



Biskra University

Computer Science Department

Théorie des langages(TL)

Formal Languages (2025)

By

Pr. CHERIF Foudil

Course Program: Formal Languages

- 1- Introduction to formal logic
- 2- Introduction to languages
- 3- Typology of grammars
- 4- Regular languages (**Type 3**)
 - a- Regular grammars
 - b- Finite state automata
 - c- Regular expressions
- 5- Algebraic languages (free context) (**Type 2**)
 - a- Transformation of grammars (empty word, recursion, ..)
 - b- Chomsky's Grammar
 - c- Greibach's Grammar
 - d- pushdown automata
- 6- **Type 1**: Contextual languages and linear terminal automata
- 7- **Type 0** languages and Turing machines

Bibliography

1. Dexter Kozen. **Automata and Computability** (1997).
2. Michael Sipser. **Introduction to the Theory of Computation**, 3rd ed. (2012).
3. John Hopcroft and Jeffrey Ullman. **Introduction to Automata Theory, Languages, and Computation**, 1st ed. (1979). .
4. Jeffrey Shallit. **A Second Course in Formal Languages and Automata Theory** (2008).
5. P. Wolper. **Introduction à la calculabilité**. 2006, Dunod.
6. P. Séébold. **Théorie des automates**. 2009, Vuibert.
7. J.M. Autebert **Théorie des langages et des automates**. 1994, Masson.
8. J. Hopcroft, J. Ullman. **Introduction to Automata Theory, Languages and Compilation** 1979, Addison- Wesley

Motivation of the course

The objective of this course is to introduce the **theory of formal languages**.

Languages allow humans to exchange information and ideas and to communicate with machines.

The languages used between humans are called '**natural languages**', they are usually **informal** and **ambiguous** and require interpretation by a human brain to be interpreted correctly.

The languages created by humans to communicate with the machine are the **formal languages** or **artificial languages**. They must be formalized and unambiguous in order to be interpreted by a machine, this is the goal of this course.

Chapter 1:

Introduction to formal logic

Chapter 1: Introduction to formal logic

1. Definition of formal systems

A formal system is a set of data which makes it possible to manipulate a set of symbols by considering only their syntax (structure) without taking into account their semantics (meaning, interpretation).

A formal system consists of a **syntax**:

1. A finite alphabet of symbols
2. A formula construction process for describing well-formed formulas of this system (language).

Example: An alphabet $V = \{ a, b, c \}$

The well-formed formulas: sequence of letters of V containing the letter **a only one time**, and the letter **c only one time** and **b is before c**

Chapter 1: Introduction to formal logic

2- Introduction to the theory of languages (formal languages):

The language theory defines programming languages, but compilation transforms programs written in these languages into machine code.

The source program is transformed into:

- 1.absolute machine language (directly executable)
- 2.translatable machine language (requires linking)
- 3.assembly language (requires assembler)
- 4.high-level language (requires a compiler)

The basic structure for the theory of languages is the **monoid** (is a structure equipped with an operation)

Chapter 1: Introduction to formal logic

A language is defined on a set called vocabulary (characters or symbols) is a subset of finite strings of characters.

A **language** is defined by a **grammar**.

Automata are symbolic machines validating the membership of a given string in the language it describes (all these notions will be studied in the following chapters.

1. finite state automata (type 3)
2. Pushdown automata (type 2)
3. linear terminal automata (type 1)
4. Turing's machines (type 0)

Chapter 1: Introduction to formal logic

3- Monoid structure:

A **monoid** is a structure with the composition law is **associative**.

Associativity: The operation must be associative, meaning that for any elements a , b , and c in the set, $(a * b) * c = a * (b * c)$.

We call **free monoid** any monoid having an **identity element**.

Identity element: There must exist an element in the set, called the identity element, such that for any element a in the set, $a * \text{identity} = \text{identity} * a = a$.

The example that interests us in our course is the set of finite character strings on a finite vocabulary, this set is provided with the operation of **concatenation** which is associative and which has an identity element, **the empty string**

Chapter 1: Introduction to formal logic

3.1 Vocabulary

A vocabulary V or alphabet is a finite set of letters or symbols called letters (letters, numbers or other symbols)

Examples:

1) $V = \{ a_1, a_2, \dots, a_n \}$ V : the alphabet a_i : the letters

2) $V = \{ 1 \}$ one-letter alphabet

3) $V = \{ 0, 1 \}$ binary alphabet

4) $V = \{ ., -, / \}$ Morse code for transmission

5) $V = \{ 0, 1, \dots, 9, a, b, \dots, z \}$ any alphabet

Chapter 1: Introduction to formal logic

3.2 Monoide V^+

We call **monoid V^+** the set of all the strings of **non-empty finite lengths** defined on V . These strings are called words and the set V^+ is infinite.

In other words V^+ is the set of words of length greater than or equal to 1 that can be constructed from the alphabet V

Example:

$$V = \{ a, b \}$$

$$V^+ = \{ a, b, aa, bb, ab, ba, bb, aaa, \dots \}$$

Chapter 1: Introduction to formal logic

3.3 Concatenation operation

The concatenation operation consists in juxtaposing two words in order to obtain a new word. It is **associative** but **not commutative** operation.

$x, y \in V^+$ x and y are two words

$$x = x_1.x_2 \dots x_k \quad / \quad x_i \in V$$

$$y = y_1.y_2 \dots y_p \quad / \quad y_i \in V$$

$$x.y = x_1.x_2 \dots x_k y_1.y_2 \dots y_p$$

$(x.y).z = x.(y.z)$. Is associative

$x.y \neq y.x$. is not commutative

Chapter 1: Introduction to formal logic

3.4 Free Monoïde V^*

The concatenation operation admits an identity element which is the empty string (length equal to zero) and denoted by ε , $x.\varepsilon = \varepsilon.x = x$

We can define $V^* = V^+ \cup \{\varepsilon\}$

3.5 Word length

The length of a word x which is generally noted $|x|$ matches each word with the number of symbols it contains.

We define a particular word called empty word, this word is not composed of any character, its length is therefore zero ($|\varepsilon| = 0$).

Chapter 1: Introduction to formal logic

3.5 Subword

We say that $y \in V^*$ is a subword (or factor) of $x \in V^*$ if there exist finite words $u, v \in V^*$ such that $x = u y v$ and $|y| \leq |x|$

If $x = \mathbf{y} v$ we say that y is left factor or **prefix** of x

If $x = v \mathbf{y}$ we say that y is right factor or **suffix** of x

Example

$V = \{ a, b \}$

$x = ab\mathbf{bbb}aa$ and $y = bbb$ so $x = ab\mathbf{y}aa$ subword

$x = bbb\mathbf{aa}$ and $y = bbb$ so $x = \mathbf{y}aa$ left factor

Chapter 2: Introduction to languages

Chapter 2: Introduction to languages

1. Définition

A language on a vocabulary V is a subset of the words defined over V , in other words a language is a part of the free monoid V^* .

$$L \subset V^*$$

We can differentiate between the empty language ($L = \emptyset$) and the language containing the only empty word ($L = \{\varepsilon\}$)

Example :

$$V = \{ a, b \}$$

$$V^* = \{ \varepsilon, a, b, aa, bb, ab, ba, bb, aaa, \dots \}$$

$$L = \{ aa, bb, ab, ba \} \quad \text{the set of words on } V^* \text{ of length equal to 2}$$

Chapter 2: Introduction to languages

2. Syntax of a language

A sentence is well-formed if and only if it belongs to the language.

The syntax of a language is the set of constraints(rules) which make it possible to define the sentences of this language.

Example: A simple measurement language can be defined by the following constraints:

- **measure** followed by an **operator** and a **measure**
- a measure is the number **1** followed by a **unit**
- Units are : **cm , liter, km**
- The opérateur is **+**

Chapter 2: Introduction to languages

2. Syntax of a language

- **measure followed by an operator and a measure**
- **a measure is the number 1 followed by a unit**
- **Units are : cm , litre, km**
- **The operator is +**

The well-formed sentences are:

1cm + 1cm

1cm + 1liter

1cm + 1km

1liter + 1cm

1liter + 1liter

1liter + 1km

1km + 1cm

1km + 1liter

1km + 1km

Chapter 2: Introduction to languages

3. Sémantic of a language

The semantics of a language is a set of constraints on this language.

Among the well-formed sentences in the measurement example, only a few are semantically correct, those whose units are of the same type (measure or weight).

These sentences are:

1cm + 1cm

1cm + 1km

1liter + 1liter

1km + 1cm

1km + 1km

Chapter 2: Introduction to languages

4. Opérations on languages

As we have seen, languages are sets of words. The usual operations concerning sets such as **union**, **intersection** and **complementation** are applicable to languages.

Consider two languages L_1 and L_2 respectively defined on the two alphabets V_1 and V_2 .

a) **Union**

The union of L_1 and L_2 is the language defined on $V_1 \cup V_2$ containing all the words which are either contained on L_1 or contained on L_2 .

$$L_1 \cup L_2 = \{ x / x \in L_1 \text{ or } x \in L_2 \}$$

b) **Intersection**

The intersection of L_1 and L_2 is the language defined on $V_1 \cap V_2$ containing all the words which contained on L_1 and on L_2 .

$$L_1 \cap L_2 = \{ x / x \in L_1 \text{ and } x \in L_2 \}$$

Chapter 2: Introduction to languages

4. Opérations on languages

c) Complementation

The complement of L_1 is the language defined on V_1 containing all the words which are not in L_1 .

$$C(L_1) = \{ x / x \notin L_1 \}$$

d) Difference

The difference of L_1 and L_2 is the language defined on L_1 containing all the words of L_1 which are not in L_2 .

$$L_1 - L_2 = \{ x / x \in L_1 \text{ and } x \notin L_2 \}$$

Chapter 2: Introduction to languages

4. Opérations on languages

e) Concatenation

The concatenation of L_1 and L_2 is the language defined on $V_1 \cup V_2$ containing all the words made up of a word from L_1 followed by a word from L_2 .

$$L_1 L_2 = \{ xy \mid x \in L_1 \text{ and } y \in L_2 \}$$

The concatenation operation is not commutative $L_1 L_2 \neq L_2 L_1$

f) Power

The power of a language is defined by:

$$L^0 = \{\varepsilon\} \qquad L^{n+1} = L^n L$$

Chapter 2: Introduction to languages

4. Opérations on languages

g) Kleen closure

The **iterative closure** of L or **Kleen closure** (**iterate of L**, or **Kleen star**) is the set of words formed by a finite concatenation of words of L.

$$L^* = \{ x \mid \exists k \geq 0 \text{ and } x_1, x_2, \dots, x_k \in L \text{ such as } x = x_1 x_2 \dots x_k \}$$

$$L^* = \{ \varepsilon \} \cup L \cup L^2 \cup L^3 \dots \cup L^n \cup \dots$$

We can similarly define the **positive Kleene closure** of L by:

$$L^+ = \bigcup_{i \geq 1} L^i = L \cup L^2 \cup L^3 \dots \cup L^n \cup \dots$$

Chapter 2: Introduction to languages

4. Opérations on languages

h) Reversal (R or \sim)

The reversal of a string $w = x_1x_2\dots x_k$ is the string with the symbols written in reverse order, $w^R = x_kx_{k-1}\dots x_2x_1$

Formally,

$$L^R = \{ x^R / x \in L \}$$

a) if $w = \varepsilon$, then $\varepsilon^R = \varepsilon$ and

b) if $w = ax$ for a symbol $a \in V$ and a string $x \in V^*$, then $(ax)^R = x^Ra$

If $w = w^R$, we say w is a **palindrome**

Chapter 2: Introduction to languages

4. Opérations on languages

i) Remarks:

$$L^+ = L L^*$$

$$L^* L^* = L^*$$

$$(L^R)^R = L$$

$$(L^*)^R = L^*$$

$$(L_1 L_2)^R = L_2^R L_1^R$$

Chapter 2: Introduction to languages

Examples :

Let L_1 , L_2 and L_3 be three languages defined by:

$$L_1 = \{\varepsilon, aa\}, \quad L_2 = \{a^i b^j \mid i, j \geq 0\} \quad \text{and} \quad L_3 = \{ab, b\}.$$

Calcule :

$$L_1.L_2, \quad L_1.L_3, \quad L_1 \cup L_2, \quad L_2 \cap L_3, \quad L_1^{10}, \quad L_1^*, \quad L_2^R$$

Solutions :

- $L_1.L_2 = L_2$;
- $L_1.L_3 = \{ab, b, aaab, aab\}$;
- $L_1 \cup L_2 = L_2$;
- $L_2 \cap L_3 = L_3$;
- $L_1^{10} = \{a^{2n} \mid 10 \geq n \geq 0\}$;
- $L_2^R = \{b^i a^j \mid i, j \geq 0\}$.

Find:

$$L_1 = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$$

Chapter 3:

Typology of grammars

Chapter 3: Typology of grammars

1- Definition of a grammar

With a grammar, we describe in a generic and productive way the well-formed expressions of a language.

A grammar is a formal system defined by an **axiom** and **rules** called **production rules**.

The sentences are **derived** from the axiom and by successive application of the rules.

The rules of the grammar are constructed with effective symbols called **terminal symbols** and tool symbols called **non-terminal symbols**, which denote pieces of correct strings during language construction.

Axiom , rules, terminal symbols, non-terminal-symbols

Chapter 3: Typology of grammars

Example1: Consider the following sentence: $- 19.5 \ 10^{-3}$

L: set of numbers of this form (decimal numbers)

ND: A decimal number

We can form a grammar that generates the set L of decimal numbers as follows:

$ND \rightarrow S E P E F$

$E \rightarrow C E$

$E \rightarrow C$

$C \rightarrow 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9$

$P \rightarrow .$

$F \rightarrow 10 S E$

$S \rightarrow + / -$

Chapter 3: Typology of grammars

Example 2: English grammar :

sentence \rightarrow <subject> <verb-phrase> <object>

subject \rightarrow This / Computers / I

verb-phrase \rightarrow <adverb> <verb> / <verb>

adverb \rightarrow never

verb \rightarrow is / run / am / eat / tell

object \rightarrow the <noun> / a <noun> / <noun>

noun \rightarrow university / world / cheese / mouse / lies

Using these rules, we can derive simple sentences like:

This is the university

Computers run the world

the cheese eat the mouse

I never tell lies.

Chapter 3: Typology of grammars

Example 2: English grammar :

Derivation of the first sentence :

$\langle \text{sentence} \rangle \rightarrow \langle \text{subject} \rangle \langle \text{verb-pharse} \rangle \langle \text{object} \rangle$

$\rightarrow \text{This} \langle \text{verb-pharse} \rangle \langle \text{object} \rangle$

$\rightarrow \text{This} \langle \text{verb} \rangle \langle \text{object} \rangle$

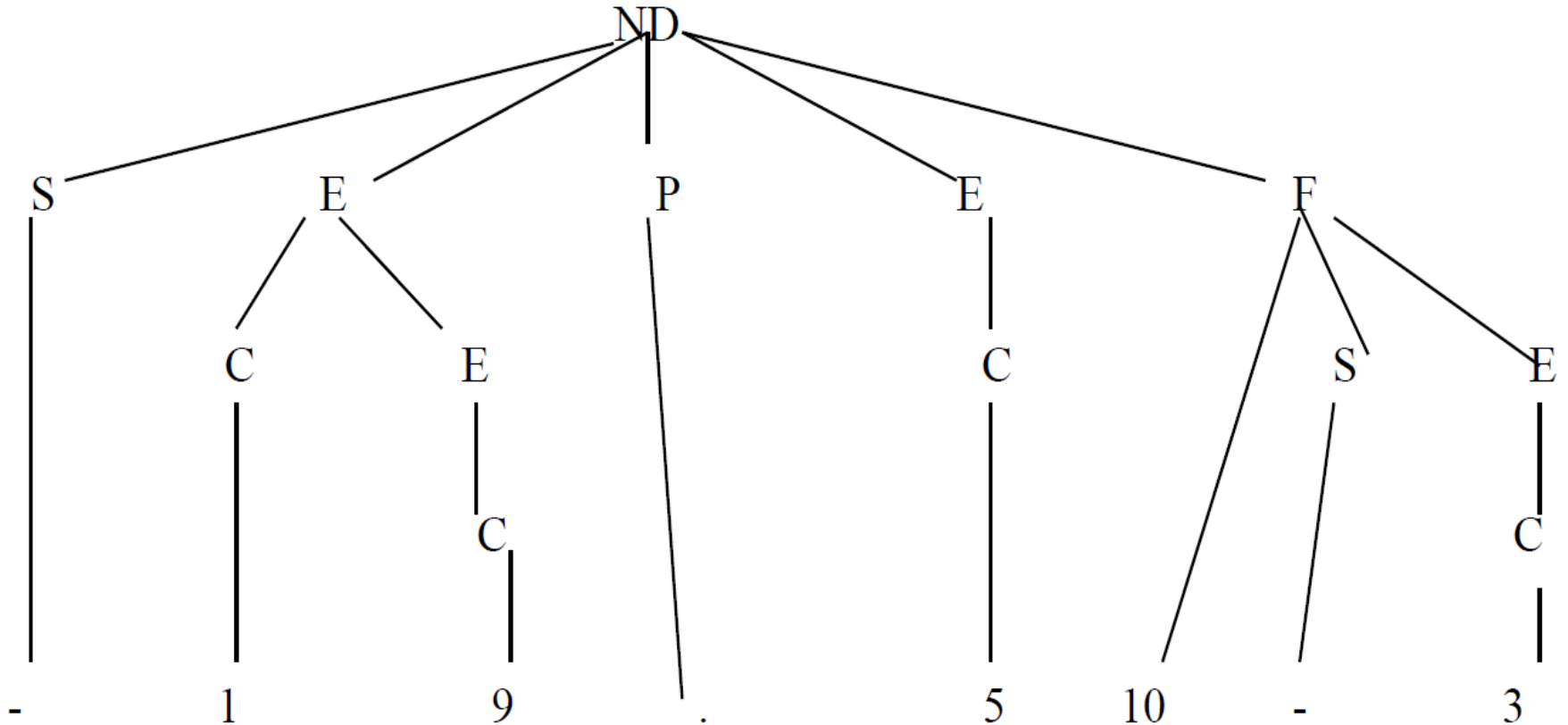
$\rightarrow \text{This is} \langle \text{object} \rangle$

$\rightarrow \text{This is the} \langle \text{noun} \rangle$

$\rightarrow \text{This is the university}$

Chapter 3: Typology of grammars

A decimal number is defined by a **derivation tree**



Chapter 3: Typology of grammars

Concepts to be defined:

1. *Decimal number*: ND is the **root** of the **derivation tree**
2. *Syntactic elements*: (ND, S, E, P, F, C) **Non-terminal vocabulary**
3. *Initial alphabet*: (0,1,2,3,4,5,6,7,8,9,+,-,.,10) **Terminal vocabulary**
4. *Set of rules*: It is the **articulation** of the different elements between them.

