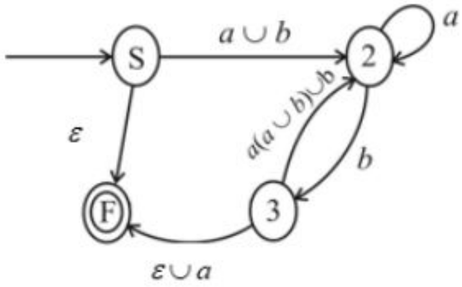
4. Optimize : two outgoing transitions from a state ⇒ I



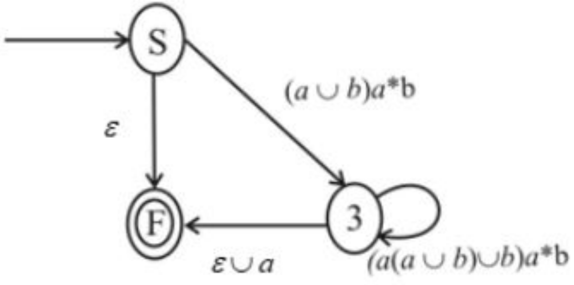
TD 4 RegEx- Solutions



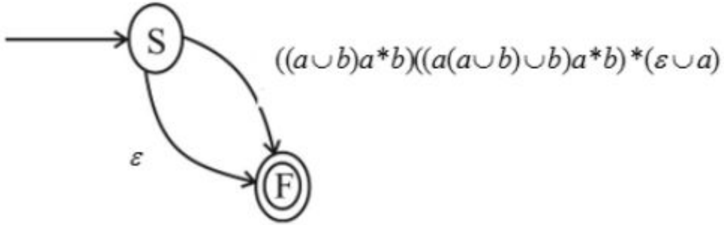
- **Exercise 3**



5. State 2 is dropped



6. We drop state 3



7. Two outgoing transition from a single state ⇒ |