Chapter 3: Typology of grammars

2- Formal definition of a grammar

A grammar is a quadruple $G = (V_t, V_n, S, R)$ where

- V_t : is the terminal vocabulary. Is a non-empty finite set.
- V_n : is the non-terminal vocabulary, the set of symbols which do not appear in the generated words, but which are used during the generation. Is a non-empty finite set.
- S : is an element of V_n , is the **starting symbol** or **axiom**. It is from this symbol that we will begin the generation of words using the rules of the grammar.

Chapter 3: Typology of grammars

2- Formal definition of a grammar

A grammar is a quadruple $G = (V_t, V_n, S, R)$ where

- R : is a set of so-called rewriting or production rules of the form:
 - $u \rightarrow v$ such as $u \in (V_t \cup V_n)^+$ and $v \in (V_t \cup V_n)^*$
 - and $V_t \cap V_n = \emptyset$

Terminology :

- A sequence of terminal and non-terminal symbols (an element of $(V_t \cup V_n)^*$ is called a **form**.
- A rule $u \rightarrow v$ such that $v \in V_t^*$ is called a **terminal rule**.

Chapter 3: Typology of grammars

3- Grammar derivation

- **Direct derivation :** " \Rightarrow "

Let a rule of R $u \rightarrow v$ and let x, y two words of $(V_t \cup V_n)^*$

we say that y derives directly from x in G ($x \Rightarrow y$) if and only if

$$x = \alpha u \beta \quad \text{and} \quad y = \alpha v \beta \quad \alpha, \beta \in (V_t \cup V_n)^*$$

- **Indirect derivation:** " \Rightarrow^* "

We say that y derives indirectly from x in G ($x \Rightarrow^* y$) if and only if

it exists a finite sequence W_0, W_1, \dots, W_n such as $W_0 = x$ $W_n = y$

and $W_i \Rightarrow W_{i+1} \quad 0 \leq i \leq n$

$$x \Rightarrow W_1 \dots W_{n-1} \Rightarrow y$$

Chapter 3: Typology of grammars

- example:

4- Language generated by a grammar

The language defined, or generated, by a grammar is the set of words that can be obtained from the starting symbol(axiom) by applying the rules of the grammar.

More formally is the set of terminal derivations of the axiom.

$$G = (V_t, V_n, S, R) \quad L(G) = \{ x / x \in V_t^* \quad \text{and} \quad S \xRightarrow{*} x \}$$

Chapter 3: Typology of grammars

Important remark:

A grammar defines a single language. On the other hand, a language can be generated by several different grammars.

These two **grammars are equivalent**.

We say that G_1 and G_2 are equivalent if and only if $L(G_1) = L(G_2)$

Chapter 3: Typology of grammars

Important remark: equivalent grammars

Let the grammar $G1 = (\{a,b\}, \{S,X\} , S , \{S \rightarrow aXa , X \rightarrow bX \mid \epsilon \})$

➔ $L(G1) = \{ ab^*a \}$. Unique language

But the language $L = \{ ab^*a \}$ can be generated by these 3 different grammars:

$G1 = (\{a,b\}, \{S,X\} , S , \{S \rightarrow aXa , X \rightarrow bX \mid \epsilon \}) ;$

$G2 = (\{a,b\}, \{S,X,Y\} , S , \{S \rightarrow aY , Y \rightarrow Xa , X \rightarrow bX \mid \epsilon \}) ;$

$G3 = (\{a,b\}, \{S,X\} , S , \{S \rightarrow aX , X \rightarrow bX \mid a \}) ;$

$L(G1)=L(G2)=L(G3) \rightarrow G1 , G2, G3$ are equivalent,

Chapter 3: Typology of grammars

Examples: languages construction

Find the languages generated by these grammars:

1. $G_1 = (\{a, b\}, \{S\}, S, R)$

$$R = (S \rightarrow a S b, \quad S \rightarrow ab)$$

2. $G_2 = (\{ _ / , _ \backslash \}, \{S, A, U, V\}, S, R)$

$$R = (\begin{array}{lll} S \rightarrow U A V & S \rightarrow U V & A \rightarrow V S U \\ A \rightarrow V U & U \rightarrow _ / & V \rightarrow _ \backslash \end{array})$$

Chapter 3: Typology of grammars

3. $G_3 = (\{a, b\}, \{S, A, B\}, S, R)$

$R = (S \rightarrow AS \quad S \rightarrow Ab \quad A \rightarrow AB \quad B \rightarrow aA)$

4. $G_4 = (\{a\}, \{S\}, S, R)$

$R = (S \rightarrow AS A \quad S \rightarrow \epsilon \quad A \rightarrow S A \quad A \rightarrow A S a)$

Examples of languages :

- $L_1 = \{ab, a, ba, bb\}$;
- $L_2 = \{\omega \in \{a, b\}^* / |\omega| > 3\}$;
- $L_3 = \{\omega \in \{a, b\}^* / |\omega| \equiv 0 [5] \}$;

Chapter 3: Typology of grammars

$$L_1 = \{ab, a, ba, bb\}$$

$$G = (\{a, b\}, \{S, A, B\}, S, R)$$

$$R = (S \rightarrow aA \quad S \rightarrow bB \quad A \rightarrow b \quad A \rightarrow \epsilon \quad B \rightarrow a \quad B \rightarrow b)$$

$$L_2 = \{\omega \in \{a, b\}^* / |\omega| > 3\}$$

$$G_4 = (\{a, b\}, \{S, A, B\}, S, R)$$

$$R = (S \rightarrow AAAAB \quad A \rightarrow a / b \quad B \rightarrow AB \quad B \rightarrow \epsilon)$$

$$L_3 = \{ \omega \in \{a, b\}^* / |\omega| \equiv 0 \pmod{5} \} \text{ or } L_3 = \{ \omega \in \{a, b\}^* / |\omega| \equiv 0 \pmod{5} \}$$

$$R = (S \rightarrow AAAAAS \quad A \rightarrow a / b \quad S \rightarrow \epsilon)$$

Chapter 3: Typology of grammars

5- Types of grammars

By introducing more or less restrictive criteria on the production rules, we obtain hierarchical classes of grammars, ordered by inclusion. The classification of grammars, defined in 1957 by **Noam CHOMSKY**, distinguishes the following four classes:

5.1- Grammar type 0

Grammars without restriction on rules,
so all grammars are type 0.

$$u \rightarrow v \quad u \in (V_n \cup V_t)^+ \quad \text{and} \quad v \in (V_n \cup V_t)^*$$



Chomsky in 2017

Born in 1928 (age 97)

Philadelphia, Pennsylvania, U.S.

Chapter 3: Typology of grammars

5.2 Grammar type 1:

a) context sensitive grammars

Type 1 grammars are also called **context sensitive** or **context sensitive grammars**.

The grammar rules are of the form:

$$u A v \rightarrow u W v$$

$$A \in V_n, \quad W \in (V_n \cup V_t)^+ \quad \text{and} \quad u, v \in (V_n \cup V_t)^*$$

In other words, the **non-terminal symbol** **A** is replaced by the **form** **W** but if we have the **contexts** **u on the left and v on the right**.

We restrict the rules by forcing the right side to be at least as long as the left side.

$$|u A v| \leq |u W v|$$

Chapter 3: Typology of grammars

5.2 Grammar type 1:

a) context sensitive grammars

This forces the **empty word** to be excluded from the grammar.

But we accept the rule $S \rightarrow \epsilon$ with this condition : « the non terminal S does not exist in the right of each rule of the grammar

b) Monotone grammar (not-decreasing grammar)

$$\alpha \rightarrow \beta \quad \text{where} \quad |\alpha| \leq |\beta|$$

Remark: *All context sensitive grammars are monotones but monotone grammars are not necessary context sensitive grammars*

Chapter 3: Typology of grammars

5.2 Grammar type 1:

Grammar 1: Monotone and not context sensitive

$S \rightarrow aSBc$

$S \rightarrow abc$

$cB \rightarrow Bc$ this rule not context sensitive

$bB \rightarrow bb$

Grammar 2: Monotone and context sensitive

$S \rightarrow aSBC$ $S \rightarrow aBC$ $CB \rightarrow HB$ $HB \rightarrow HC$ $HC \rightarrow BC$

$aB \rightarrow ab$ $bB \rightarrow bb$ $bC \rightarrow bc$ $cC \rightarrow cc$

Grammar 1 and grammar 2 are equivalent have the same language

Chapter 3: Typology of grammars

5.3- Grammar type 2:

Type 2 grammars are also called **context-free**, **algebraic** or **Chomsky grammars**. It is the most widely used grammar in language theory and compilation.

The grammar rules are of the form:

$$A \rightarrow W$$

$$A \in V_n, \quad W \in (V_n \cup V_t)^*$$

In other words, the left member consists of a single non-terminal symbol.

Chapter 3: Typology of grammars

5.4- Grammar type 3:

Type 3 grammars are also called **regular grammars** on the right (respectively on the left), **linear grammars**.

The grammar rules are of **one** of these 02 forms: $A, B \in V_n$ and $a \in V_t$

Form1: $A \rightarrow a B$

Form2: $A \rightarrow B a$

or $A \rightarrow a$

or $A \rightarrow a$

or $A \rightarrow \epsilon$

or $A \rightarrow \epsilon$

The left member of each rule consists of a single non terminal symbol, and the right member consists of a terminal symbol possibly followed (respectively preceded) by a single non terminal.

Chapter 3: Typology of grammars

Examples:

$G = (\{a, b, c\}, \{S, A, B, C, D\}, S, R)$

$R_1 = (S \rightarrow ACaB \quad AB \rightarrow AbBc \quad A \rightarrow bcA \quad B \rightarrow b)$

$R_2 = (S \rightarrow ACaB \quad Bc \rightarrow acB \quad CB \rightarrow DB \quad aD \rightarrow Db)$

$R_3 = (S \rightarrow aAB \quad B \rightarrow aAB \quad aA \rightarrow aaA \quad bbAa \rightarrow bbBa \quad A \rightarrow bcA \quad B \rightarrow \epsilon)$

Chapter 3: Typology of grammars

6- Language type:

Each type of grammar is associated with a type of language:

Type 3 grammars generate regular languages,

type 2 grammars generate context-free languages

type 1 grammars generate contextual languages

and type 0 grammars generate all "**decidable**" languages, in other words, all languages that can be recognized in **finite time** by a machine.

Languages that cannot be generated by a type 0 grammar are called "**undecidable**".

Chapter 3: Typology of grammars

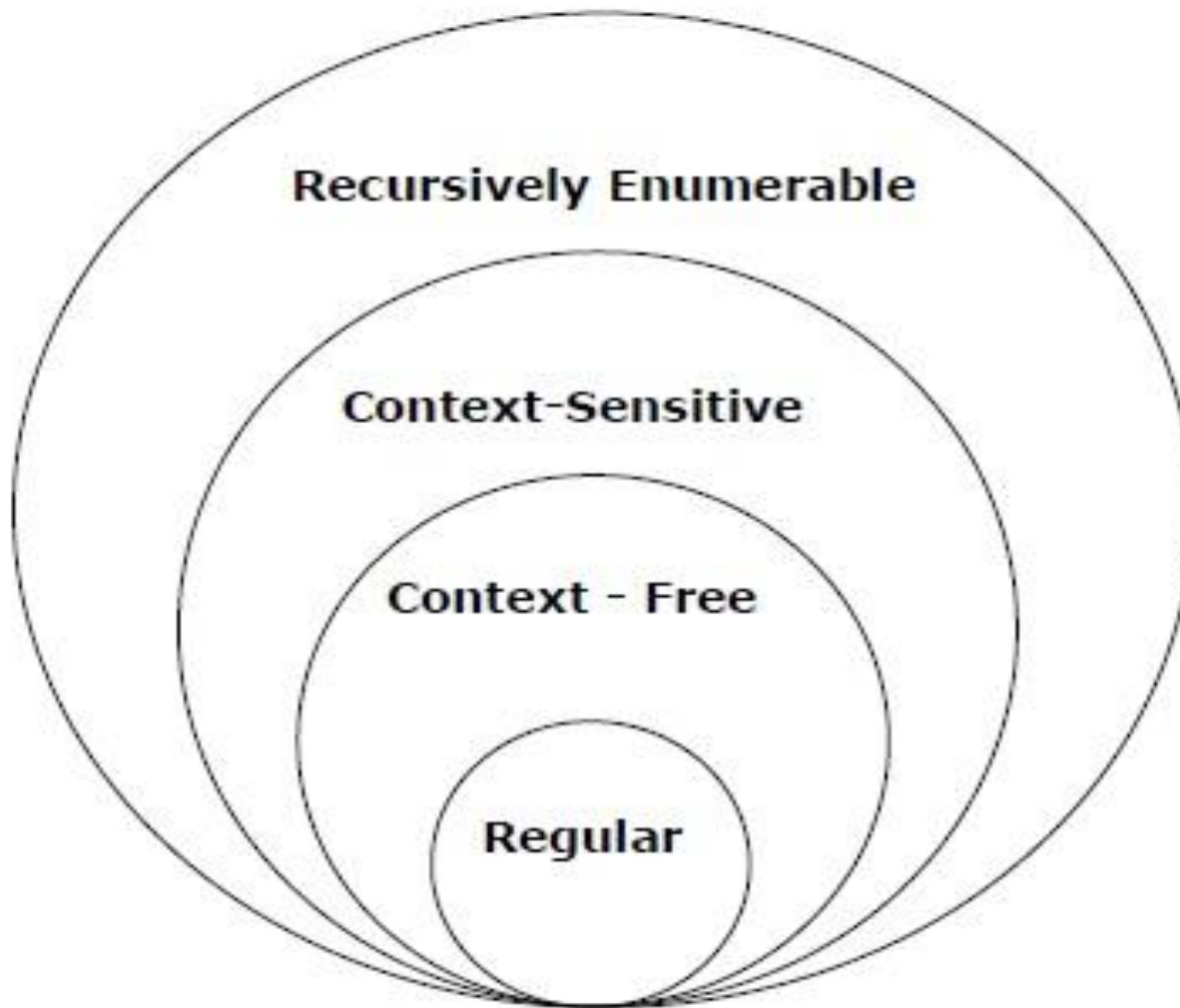
Language type:

These languages are ordered by inclusion: the set of languages generated by type n grammars is strictly included in that of type $n-1$ grammars (for $n = 1, 2, 3$).

Examples :

- a type 3 grammar is also type 2, 1, 0
- a type 2 grammar is also type 1, 0 but not type 3
- a type 1 grammar is also type 0

Chapter 3: Typology of grammars



Chapter 3: Typology of grammars(conclusion)

Each type of grammar is associated with a type of automata which makes it possible to recognize the languages of its class:

- regular languages are recognized by **finite automata**
- context-free languages are recognized by **push down automata**
- Contextual languages are recognized by **linear bounded machines**
- and type 0 languages are recognized by **Turing machines**

The Turing machine is considered the most powerful model, where any language that cannot be processed by one Turing machine, cannot be processed by another machine.

Chapter 4:

Regular languages

Chapter 4: Regular languages

4.1 Regular grammar definition:

Type 3 grammars are also called **regular grammars** on the right (respectively on the left), **linear grammars**.

The grammar rules are of the form:

$$A \rightarrow a B \quad (\text{respectivement } A \rightarrow B a)$$

or
$$A \rightarrow a$$

or
$$A \rightarrow \epsilon$$

$$A, B \in V_n \quad \text{et} \quad a \in V_t$$

Chapter 4: Regular languages

Regular languages are languages generated by regular grammars.

Regular grammars are used in the lexical analysis phase of compilation.

Example:

$G = (\{a, b\}, \{S, A\}, S, R)$

$R = ($ $S \rightarrow a S$

$S \rightarrow b A$

$A \rightarrow a$ $)$

Chapter 4: Regular languages

4.2 Automata definition (**automata plural or automaton singular**)

In formal language, we have two systems:

- The generation systems which are the **grammars**
- The recognition systems which are the **automata**

Automata is a virtual machines (**programs**), which takes as input a string of symbols and performs a string recognition algorithm.

If the algorithm terminates under certain conditions, the automata accepts this string.

The language recognized by an automata is the set of strings it accepts.

Fields of application: compilation and real-time applications

Chapter 4: Regular languages

4.2 Automata definition (Fields of application)

Finite automata has several applications in many areas such as:

compiler design, special purpose hardware design, real-time applications
,protocol specification,...

a) **Compiler Design**

Lexical analysis is an important phase of a compiler. A program such as a C program is scanned and the different tokens (constructs such as variables, keywords, numbers) in the program are identified.

Chapter 4: Regular languages

4.2 Automata definition (Fields of application)

b) Vending Machines:

A vending machine is an automated machine that dispenses numerous items such as cold drinks, snacks, etc. to sale automatically, after a buyer inserts currency or credit into the machine.

Vending machine works on finite state automata to control the functions process.

c)Text Parsing:

Text parsing is a technique which is used to derive a text string using the production rules of a grammar to check the acceptability of a string

Chapter 4: Regular languages

4.2 Automata definition (Fields of application)

d)Traffic Lights:

The optimization of traffic light controllers in a city is a representation of handling the instructions of traffic rules. Its process depends on a set of instruction works in a loop with switching among instruction to control traffic,

e)Video Games:

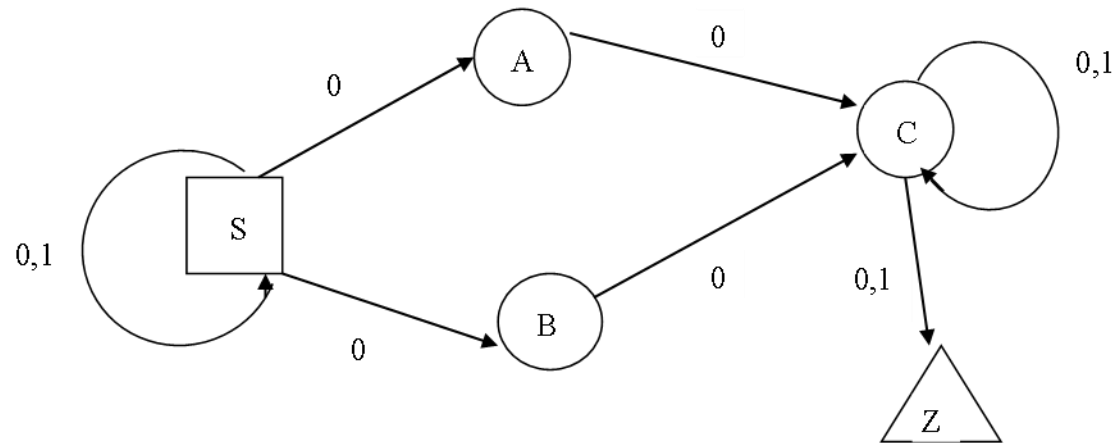
Video games levels represent the states of automata. In which a sequence of instructions are followed by the players to accomplish the task

Chapter 4: Regular languages

4.3 Finite State Automata (FSA)

4.3.1 Definition

A FSA is composed of a finite set of states (represented graphically by **circles**), a transition function describing the action that allows to pass from one state to another (**these are the arrows**), an initial state (the state denoted by **square**) and one or more final states (denoted by **triangles**).



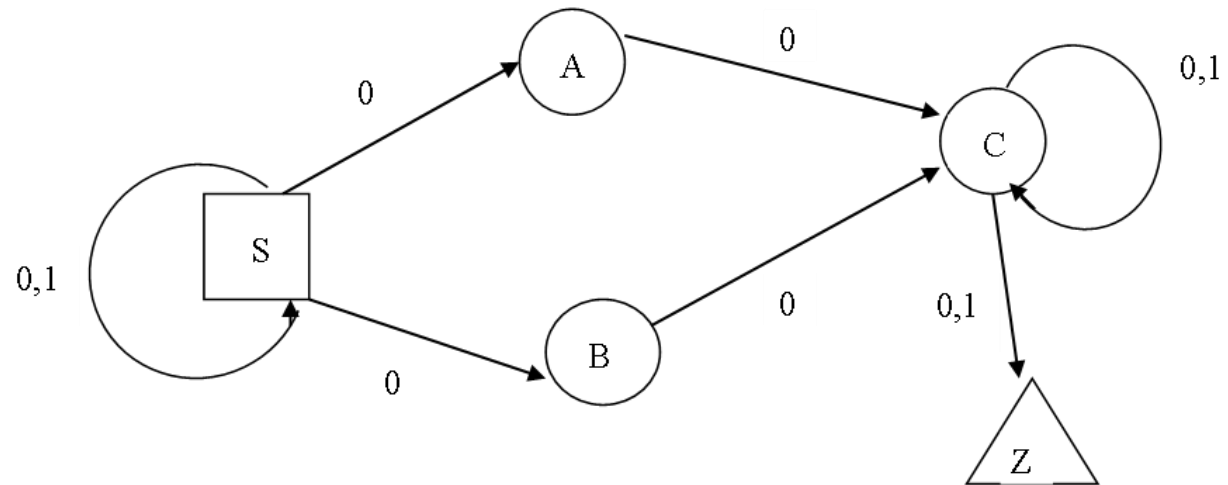
Chapter 4: Regular languages

4.3 Finite State Automata (FSA)

4.3.1 Definition

A FSA is therefore a directed and valued graph where the nodes correspond to the states and the values of the arcs to the terminal symbols,

FSA does not use any memory to recognize a string.



Chapter 4: Regular languages

4.3.2 Formal definition

A finite state automata is a quintuplet:

$$A = (V_t, Q, q_0, f, F)$$

- V_t : is the terminal vocabulary, non-empty finite set of symbols
- Q : is the state set of the automata, non-empty finite set
- q_0 : The set Q contains a particular state q_0 called initial state. $q_0 \in Q$
- F : The set Q contains a subset of particular states F called final states. $F \subset Q$
- f : is an application of $Q \times V_t \cup \{ \epsilon \} \rightarrow Q$

The automata stops on a final state or the complete reading of the input string.

Chapter 4: Regular languages

Representations of a FSA

There are three main representations for a FSA:

- The matrix representation,
- A directed and weighted graph
- Or transition functions (relations)

a) Transition function

f is the transition function of A $f(q, a) = q_1$

Indicates that if the automata is in state q and it encounters the symbol a , it goes to state q_1 .

Moreover for all q of Q $f(q, \epsilon) = q$

Chapter 4: Regular languages

Example:

Let A be the FSA defined by the quintuplet (V_t, Q, q_0, f, F) such that:

$$V_t = \{a, b\},$$

$$Q = \{q_0, q_1\},$$

q_0 : initial state

$$F = \{q_1\}$$

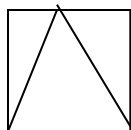
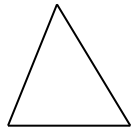
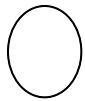
and we define the following transitions:

$$f(q_0, a) = q_0, \quad f(q_0, b) = q_1, \quad f(q_1, b) = q_1$$

Chapter 4: Regular languages

b) Directed Graph :

We represent a finite state automata by a directed and valued graph, whose arcs carry symbols of V_t and whose nodes carry the states.



state

initial state

final state

Initial and final state

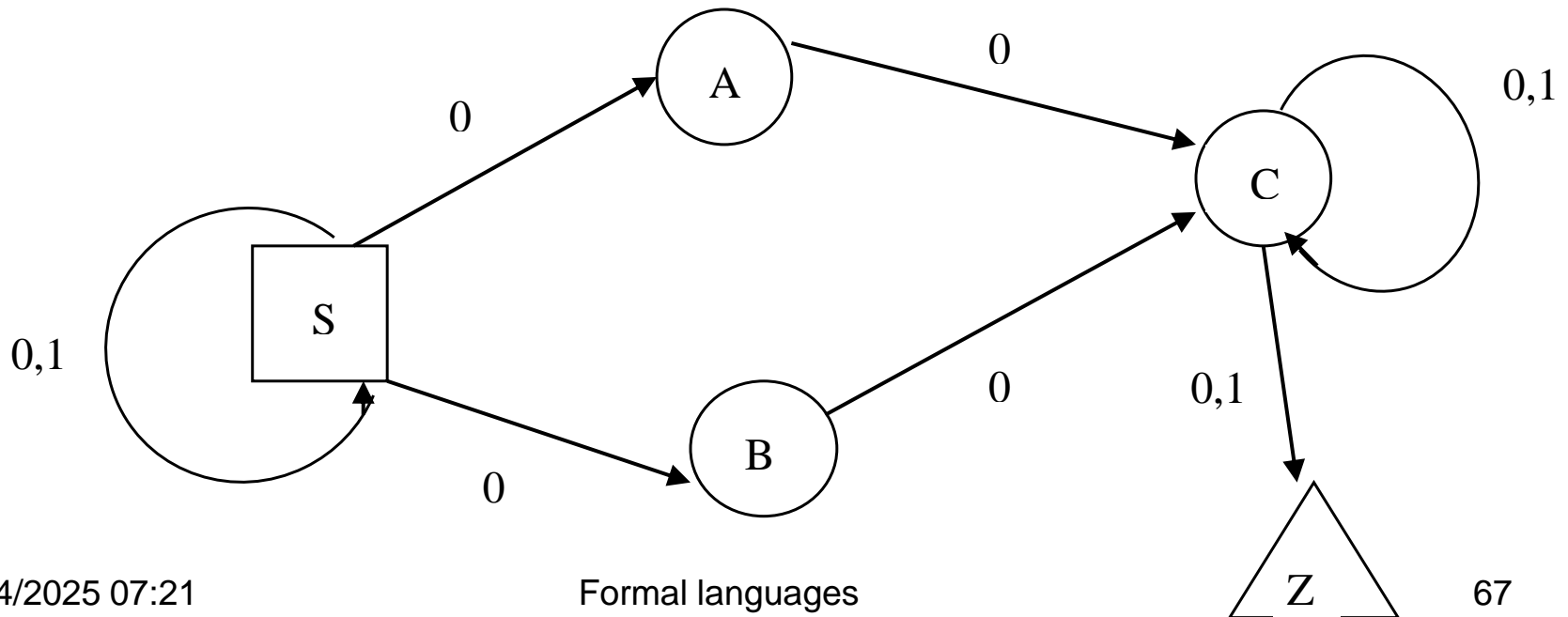
Chapter 4: Regular languages

Example: From this graph, we can define the automata : (V_t, Q, q_0, f, F)

$V_t = \{0,1\}$ $Q = \{S, A, B, C, Z\}$ $q_0 = S$ $F = \{Z\}$

$f(S,0)=S$ $f(S,1)=S$ $f(S,0)=A$ $f(S,1)=B$ $f(A,0)=C$ $f(B,1)=C$ $f(C,0)=C$

$f(C,1)=C$ $f(C,0)=Z$ $f(C,1)=Z$



Chapter 4: Regular languages

c) Table of transitions (matrix):

Let A be the FSA defined by the quintuplet (V_t, Q, q_0, f, F) such that:

$V_t = \{a, b\}$, $Q = \{q_0, q_1\}$, q_0 : initial state $F = \{q_1\}$

and we define the following transitions: $f(q_0, a) = q_0$, $f(q_0, b) = q_1$,

$f(q_1, b) = q_1$

The transition function can be represented by this matrix:

Q \ V _t	a	b
	q ₀	q ₁
q ₀	q ₀	q ₁
q ₁		q ₁

Chapter 4: Regular languages

Language recognized by a finite state automata:

The language recognized by a finite state automata is the set of strings whose symbols lead from the initial state to one of its final states by a succession of transitions using all its symbols in order.

Chapter 4: Regular languages

Definition of a configuration

The configuration of the FSA A , at a certain time, is given by the current state of the FSA and of the word which remains to be read:

(current state, word which remains to be read).

The initial configuration is (q_0, ω) where q_0 is the initial state of the FSA and ω the word submitted to A (to be recognized).

The final configuration is given by (q_f, ε) where q_f is a final state and the empty word indicates that there is nothing left to read.

➔ (the word belongs to the language).

Chapter 4: Regular languages

Definition of a direct derivation:

We say that a configuration $(q_1, a\omega)$ directly derives the configuration (q_2, ω) if and only if $f(q_1, a) = q_2$ where f is the transition function, $a \in V_t$ and $\omega \in V_t^*$.

We denote $(q_1, a\omega) \models (q_2, \omega)$.

Definition of an indirect derivation:

We say that a configuration (q, ω_1) indirectly derives another configuration (p, ω_2) , if and only if there exist 0, 1 or several direct derivations which, from (q, ω_1) , lead to the configuration (p, ω_2) .

We denote $(p, \omega_1) \models^* (q, \omega_2)$.

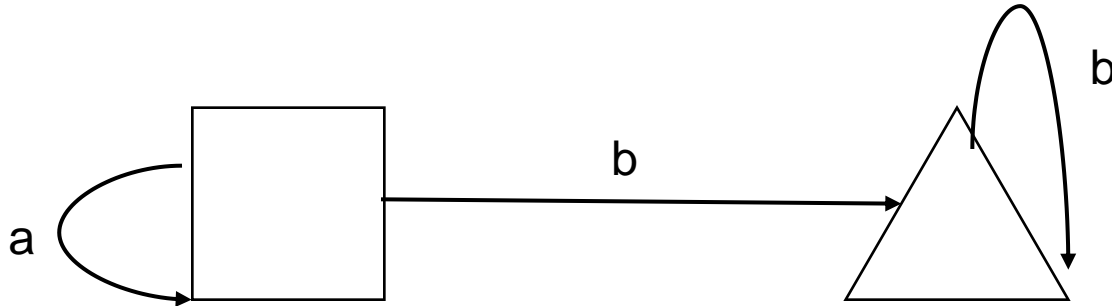
Chapter 4: Regular languages

Definition of the language recognized by a FSA :

The language recognized by the FSA A denoted $L(A)$ is defined by:

$$L(A) = \{ \omega \in Vt^* / (q_0, \omega) \models^* (q_f, \varepsilon) \} \quad \text{with} \quad q_f \in F$$

Chapter 4: Regular languages



Simple example :

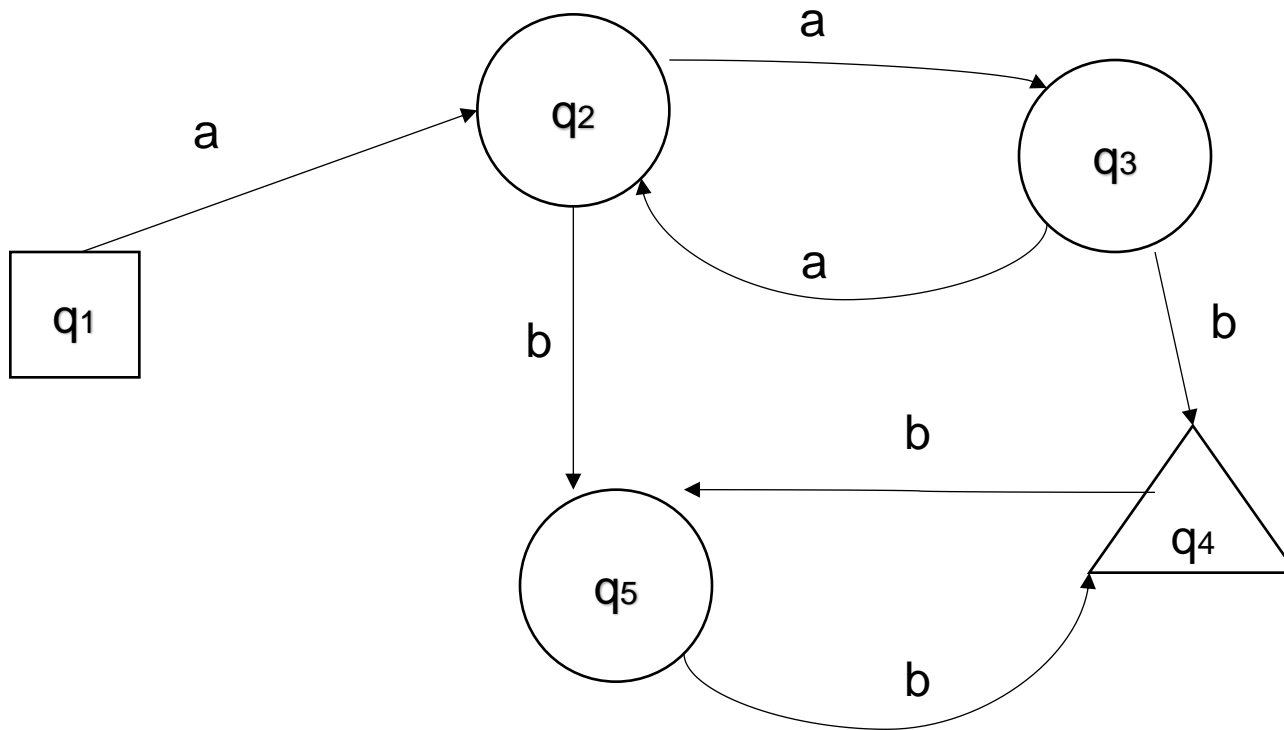
The minimal string is : b

The language recognized by this FSA is:

$$L(A) = \{ a^n b b^m / n \geq 0 ; m \geq 0 \}$$

Chapter 4: Regular languages

Example 2: Find the language recognized by this Automata



Note: the state 4 is a final state and at the same time is an internal state

Chapter 4: Regular languages

The possible paths:

- $q_1 q_2 q_5 q_4$
- $q_1 q_2 q_3 q_4$
- $q_1 q_2 q_3 q_2 q_5 q_4$
- $q_1 q_2 q_3 q_4 q_5 q_4$

The minimum strings:

- aab et abb

The recognized strings:

- $a (aa)^* b b (bb)^*$ - $aa (aa)^* b (bb)^*$

$$L(A) = \{ a (aa)^* b b (bb)^* \} \cup \{ aa (aa)^* b (bb)^* \}$$

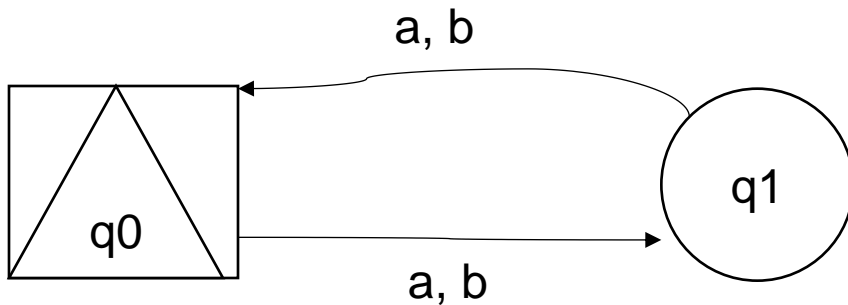
Chapter 4: Regular languages

Remarks:

- 1- A finite state automata recognizes a single language, but the same language can be recognized by several automata,
- 2- We say that two finite state automata A_1 and A_2 are equivalent if and only if these two automata recognize the same language. $L(A_1) = L(A_2)$

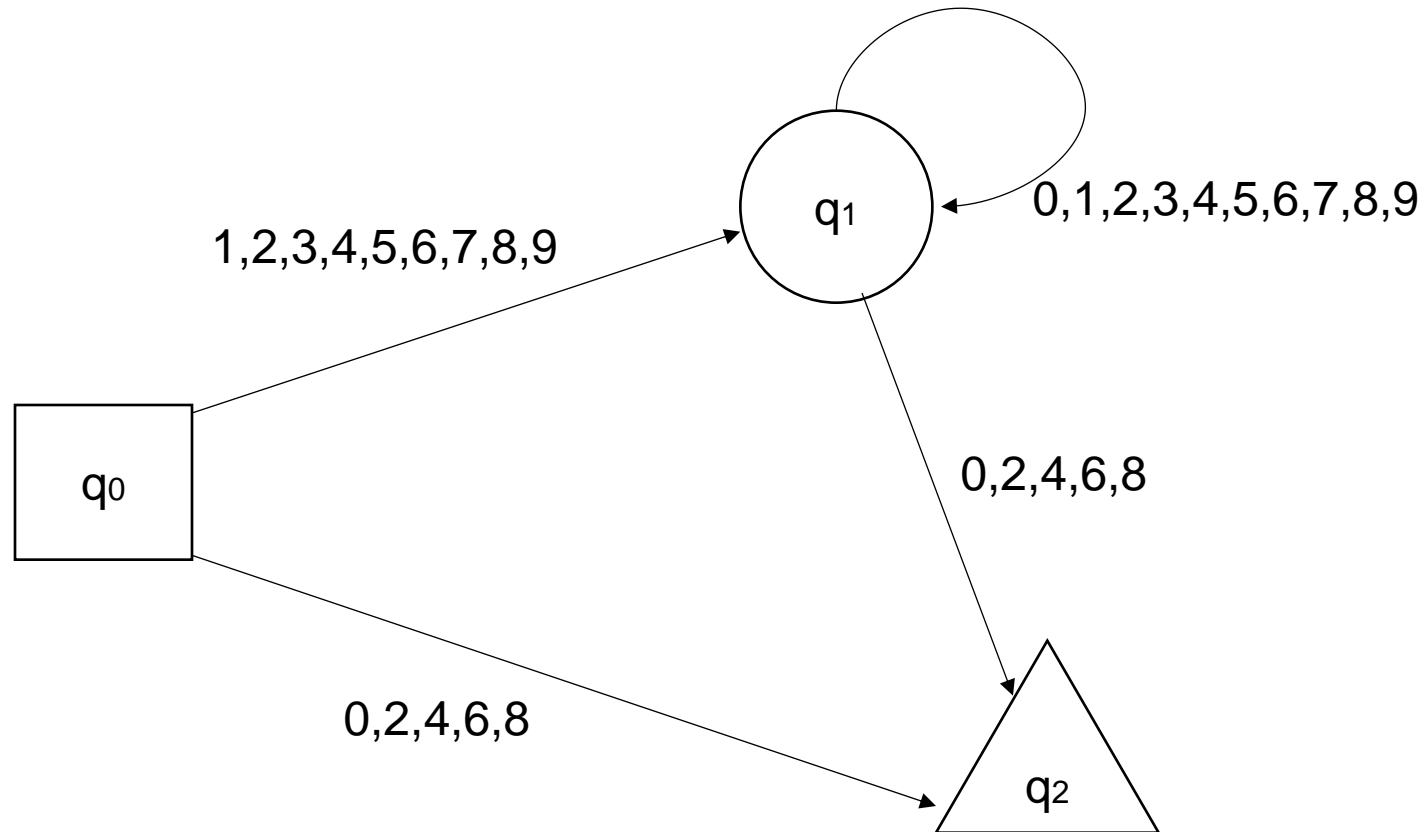
Example of automata construction:

$$L1 = \{ w / w \in \{a,b\}^* \text{ et } |w| \equiv 0 [2] \}$$



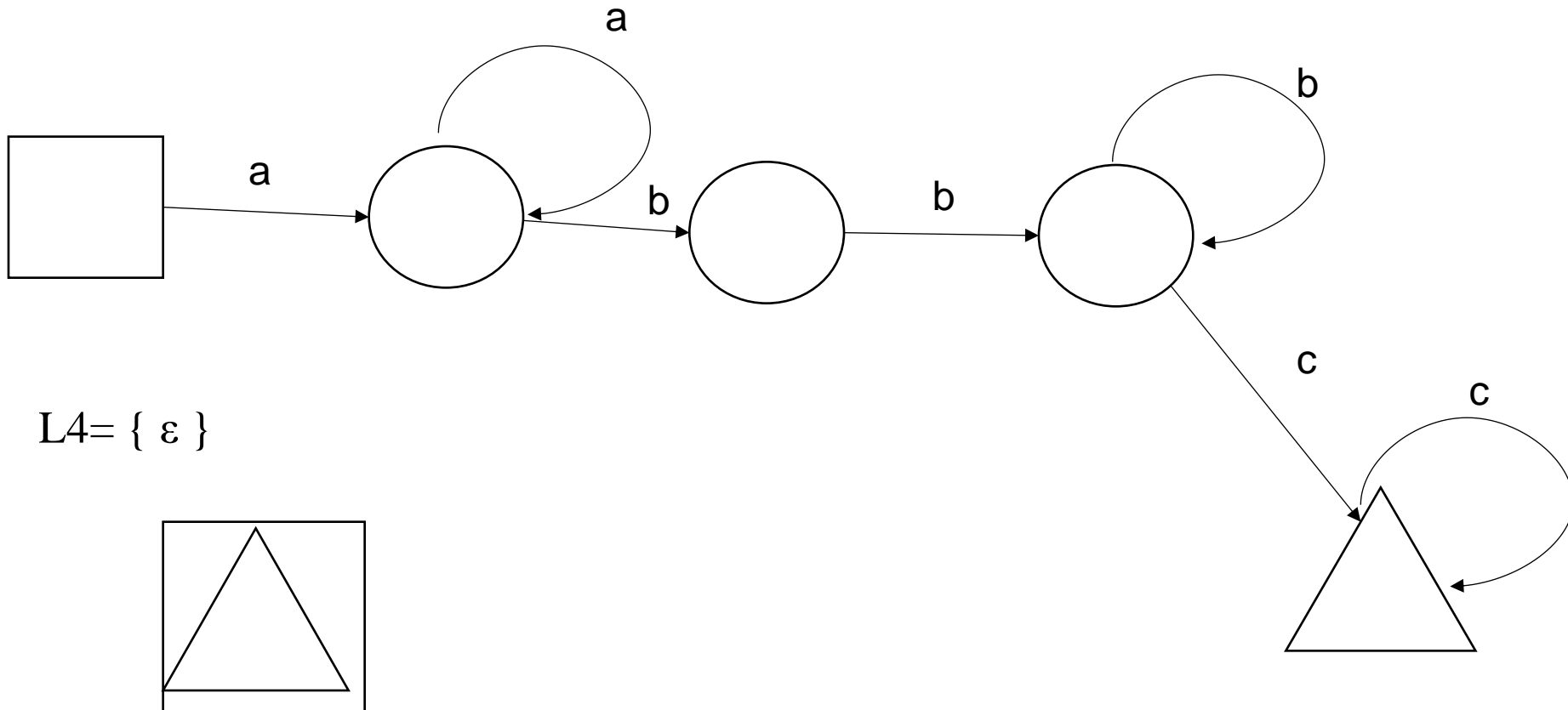
Chapter 4: Regular languages

$L_2 = \{ w \mid w \in N \text{ and } w \equiv 0 [2] \}$



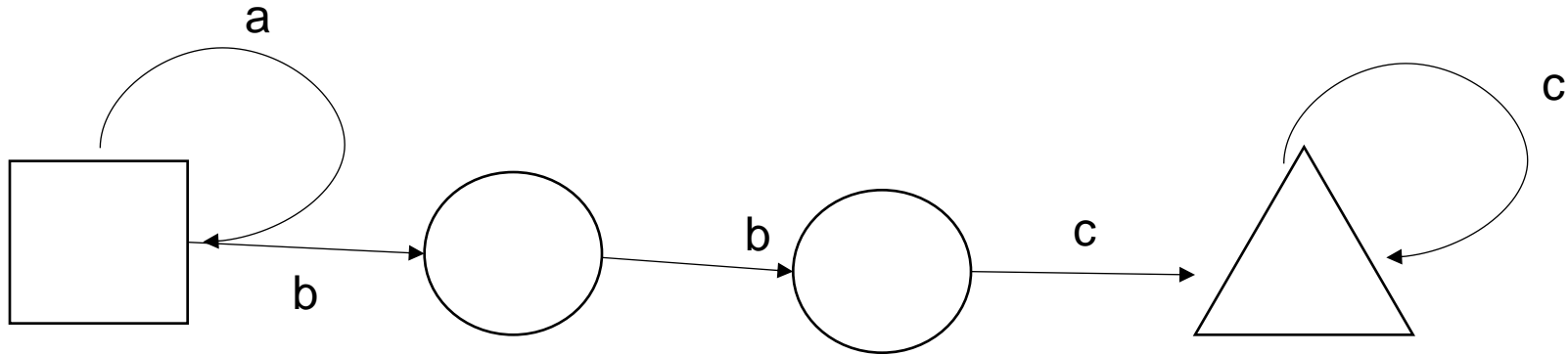
Chapter 4: Regular languages

$$L3 = \{ a^i b^j c^k \mid i \geq 1, k \geq 1 \text{ and } j > 1 \}$$



Chapter 4: Regular languages

$$L5 = \{ a^i b^2 c^k \mid i \geq 0 \text{ and } k > 0 \}$$



Chapter 4: Regular languages

4.4 Variants of finite state automata

a) **Deterministic Finite State Automata (DFSA)**

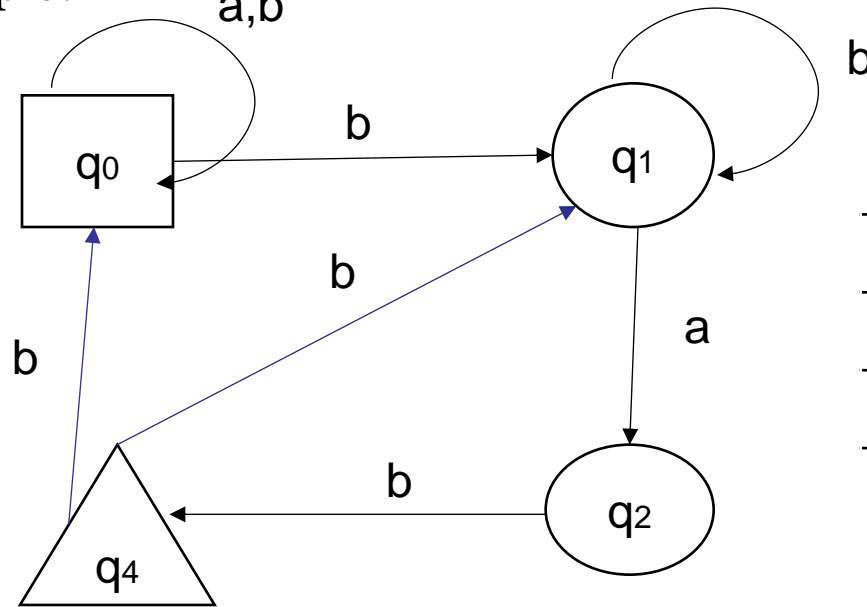
A deterministic finite state automata is an automata such that its transition function is a function. For any state and for any symbol, there exists at most one state in which the automata can pass.

b) **Nondeterministic finite state automata (NDFSA)**

A nondeterministic finite state automata is an automata such that there exists at least one pair formed by a state and a symbol, which admits more than one image by the transition function. The automata must make choices to progress.

Chapter 4: Regular languages

Example:



	a	b
q0	q0	q0, q1
q1	q2	q1
q2		q4
q4	q0, q1	

This automata is non-deterministic: for the same state and the same symbol we have two states, we find more than one value in a cell of the transition table

$$f(q_0, b) = q_0 \quad \text{et} \quad f(q_0, b) = q_1$$

$$f(q_4, b) = q_0 \quad \text{et} \quad f(q_4, b) = q_1$$

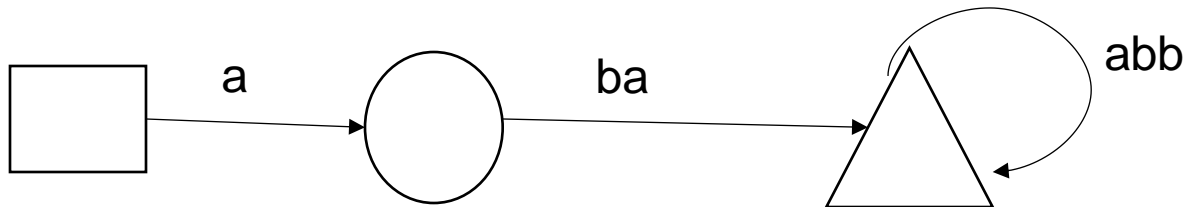
Chapter 4: Regular languages

The problem of the NDFSA :

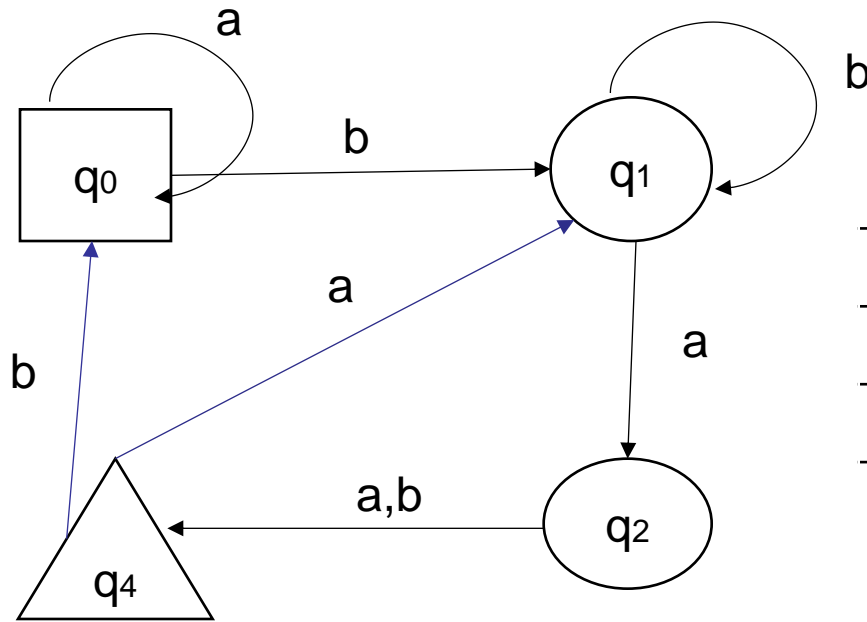
These automata analyze the strings more slowly than deterministic automata, we must make choices to progress and make feedback afterwards in the case of failure

c) Generalized finite state automata GFSA)

- Transitions can be generated by words instead of symbols.
- The transitions caused by the empty word (ϵ) are called spontaneous or empty transitions (ϵ -transition). It is a change of state without reading.



Chapter 4: Regular languages



	a	b
q0	q0	q1
q1	q2	q1
q2	q4	q4
q4	q1	q0

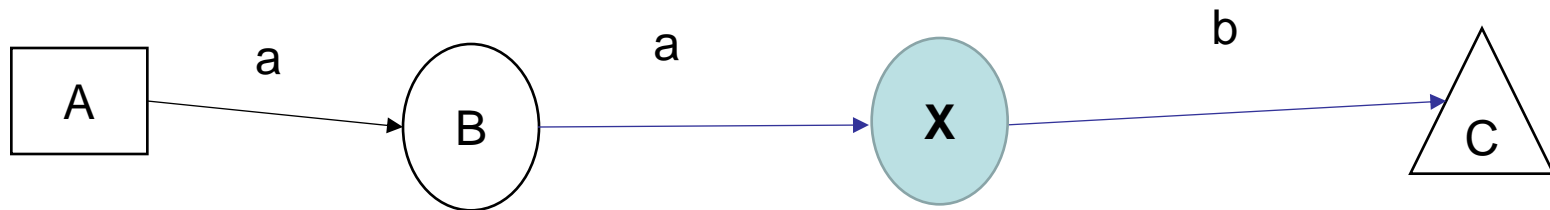
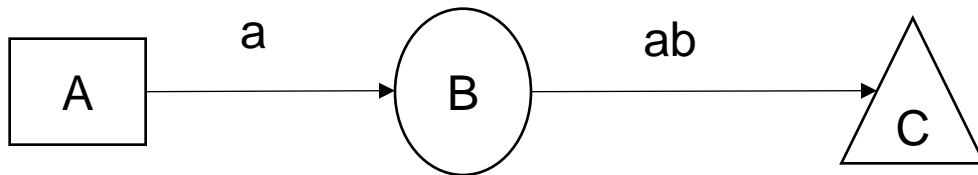
d) Complete state machine(CSM)

An automata is called complete if it is **deterministic** and for each state and for each symbol there is exactly one transition..

Chapter 4: Regular languages

4.5 Transformation of a generalized automata into a simple automata

Any generalized finite state automata admits an equivalent simple automata by adding additional transitions.



Chapter 4: Regular languages

4.6 Transformation from a non-deterministic FSA to a deterministic automata

To any non-deterministic finite state automata corresponds an equivalent deterministic finite state automata and vice versa.

The transition from a non-deterministic automata to a deterministic automata is done according to the following algorithm:

The goal is to find the elements of the deterministic automata.

$A = (V_t, Q, q_0, f, F)$ non-déterministic automata given

$A' = (V'_t, Q', q'_0, f', F')$ déterministic automata accepting the same language

Chapter 4: Regular languages

$$1) V'_t = V_t$$

$$2) q'_0 = \{q_0\}$$

3) Q' is included in the set of combinations of Q , $P(Q)$

$$4) F' = \{B \in Q' \mid B \cap F \neq \emptyset\} \implies B \in F'$$

$$5) f' : f'(B, x) = M$$

$$\text{let } B = \{q_1, q_2, \dots, q_i, \dots, q_k\}$$

$$f(q_1, x) = M_1 \dots f(q_i, x) = M_i, \dots, f(q_k, x) = M_k$$

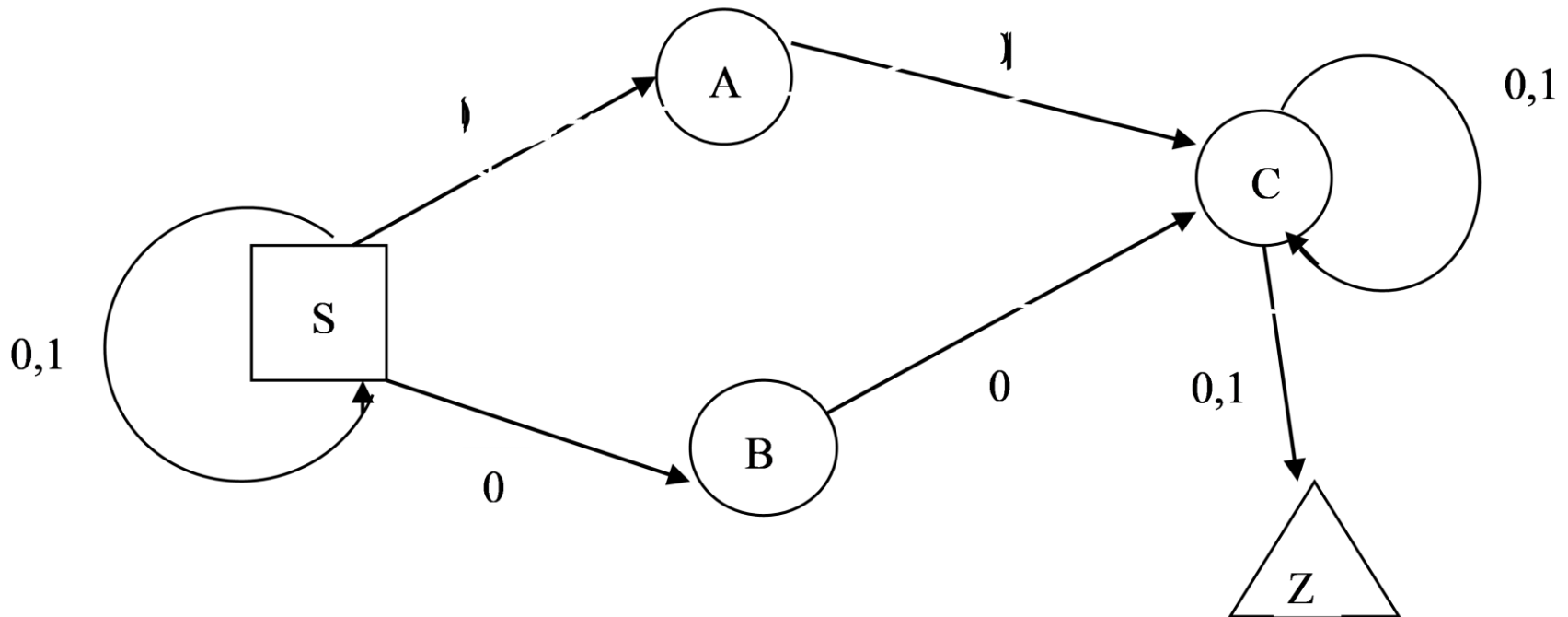
$$M_i \in Q' \quad M_i \text{ included in } P(Q)$$

$$M = \bigcup_{1 \leq i \leq k} M_i$$

6) Remove empty transitions.

Chapter 4: Regular languages

Example: find the deterministic automata



The construction of the new transitions table is done as follows :

Chapter 4: Regular languages

	0	1
$\{S\} = q_0$	$\{S,B\} q_1$	$\{S,A\} q_2$
$\{S,B\} = q_1$	$\{S,B,C\} q_3$	$\{S,A\} q_2$
$\{S,A\} = q_2$	$\{S,B\} q_1$	$\{S,A,C\} q_4$
$\{S,B,C\} = q_3$	$\{S,B,C,Z\} q_5$	$\{S,A,C,Z\} q_6$
$\{S,A,C\} = q_4$	$\{S,B,C,Z\} q_5$	$\{S,A,C,Z\} q_6$
$\{S,B,C,Z\} = q_5$	$\{S,B,C,Z\} q_5$	$\{S,A,C,Z\} q_6$
$\{S,A,C,Z\} = q_6$	$\{S,B,C,Z\} q_5$	$\{S,A,C,Z\} q_6$

New transition table of the deterministic automata

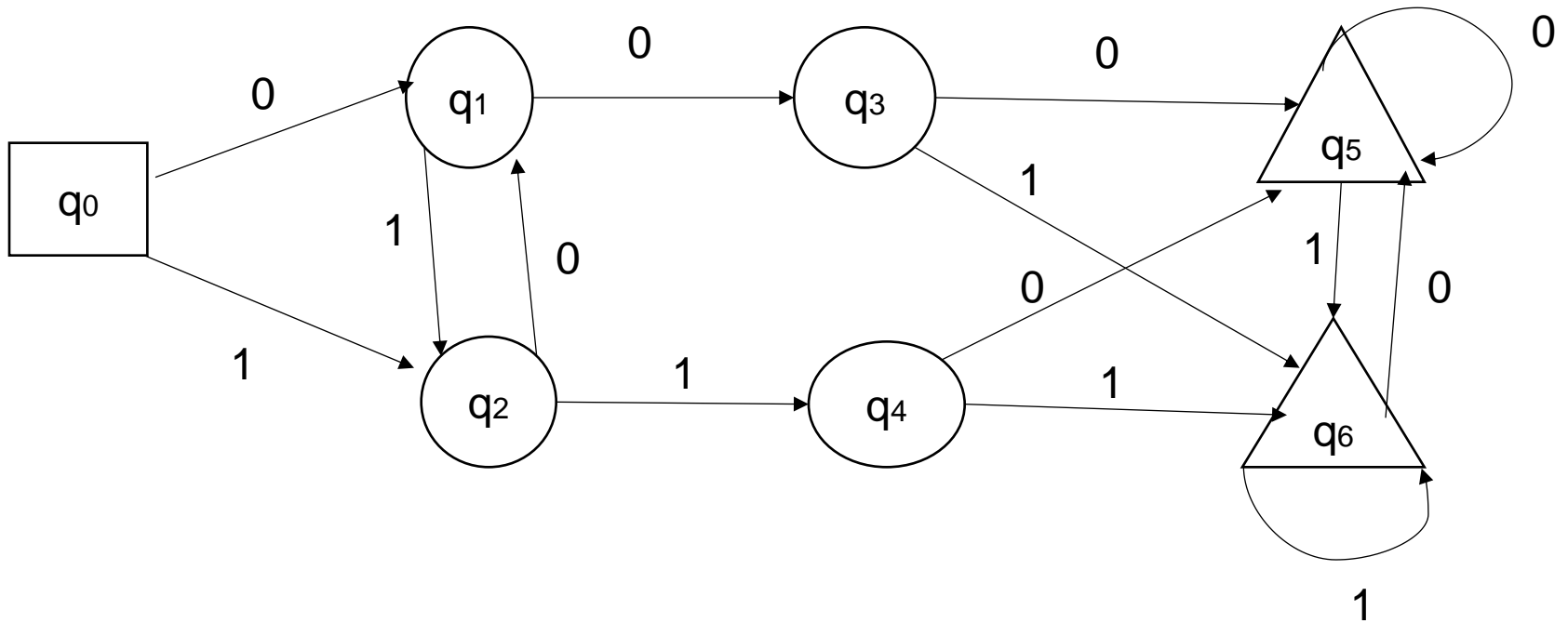
Initial state : q_0

$F = \{q_5, q_6\}$

$Z \in q_5$ and $Z \in q_6$

Chapter 4: Regular languages

The FSA deterministic equivalent



Chapter 4: Regular languages

4.6 Elimination of empty transitions

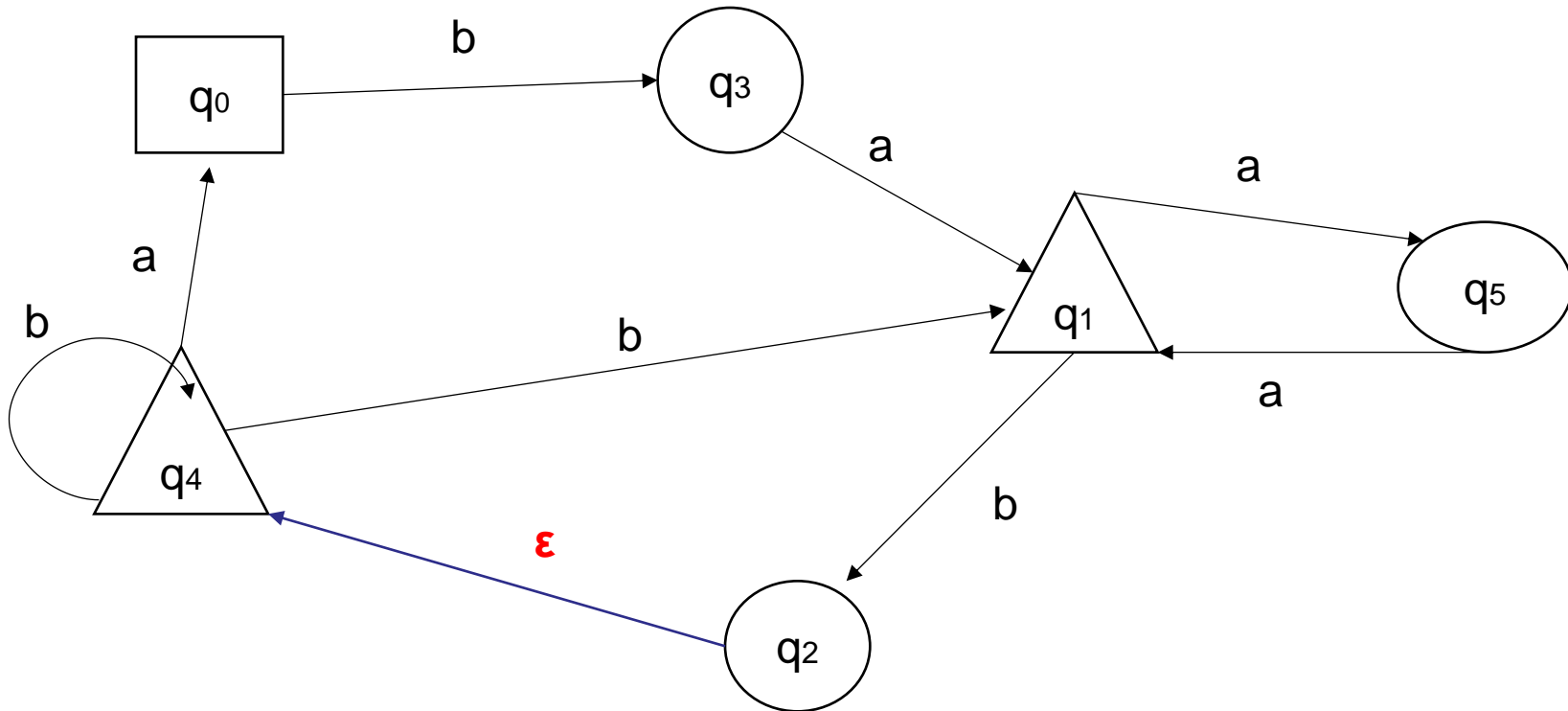
Removing these transitions gives us a simple FSA , to do this we must first remove the transitions by ε :

$$\begin{aligned} & \text{if } q_j \in F \quad \text{then} \quad q_i \in F \\ \delta(q_i, \varepsilon) = q_j \quad \{ & \quad \text{and} \\ \forall a \in Vt : \text{if } \delta(q_j, a) = q_k \quad \text{then} \quad & \delta(q_i, a) = q_k \end{aligned}$$

An example of this transformation is shown on the following FSA:

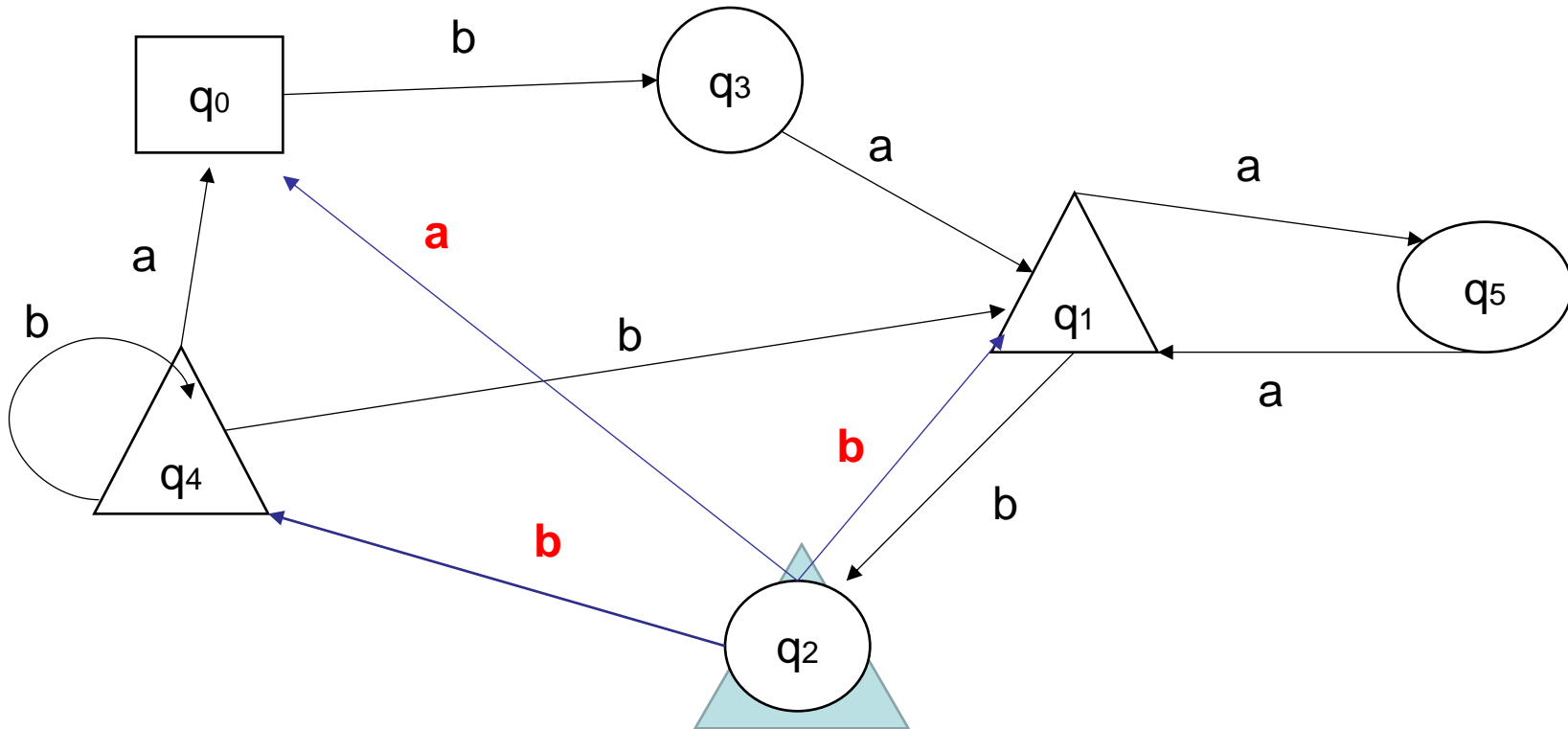
Chapter 4: Regular languages

We have a single ϵ transition $f(q_2, \epsilon) = q_4$, q_4 final state so q_2 becomes final state and we have 3 transitions $f(q_4, a) = q_0$, $f(q_4, b) = q_1$ and $f(q_4, b) = q_4$, so we add 3 transitions replacing q_4 by q_2



Chapter 4: Regular languages

We remove the empty transition and replace it with these transitions: $f(q_2, a) = q_0$, $f(q_2, b) = q_1$ and $f(q_2, b) = q_4$



Chapter 4: Regular languages

4.7 Transition from regular grammar to FSA

For any regular grammar $G = (V_t, V_n, S, R)$, there exists a FSA

$A = (V_t, Q, q_0, f, F)$ such that $L(G) = L(A)$.

The passage is done by associating a transition to each rule of the grammar.

The built automata is not automatically deterministic.

Let $G = (V_t, V_n, S, R)$, a regular grammar, the question is how to find a FSA

$A = (V_t', Q, q_0, f, F)$ such that $L(G) = L(A)$

Chapter 4: Regular languages

4.7 Transition from regular grammar to FSA

$A = (Vt', Q, q_0, f, F)$ such that $L(G) = L(A)$

1) $Vt' = Vt$

2) $Q = Vn \cup q_f$ such that $q_f \in F$

3) $q_0 = S$

4) $F = \{q_f\}$.

5) For each rule: $A \rightarrow a B$, we associate the transition $f(A, a) = B$.

For each rule of the form $A \rightarrow a$, we associate the transition $f(A, a) = q_f$.

6) If the grammar has the rule $S \rightarrow \varepsilon$ then S becomes a final state $F = \{q_f, S\}$

Chapter 4: Regular languages

4.7 Example

Let's find the FSA equivalent to the following grammar:

$$G = (\{a,b,c\}, \{S,A,B\}, S, R)$$

$$R = (S \rightarrow aS / bA$$

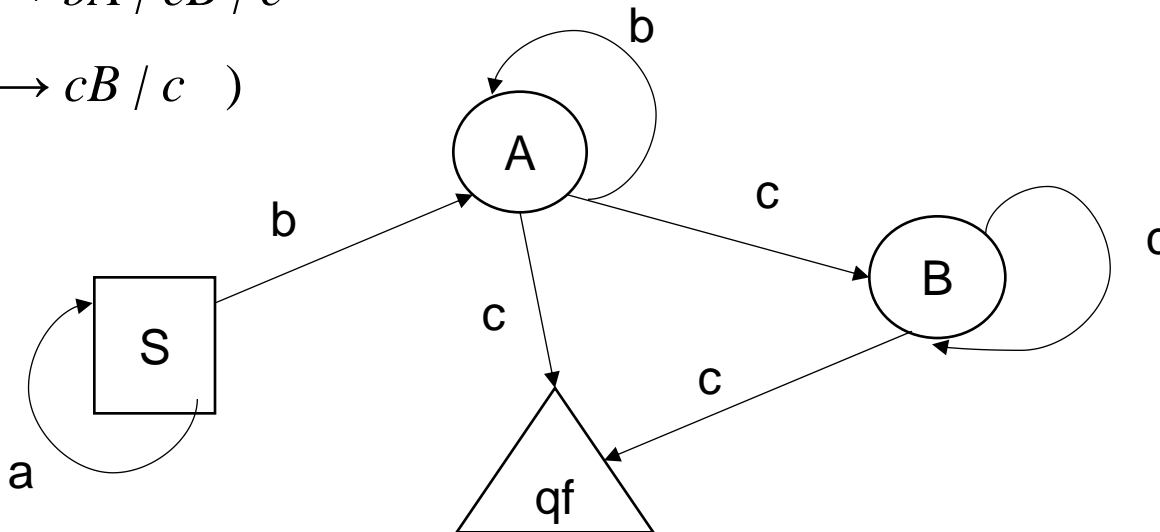
$$A \rightarrow bA / cB / c$$

$$B \rightarrow cB / c)$$

$$Q = \{ S, A, B, qf \}$$

$$F = \{ qf \}$$

initial state : S



Chapter 4: Regular languages

4.8 Transition from FSA to regular grammar:

For any FSA $A = (V_t, Q, q_0, f, F)$ there exists an equivalent regular grammar

$G = (V_{t'}, V_n, S, R)$ such that $L(G) = L(A)$.

The transition is done as follows:

- 1) $V_{t'} = V_t$ 2) $V_n = Q$ 3) $q_0 = S$
- 4) $\left\{ \begin{array}{lll} \text{if } f(q, a) = p \in A & \text{then} & q \rightarrow ap \in R \\ \text{If } f(q, a) = q_f \in A & \text{then} & q \rightarrow a \in R \\ \text{if } q_0 \in F & \text{then} & q_0 \rightarrow \varepsilon \in R \end{array} \right.$

4.8 Example : Find the grammar equivalent to this automata:

$G = (\{a,b\}, \{S,A,B,C,D\}, S, R)$

$R = f(S,a) = A$ become $S \rightarrow aA$

$f(A,a) = B$ become $A \rightarrow aB$

$f(A,b) = C$ become $A \rightarrow bC$

$f(S,a) = A$ become $S \rightarrow aA$

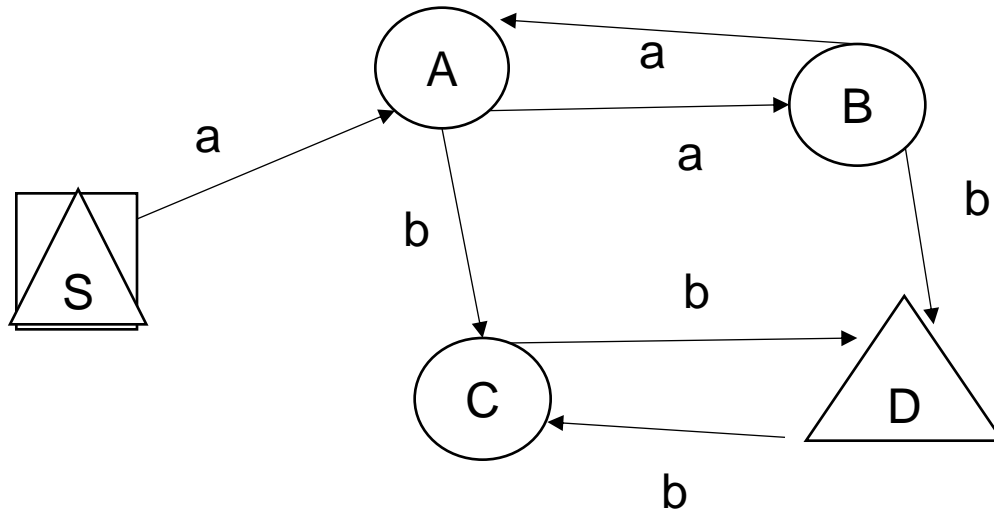
$f(B,a) = A$ become $B \rightarrow aA$

$f(B,b) = D$ (D **internal**) become $B \rightarrow bD$

$f(B,b) = D$ (D final) become $B \rightarrow b$

$f(D,b) = C$ become $D \rightarrow bC$

$f(C,b) = D$ (D **internal**) become $C \rightarrow bD$ $f(C,b) = D$ (D final) become $C \rightarrow b$



The State **S** is an initial and final state at the same time, so we add the rule:

$S \rightarrow \epsilon$

Chapter 4: Regular languages

4.9 Regular expressions

a) Definition :

Regular expressions (RE) provide another method of defining regular languages. They are more practical than the other two systems (regular grammars and automata).

Each regular expression describes a set of terminal strings.

The regular expression formalism uses 03 operations:

1. Concatenation
2. Closing noted * (power)
3. The alternative noted + or / (choice between two expressions)

Chapter 4: Regular languages

4.9 Regular expressions

b) Formal definition:

- ϕ is a regular expression which denotes (represents) the empty language
- ε is a regular expression which denotes the language $\{\varepsilon\}$
- a (where $a \in V_t$) is a regular expression which denotes the language $\{a\}$.

Induction:

ε and a are regular expressions;

If e, e' are regular expressions then $e+e'$, $e.e'$, e^* are regular expressions.

Remarks :

- The exponent has a higher priority than the concatenation which has a higher priority than the sum.
- Two regular expressions are ε -equivalent if and only if they denote the same language.

Chapter 4: Regular languages

4.9 Regular expressions

c) Examples :

$(a + b)^* = \{ \epsilon, a, b, aa, bb, ab, ba, aaa, bbb, aab, aba, \dots \}$ infinite language

$a^*b + b^*a = \{ b, ab, aab, aaab, aa\dots ab, a, ba, bba, bbba, bb\dots ba, \dots \}$

$ab^*(c + a) = ab^*c + ab^*a = \{ ac, abc, abbc, abbbc, \dots, aa, aba, abba, abbba, \dots \}$

Chapter 4: Regular languages

Regular Expressions	Regular Set
$(0 + 10^*)$	$L = \{ 0, 1, 10, 100, 1000, 10000, \dots \}$
(0^*10^*)	$L = \{1, 01, 10, 010, 0010, \dots\}$
$(0 + \epsilon)(1 + \epsilon)$	$L = \{\epsilon, 0, 1, 01\}$
$(a+b)^*$	Set of strings of a's and b's of any length including the null string. So $L = \{ \epsilon, a, b, aa, ab, bb, ba, aaa, \dots \}$
$(a+b)^*abb$	Set of strings of a's and b's ending with the string abb. So $L = \{abb, aabb, babb, aaabb, ababb, \dots\}$
$(11)^*$	Set consisting of even number of 1's including empty string, So $L = \{\epsilon, 11, 1111, 111111, \dots\}$
$(aa)^*(bb)^*b$	Set of strings consisting of even number of a's followed by odd number of b's, so $L = \{b, aab, aabbb, aabbbbb, aaaab, aaaabbb, \dots\}$
$(aa + ab + ba + bb)^*$	String of a's and b's of even length can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null, so $L = \{aa, ab, ba, bb, aaab, aaba, \dots\}$

Chapter 4: Regular languages

4.10 Transition from regular expression to regular grammar

$0 (0 + 1)^* 0$

$G = (\{0, 1\}, \{S, A\}, S, R)$

$R = (S \rightarrow 0 A \quad A \rightarrow 0 A \quad A \rightarrow 1 A \quad A \rightarrow 0 \quad)$

$(0 + 1)^* 0 (0 + 1) (0 + 1)$

$G = (\{a, b\}, \{S, A, B\}, S, R)$

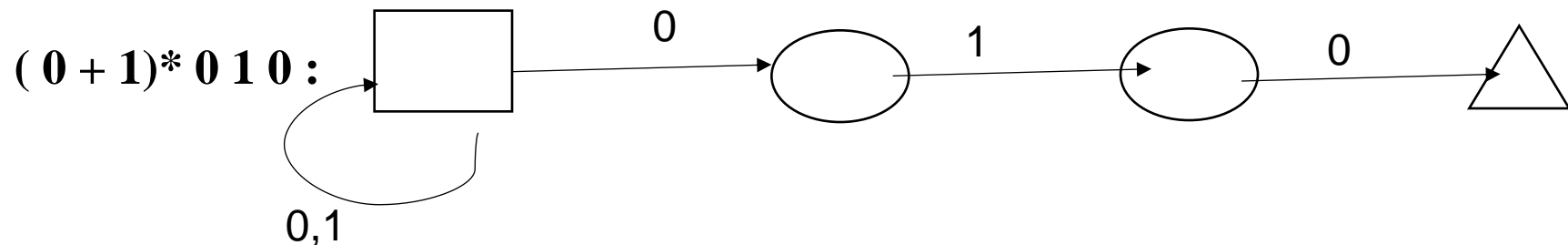
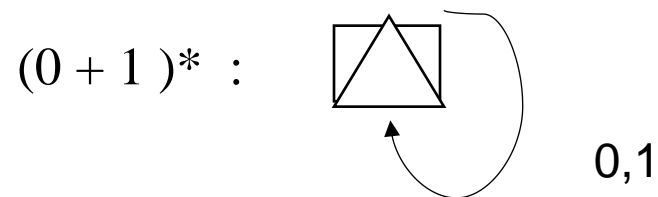
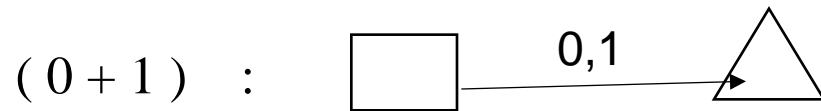
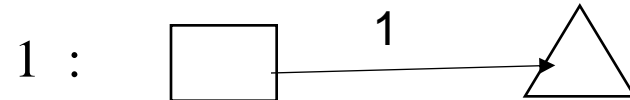
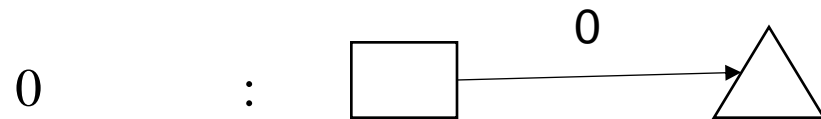
$R = (S \rightarrow 0S / 1S \quad S \rightarrow 0 A \quad A \rightarrow 0 B / 1B \quad B \rightarrow 0 / 1 \quad)$

Chapter 4: Regular languages

4.11 Transition from a regular expression to a FSA

There are automatas for each regular expression

Construction example: $(0 + 1)^* 0 1 0$



Chapter 4: Regular languages

4.11 Methods to show that a language is regular

We can show the regularity of a language L , by one of the following methods:

- All finite languages are regular
- If we find a FSA which recognizes a language L , then L is regular
- If we find a regular grammar generating L , then this language is regular
- We can exploit closure properties to show that a language is regular. (properties of regular expressions)

Chapter 4: Regular languages

4.13 Properties of regular languages

- The union of two regular languages is a regular language.
- The concatenation of two regular languages is a regular language
- The iteration of a regular language is a regular language

Chapter 4: Regular languages

4.14 Arden's Theorem

In order to find out a regular expression of a FSA , we use Arden's Theorem along with the properties of regular expressions.

Statement

Let P and Q be two regular expressions.

If P does not contain null string, then $R = Q + RP$ has a unique solution that is

$$R = QP^*$$

Proof

$$R = Q + (Q + RP)P \quad [\text{After putting the value } R = Q + RP]$$

$$= Q + QP + RPP$$

Chapter 4: Regular languages

4.14 Arden's Theorem

Proof

$$\begin{aligned} R &= Q + (Q + RP)P \quad [\text{After putting the value } R = Q + RP] \\ &= Q + QP + RPP \end{aligned}$$

When we put the value of R recursively again and again, we get the following equation:

$$R = Q + QP + QP^2 + QP^3 + \dots \qquad R = Q (\epsilon + P + P^2 + P^3 + \dots)$$

$$R = QP^* \quad [\text{As } P^* \text{ represents } (\epsilon + P + P^2 + P^3 + \dots)]$$

Assumptions for Applying Arden's Theorem

- The transition diagram must not have NULL transitions
- It must have only one initial state

Chapter 4: Regular languages

4.14 Arden's Theorem

Method

Step 1 – Create equations as the following form for all the states of the FSA having n states with initial state q_1 .

$$q_1 = q_1 R_{11} + q_2 R_{21} + \dots + q_n R_{n1} + \epsilon$$

$$q_2 = q_1 R_{12} + q_2 R_{22} + \dots + q_n R_{n2}$$

.....

.....

.....

$$q_n = q_1 R_{1n} + q_2 R_{2n} + \dots + q_n R_{nn}$$

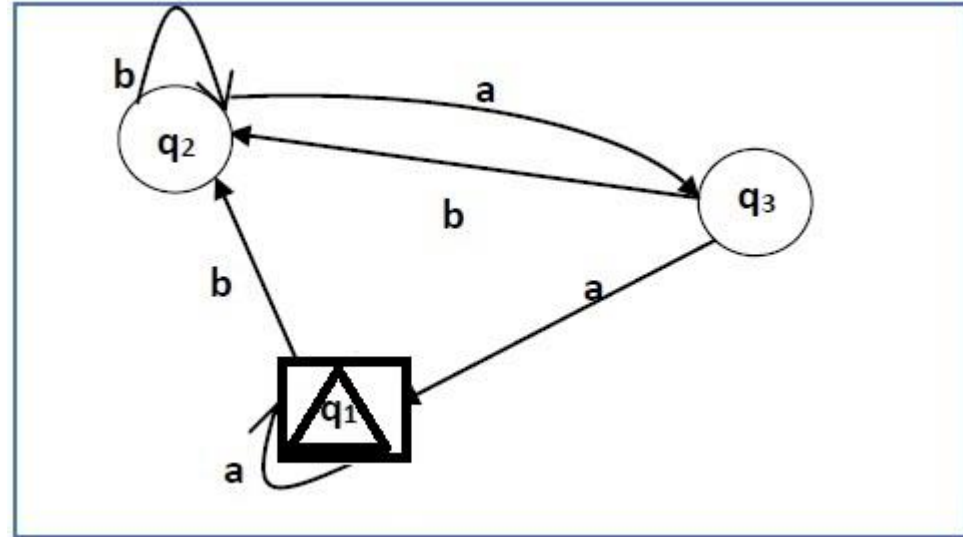
R_{ij} represents the set of labels of edges from q_i to q_j , if no such edge exists, then $R_{ij} = \emptyset$

Step 2 – Solve these equations to get the equation for **the final state** in terms of R_{ij}

Chapter 4: Regular languages

4.14 Arden's Theorem

Problem: Construct a regular expression corresponding to the automata given below:



Solution:

Here the initial state and final state is q_1 .

The equations for the three states q_1 , q_2 , and q_3 are as follows:

$$q_1 = q_1a + q_3a + \epsilon \quad (\epsilon \text{ move is because } q_1 \text{ is the initial state})$$

$$q_2 = q_1b + q_2b + q_3b$$

$$q_3 = q_2a$$

Chapter 4: Regular languages

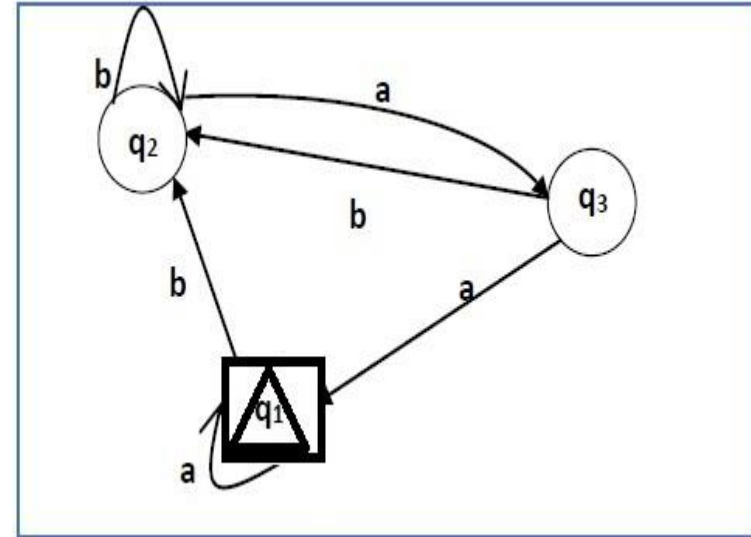
4.14 Arden's Theorem

Now, we will solve these three equations:

$$\begin{aligned} q_2 &= q_1b + q_2b + q_3b \\ &= q_1b + q_2b + (q_2a)b \text{ (Substituting value of } q_3) \\ &= q_1b + q_2(b + ab) \\ &= q_1b(b + ab)^* \text{ (Applying Arden's Theorem)} \end{aligned}$$

$$\begin{aligned} q_1 &= q_1a + q_3a + \varepsilon \\ &= q_1a + q_2aa + \varepsilon \text{ (Substituting value of } q_3) \\ &= q_1a + q_1b(b + ab)^*aa + \varepsilon \text{ (Substituting value of } q_2) \\ &= q_1(a + b(b + ab)^*aa) + \varepsilon \\ &= \varepsilon(a + b(b + ab)^*aa)^* \\ &= (a + b(b + ab)^*aa)^* \end{aligned}$$

Hence, the regular expression is $(a + b(b + ab)^*aa)^*$.



Chapter 4: Regular languages

4.14 Arden's Theorem

Problem

Construct a regular expression corresponding to this automata:

Solution:

Here the initial state is q_1 and the final state is q_2

Now we write down the equations:

$$q_1 = q_1 0 + \varepsilon$$

$$q_2 = q_1 1 + q_2 0$$

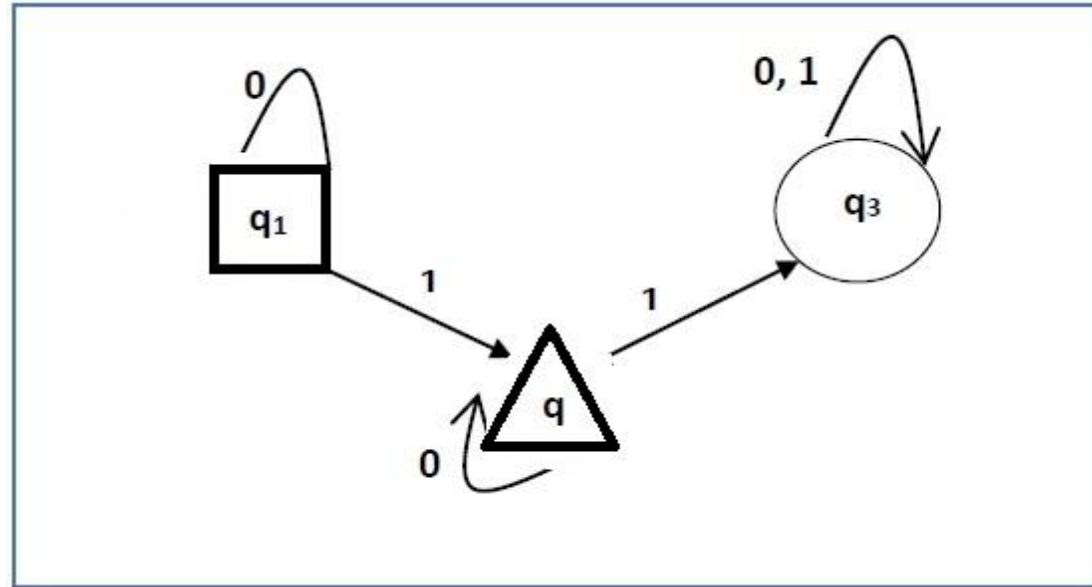
$$q_3 = q_2 1 + q_3 0 + q_3 1$$

Now, we will solve these three equations;

$$q_1 = \varepsilon 0^* \quad [R = Q + RP] \quad \text{So,} \quad q_1 = 0^*$$

$$q_2 = 0^* 1 + q_2 0 \quad \text{So,} \quad q_2 = 0^* 1 (0)^* \quad [\text{By Arden's theorem}]$$

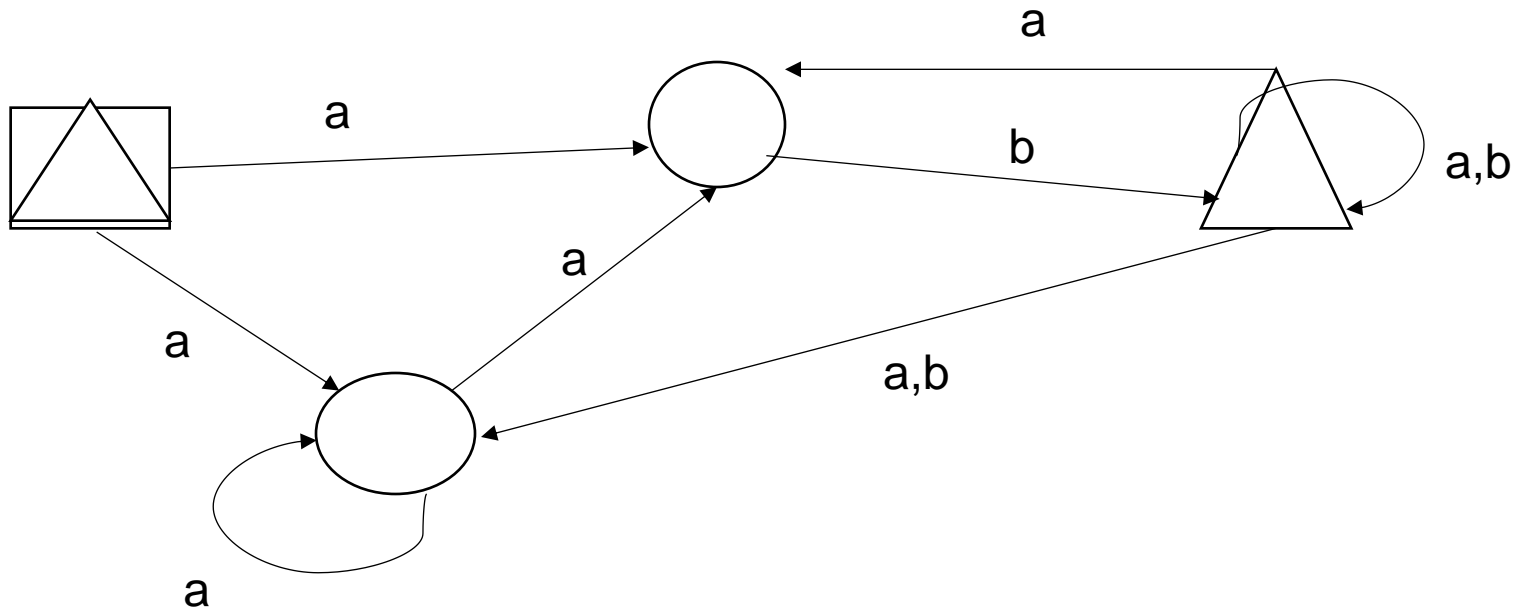
Hence, the regular expression is **$0^* 1 0^*$** .



Chapter 4: Regular languages

4.14 important Example

- Build a FSA equivalent to this RE :
- $(a^* a b (a + b)^*)^*$



Remark : Avoid the loop in the initial state.

Chapter 4: Regular languages

4.15 Minimization of DFSA

DFSA minimization stands for converting a given DFSA to its equivalent DFSA with minimum number of states.

There are many methods to minimize DFSA. The most used is equivalence method.

Step-01:

- Eliminate all the **dead states** and **inaccessible states** from the given DFSA (if any).

Step-02:

- Draw a state transition table for the given DFSA.
- Transition table shows the transition of all states on all input symbols

Chapter 4: Regular languages

Step-03:

Now, start applying equivalence theorem.

- Take a counter variable k and initialize it with value 0. $k \leftarrow 0$
- Divide Q (set of states) into two sets such that one set contains **all the non-final states** and other set contains all **the final states**.
- This partition is called P_0 . 0 equivalence

Step-04:

Increment k by 1.

- Find P_k by partitioning the different sets of P_{k-1} .
- In each set of P_{k-1} , consider all the possible pair of states within each set and if the two states are distinguishable, partition the set into different sets in P_k .

Two states q_1 and q_2 are distinguishable in partition P_k for any input symbol 'a', if $f(q_1, a)$ and $f(q_2, a)$ are in different sets in partition P_{k-1} .

Chapter 4: Regular languages

Step-05:

- Repeat step-04 until **no change** in partition occurs.
- In other words, when you find $P_k = P_{k-1}$, stop.

Step-06:

- All those states which belong to the same set **are equivalent**.
- The equivalent states are **merged** to form a single state in the minimal DFSA.

Chapter 4: Regular languages

4.15 Minimization of DFSA : Example

Minimize this automata:

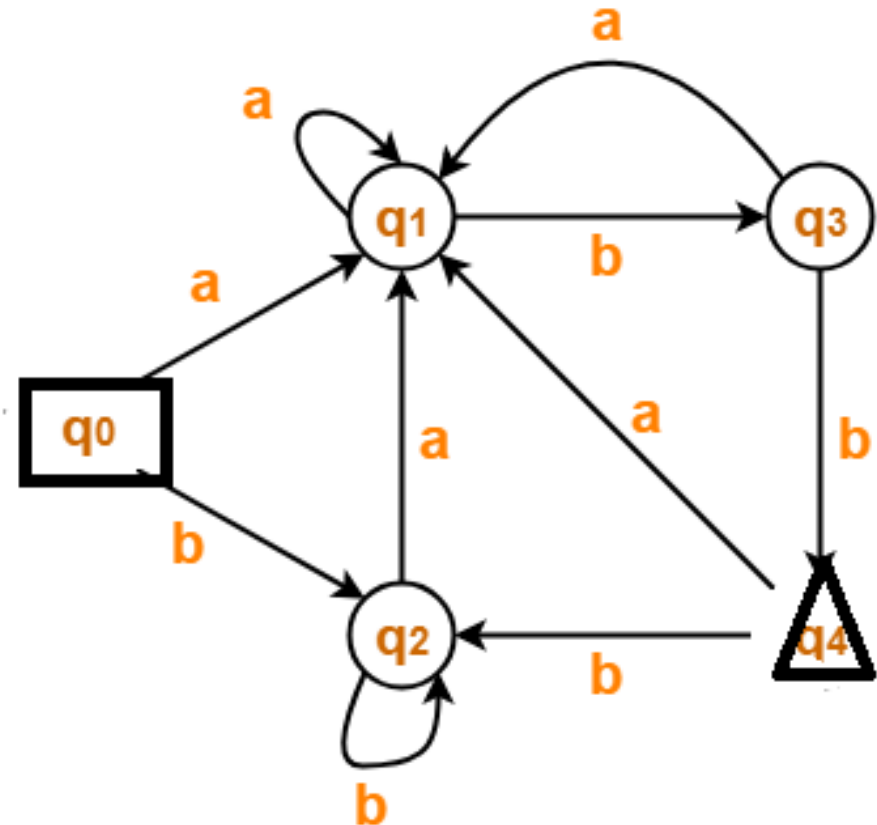
Step-01:

The given DFSA contains no dead states and inaccessible states.

Step-02:

Draw a state transition table-

	a	b
→ q0	q1	q2
q1	q1	q3
q2	q1	q2
q3	q1	q4
← q4	q1	q2



Chapter 4: Regular languages

4.15 Minimization of DFSA : Example

Step-03:

Now using Equivalence Theorem, we have:

$$P_0 = \{ q_0, q_1, q_2, q_3 \} \{ q_4 \}$$

$$P_1 = \{ q_0, q_1, q_2 \} \{ q_3 \} \{ q_4 \}$$

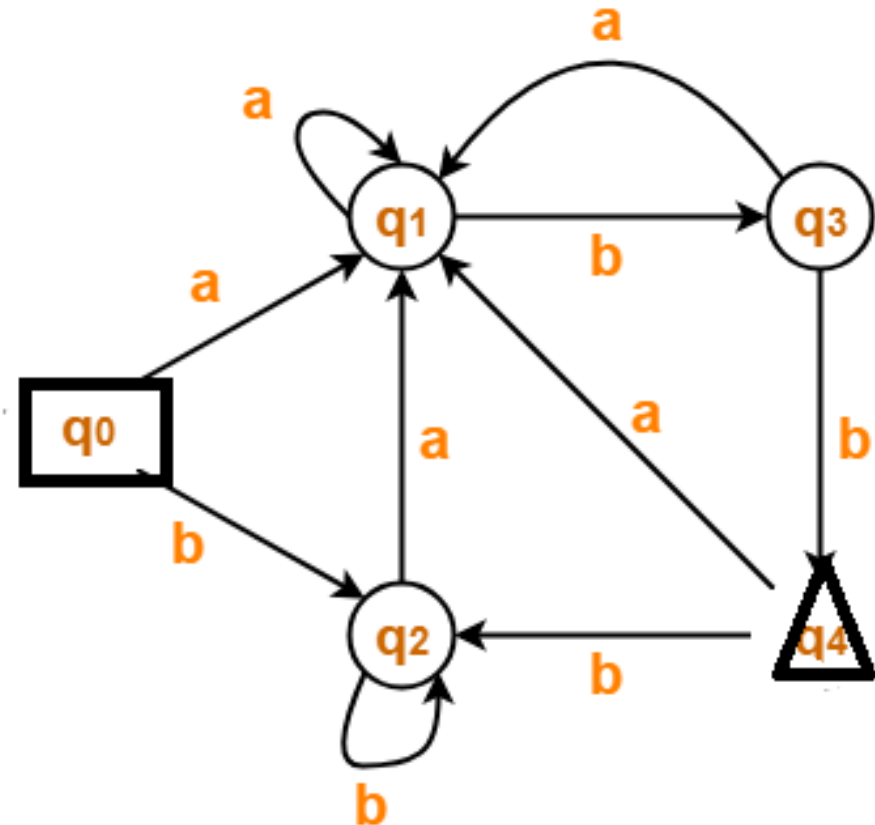
$$P_2 = \{ q_0, q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$$

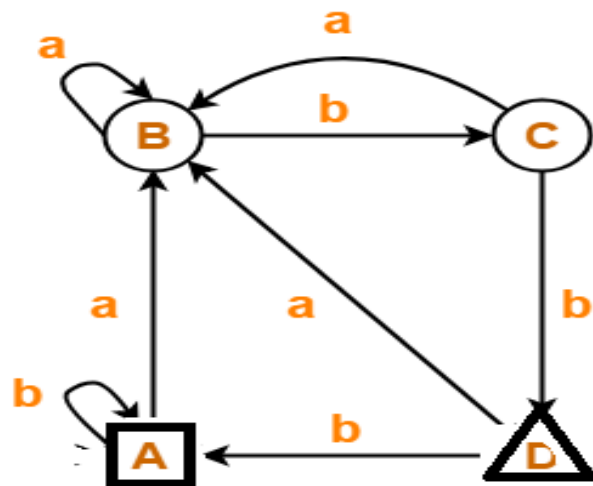
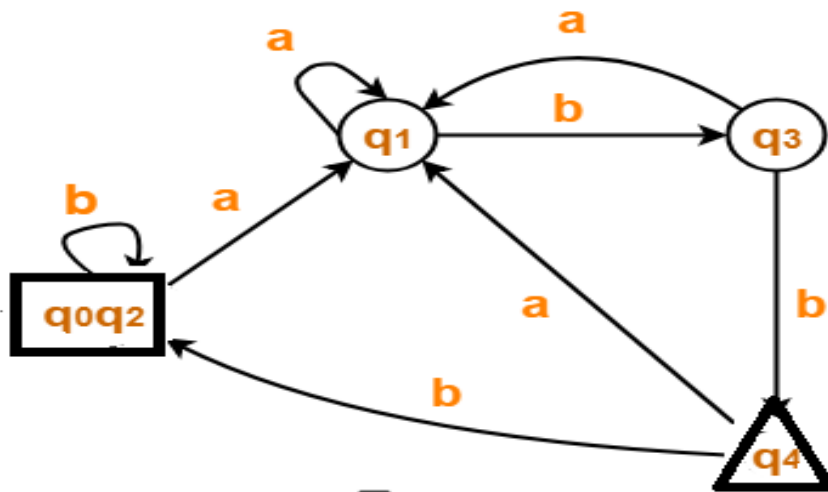
$$P_3 = \{ q_0, q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$$

Since $P_3 = P_2$, so we stop.

From P_3 , we infer that states **q_0** and **q_2** are equivalent and can be merged together.

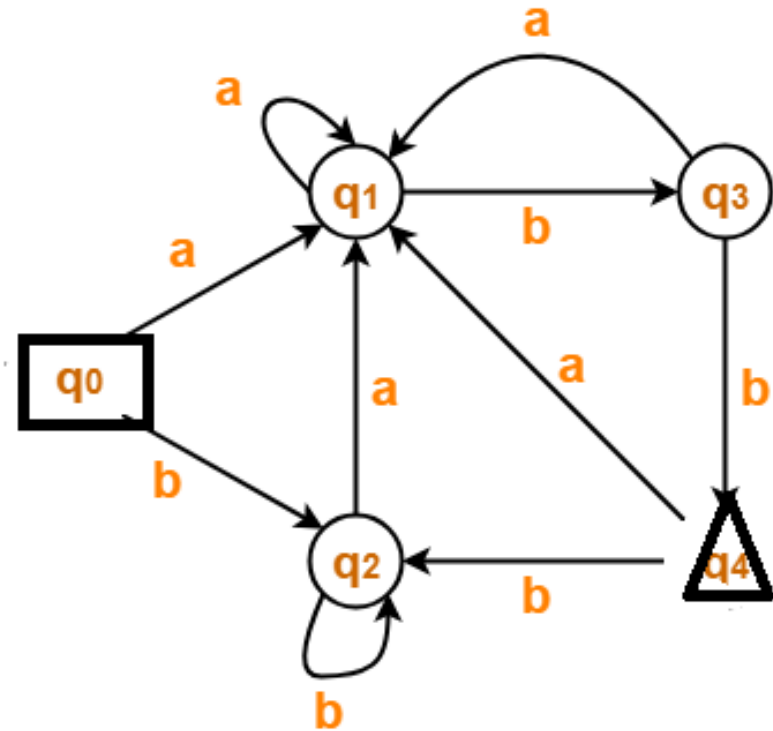
So, Our minimal DFSA is:



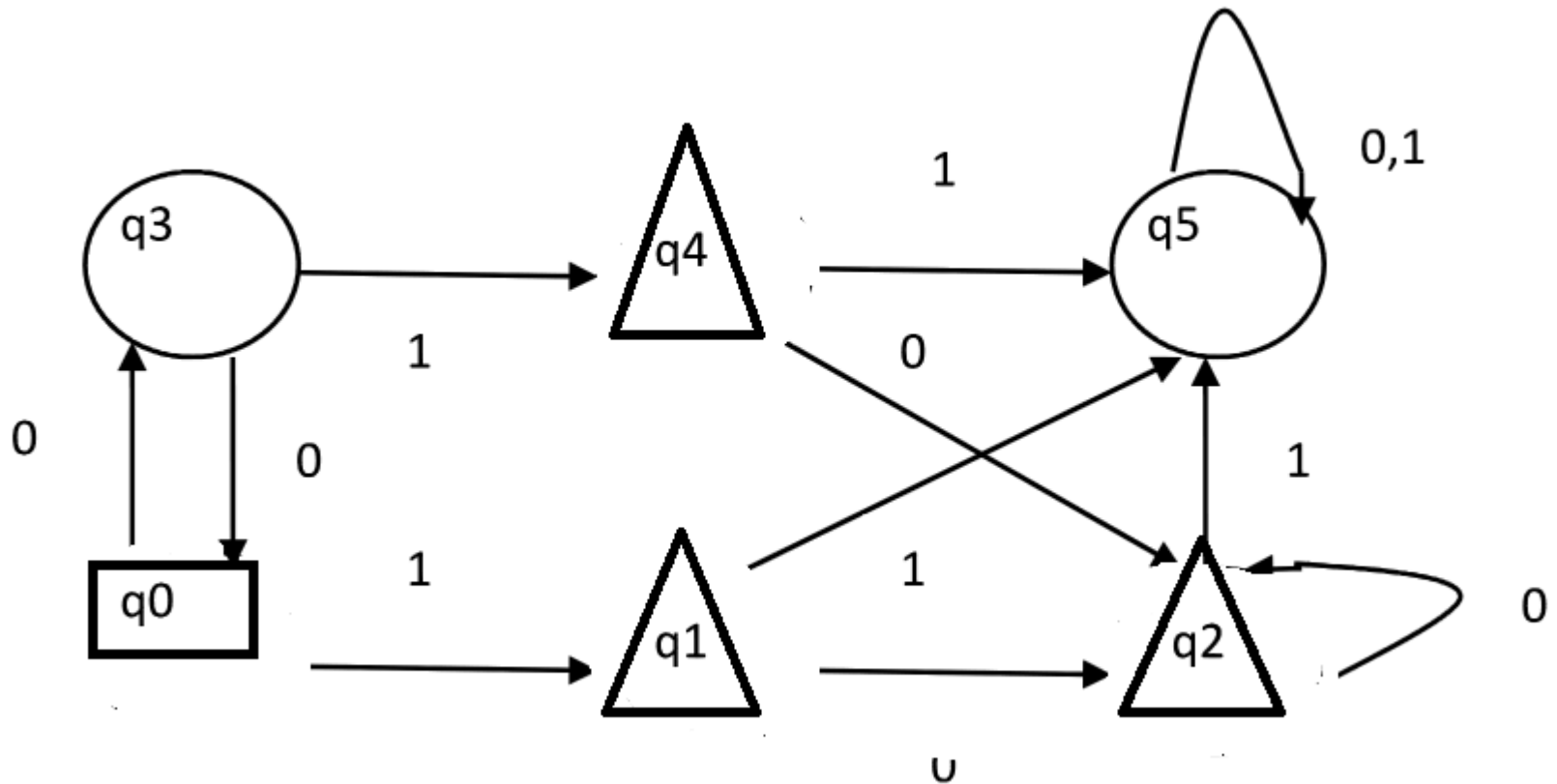


DFSA minimal

.



DFSA non minimal



DFSA non minimal

Chapter 5:

Algebraic languages

Chapter 5: Algebraic languages

5.1 Introduction

Type 2 languages, also called **algebraic languages** or **context-free languages**.

These languages are accepted by abstract machines similar to FSA called push-down automata.

This machine uses a memory called a stack.

- Grammar transformation
- FNC form
- FNG form
- Push-down automata
- Grammar ---- automata

Chapter 5: Algebraic languages

5.2 Review of Algebraic Grammars and Languages

Definition of a Type 2 Grammar:

A grammar $G=(V_t, V_n, S, R)$ is said to be context-free (algebraic or Type 2) if and only if all of its production rules are in the form:

$$A \rightarrow W \quad \text{with} \quad A \in V_n \quad \text{and} \quad W \in (V_t \cup V_n)^*.$$

Definition of Type 2 Languages (context-free or algebraic):

These are the languages generated by Type 2 grammars.

Note:

The set of regular languages is included in the set of algebraic languages.

Chapter 5: Algebraic languages

5.2 Review of Algebraic Grammars and Languages

Converse:

Every regular language is also context-free, but the converse is false.

Notes:

The set of regular languages is included in the set of algebraic languages.

A context-free language:

- is not recognized by a finite state automata
- is not described by a regular expression
- there is no regular grammar to generate it.

Chapter 5: Algebraic languages

5.2 Review of Algebraic Grammars and Languages

Syntax Tree:

Given the use of only one nonterminal symbol on the left-hand side of the production rules in context-free grammars, it is always possible to construct a derivation tree for any generated word. Let $G=(V_t, V_n, S, R)$ be a grammar and let $\omega \in L(G)$.

A syntax tree associated with ω is constructed as follows:

- The root of the tree is labeled with the axiom
- Intermediate (internal) nodes contain nonterminals
- Leaves are terminals
- Reading from left to right the leaves of the tree reconstructs the word to which the tree is associated.

Chapter 5: Algebraic languages

Syntax Tree:

Let the grammar $G=(V_t, V_n, ND, R)$

$V_t=\{0,1,2,3,4,5,6,7,8,9,+,-,.,10\}$

$V_n=\{ND, S, E, P, F, C\}$

Axiom = ND

$R= (ND \rightarrow S E P E F$

$E \rightarrow C E / C$

$C \rightarrow 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9$

$P \rightarrow .$

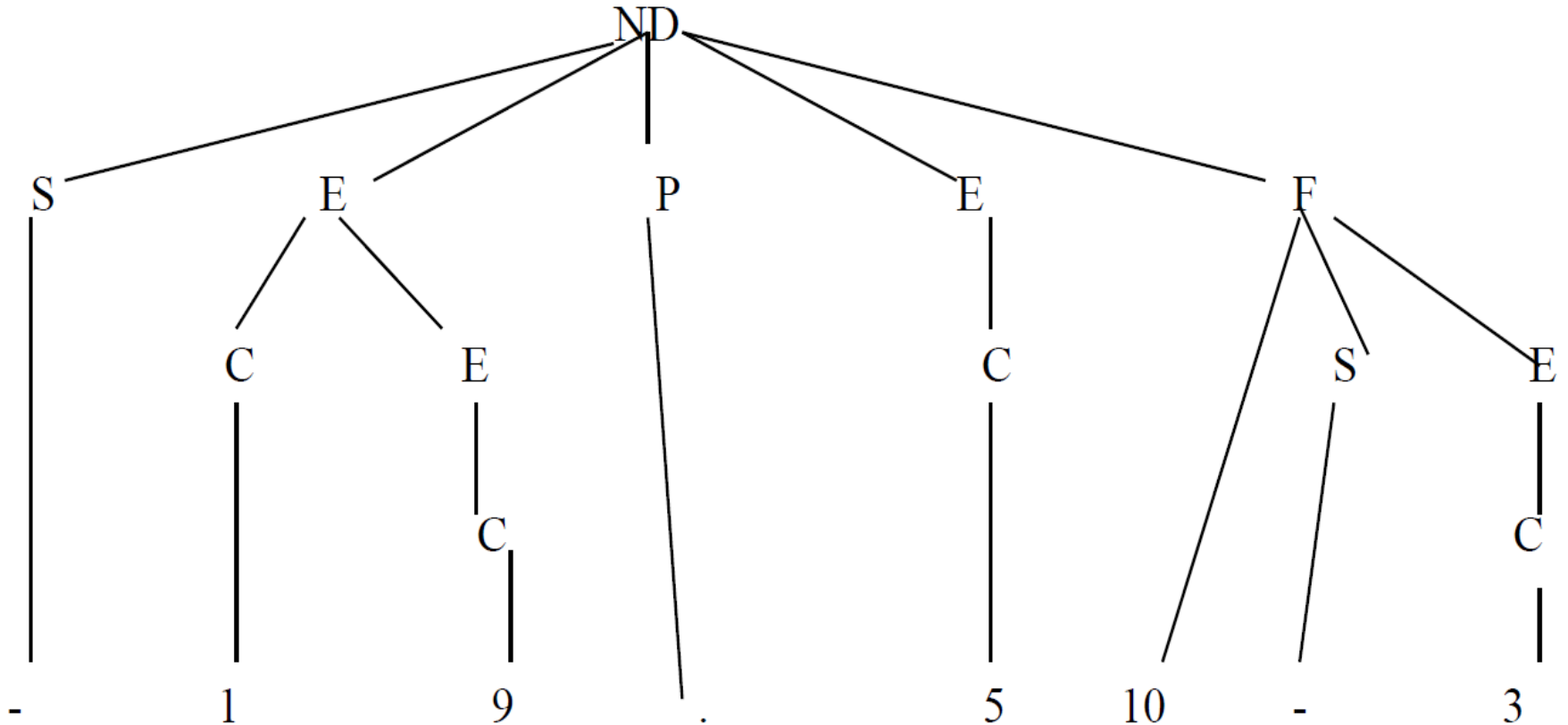
$F \rightarrow 10 S E$

$S \rightarrow + / -)$

Example: Build the syntax tree for the word.: $\omega = - 19.5 10^{-3}$

Chapter 5: Algebraic languages

A decimal number is defined by a **derivation tree**



Chapter 5: Algebraic languages

5.2 Review of Algebraic Grammars and Languages

Definition of an ambiguous word:

A word ω is said to be **ambiguous** if and only if there exist two different derivation trees associated with it, using the **leftmost derivation**.

Definition of an ambiguous grammar:

A grammar G is said to be **ambiguous** if and only if there exists at least one ambiguous word belonging to $L(G)$.

Notes:

- 1- Some languages can be generated by both ambiguous and unambiguous grammars.
- 2- There is no algorithm that can find an unambiguous grammar (if it exists) that generates a language.

Chapter 5: Algebraic languages

5.2 Review of Algebraic Grammars and Languages

Example: Let G be the grammar that has the following production rules:

$$R = (S \rightarrow S \wedge S \quad (1)$$

$$S \rightarrow S \vee S \quad (2)$$

$$S \rightarrow S \Rightarrow S \quad (3)$$

$$S \rightarrow S \Leftrightarrow S \quad (4)$$

$$S \rightarrow \neg S \quad (5)$$

$$S \rightarrow q \quad (6)$$

$$S \rightarrow p \quad (7) \quad)$$

$$G = (V_t, V_n, S, R)$$

$$V_t = \{ \wedge, \vee, \Rightarrow, \Leftrightarrow, \neg, q, p \}$$

$$V_n = \{ S \}$$

Question: Show that G is ambiguous.

Chapter 5: Algebraic languages

5.2 Review of Algebraic Grammars and Languages

The word $p \wedge q \Leftrightarrow p$ is ambiguous, because there exist two different derivations that allow us to generate it.

$$S \xrightarrow{(4)} S \Leftrightarrow S \xrightarrow{(1)} S \wedge S \Leftrightarrow S \xrightarrow{(7)} p \wedge S \Leftrightarrow S \xrightarrow{(6)} p \wedge q \Leftrightarrow S \xrightarrow{(7)} p \wedge q \Leftrightarrow p$$

$$4 \rightarrow 1 \rightarrow 7 \rightarrow 6 \rightarrow 7$$

$$S \xrightarrow{(1)} S \wedge S \xrightarrow{(6)} p \wedge S \xrightarrow{(4)} p \wedge S \Leftrightarrow S \xrightarrow{(6)} p \wedge q \Leftrightarrow S \xrightarrow{(7)} p \wedge q \Leftrightarrow p$$

$$1 \rightarrow 6 \rightarrow 4 \rightarrow 6 \rightarrow 7$$

We have two different paths, and therefore the grammar **G** is **ambiguous**.

Chapter 5: Algebraic languages

5.3 Reduced Grammar

Let $G=(V_t, V_n, S, R)$ be a context-free grammar.

Productive (useful) and unproductive (useless) symbols:

A non-terminal $A \in V_n$ is said to be **useful** if and only if there exists $\omega \in V_t^*$ such that $A \Rightarrow^* \omega$

A non-terminal $A \in V_n$ is said to be **useless** if and only if $\forall \omega \in V_t^*$, **there is no** indirect derivation such that $A \Rightarrow^* \omega$

Chapter 5: Algebraic languages

5.3 Reduced Grammar

Let $G(V_t, V_n, S, R)$ be a context-free grammar.

Accessible and inaccessible symbols:

A non-terminal $A \in V_n$ is said to be **accessible** if and only if there exists

$\alpha \in (V_t \cup V_n)^*$ such that $S \Rightarrow^* \alpha$ and **A appears in α**

A non-terminal $A \in V_n$ is said to be **inaccessible** if and only if $\forall \alpha \in (V_t \cup V_n)^*$,
if $S \Rightarrow \alpha$ then **A does not appear in α .**

Chapter 5: Algebraic languages

5.3 Reduced Grammar

Remark:

Production rules (derivations) that contain useless and inaccessible non-terminals can be **removed** without any impact on the language generated by the grammar.

Unit production ($A \rightarrow B$)

A production rule of the form $A \rightarrow B$ where $A, B \in V_n$ is called a **unit production**.

Remark:

To remove the unit production $A \rightarrow B$, we simply add all the productions of B to the production rules of A . This removal may result in the appearance of other unit productions, so a recursive algorithm should be applied.

Chapter 5: Algebraic languages

5.3 Reduced Grammar

A grammar is said to be **reduced** if and only if all the non-terminals in its production rules **are accessible and productive (useful)**.

5.4 Clean grammar

A grammar is said to be **clean** if and only if:

1. It is **reduced**
2. It does not contain **unit productions**
3. Only the **axiom can generate ϵ** , with the condition that it does not appear in any right-hand side of the rules.

Chapter 5: Algebraic languages

5.4 Clean grammar

To eliminate ε -productions (**empty word**),:

- We must first determine the set of non-terminals that can derive to ε (directly or indirectly);
- then, we modify the productions containing these non-terminals, in such a way as to replace in all the left-hand sides of the productions the nullable symbols by the empty word, in all **possible ways**.

Chapter 5: Algebraic languages

5.4 Clean grammar

Exercise 1: Let G be the context-free grammar $G(V_t, V_n, S, R)$.

$R = (\quad S \rightarrow AB \mid EaE$

$\quad A \rightarrow Aa \mid aB$

$\quad B \rightarrow bB \mid aA$

$\quad C \rightarrow AB \mid aS$

$\quad E \rightarrow D$

$\quad D \rightarrow dD \mid \varepsilon \quad)$

$V_t = \{a, b, d\}$

$V_n = \{S, A, B, D, E\}$

S : Axiome

- 1. Find the language generated by G .**
- 2. Transform G into a reduced grammar.**
- 3. Transform G into a proper grammar.**
- 4. Verify the language found in question 1.**

Chapter 5: Algebraic languages

Solution:

1- $L(G) = \{d^n a d^m \mid n, m \geq 0\}$ we use only the symbol E

2- The non-terminal C is not reachable, A and B are not productive, so we remove the rules containing A , B or C and we get the following reduced grammar:

$$S \rightarrow EaE$$
$$E \rightarrow D$$
$$D \rightarrow dD \mid \varepsilon$$

Chapter 5: Algebraic languages

3- The grammar has a unit production $E \rightarrow D$, we remove it and replace all the E's with D's and we get the following grammar:

$$S \rightarrow EaE \quad E \rightarrow dD \mid \varepsilon \quad D \rightarrow dD \mid \varepsilon$$

The grammar is still not proper due to the rules $D \rightarrow \varepsilon$, $E \rightarrow \varepsilon$. Therefore, we eliminate them and for each D that appears on the right-hand side of a rule, we create another rule.

We obtain the following proper grammar:

$$G' = (\{a, d\}, \{S, D, E\}, S, R')$$

$$R' = (S \rightarrow EaE \mid aE \mid Ea \mid a \quad E \rightarrow dD \mid d \quad D \rightarrow dD \mid d)$$

4- The language found is the same, but it is easier to find using the proper grammar.

Chapter 5: Algebraic languages

Exercise 2:

Reduce the following grammar : $G = (\{a,b\}, \{S, A, B, C, D\}, S, R)$

$$R = (S \rightarrow aAB \mid bC \mid aab \quad A \rightarrow aA \mid aB \mid a \quad C \rightarrow bb \quad D \rightarrow b)$$

Solution: All non-terminals are useful, derive towards terminal strings except for the symbol B. We remove all rules containing B.

$$G' = (\{a,b\}, \{S, A, C, D\}, S, R')$$

$$R' = (S \rightarrow bC \mid aab \quad A \rightarrow aA \mid a \quad C \rightarrow bb \quad D \rightarrow b)$$

It can be noticed that the symbol D and A are unreachable from the axiom S.

$$G'' = (\{a,b\}, \{S, A, C\}, S, R'') \text{ reduced grammar}$$

$$R'' = (S \rightarrow bC \mid aab \quad C \rightarrow bb)$$

Chapter 5: Algebraic languages

Exercise 3 :

Reduce the following grammar : $G = (\{a, b\}, \{S, A, B, C\}, S, R)$

$R = (S \rightarrow aaAb \quad A \rightarrow bA \mid a \quad B \rightarrow bB \mid b \quad C \rightarrow a)$

Solution:

All non-terminals are useful, they derive towards terminal strings except for symbol B, however symbols B and C are inaccessible from the axiom S.

We remove all rules containing B and C.

$G' = (\{a, b\}, \{S, A\}, S, R')$ reduced grammar.

$R' = (S \rightarrow aaAb \quad A \rightarrow bA \mid a)$

Note: If the axiom is useless, the language generated by this grammar is empty

Chapter 5: Algebraic languages

Exercise 4 : Find a clean grammar for the following grammar:

$$G = (\{a, b, c\}, \{S, A, B, C, D\}, S, R)$$

$$R = (S \rightarrow aSA \mid aS \mid ab \mid aSb \mid BC \quad A \rightarrow SS \mid S \mid c \quad B \rightarrow D \quad C \rightarrow b \\ D \rightarrow a \mid C)$$

Solution:

- All non-terminals are useful, they derive towards terminal strings.
- All symbols are reachable from the axiom S.
- No rule contains the empty word.
- We have 3 rules of the form $(A \rightarrow B, A, B \in V_n)$

$$A \rightarrow S \quad B \rightarrow D \quad \text{et} \quad D \rightarrow C$$

Chapter 5: Algebraic languages

We replace the rule $A \rightarrow S$ par : $A \rightarrow aSA \mid aS \mid ab \mid aSb \mid BC$

We replace the rule $D \rightarrow C$ par : $D \rightarrow b$

We replace the rule $B \rightarrow D$ par : $B \rightarrow a \mid b$

$G' = (\{a, b, c\}, \{S, A, B, C, D\}, S, R')$

$R' = (S \rightarrow aSA \mid aS \mid ab \mid aSb \mid BC$

$A \rightarrow SS \mid c \mid aSA \mid aS \mid ab \mid aSb \mid BC$

$B \rightarrow a \mid b$

$C \rightarrow b$

$D \rightarrow b)$

We notice that the symbol D becomes unreachable, so we remove the last rule.

Chapter 5: Algebraic languages

$$G'' = (\{a, b, c\}, \{S, A, B, C\}, S, R'')$$

$$R'' = (S \rightarrow aSA \mid aS \mid ab \mid aSb \mid BC$$

$$A \rightarrow SS \mid c \mid aSA \mid aS \mid ab \mid aSb \mid BC$$

$$B \rightarrow a \mid b$$

$$C \rightarrow b)$$

The grammar G'' is clean.

Chapter 5: Algebraic languages

Exercise 5: Remove the empty word from the following grammar and reduce it:

$G = (\{a, b, c\}, \{S, A, B, D, E\}, S, R)$

$R = (S \rightarrow BE \mid AaBD \mid BS \quad A \rightarrow cB \mid \varepsilon \quad B \rightarrow bc \mid AA \quad D \rightarrow cc$
 $E \rightarrow b \mid \varepsilon)$

Solution: The symbols A, E, B, S derive directly or indirectly to the empty word (ε), so we remove these rules and add other rules:

$R' = (S \rightarrow BE \mid AaBD \mid BS \quad A \rightarrow cB \quad B \rightarrow bc \mid AA \quad D \rightarrow cc \quad E \rightarrow b$

plus these rules

$S \rightarrow B \mid E \mid aBD \mid AaD \mid aD \mid B \mid S$

$A \rightarrow c \quad B \rightarrow A)$

Chapter 5: Algebraic languages

$R' = (S \rightarrow BE \mid AaBD \mid BS \mid B \mid E \mid aBD \mid AaD \mid aD$

$A \rightarrow cB \mid c \quad B \rightarrow bc \mid AA \mid A \quad D \rightarrow cc \quad E \rightarrow b)$

The grammar is reduced (without useless and unreachable symbols), but contains rules of the form $(A \rightarrow B)$

$R'' = (S \rightarrow BE \mid AaBD \mid BS \mid bc \mid AA \mid cB \mid c \mid b \mid aBD \mid AaD \mid aD$

$A \rightarrow cB \mid c$

$B \rightarrow bc \mid AA \mid cB \mid c$

$D \rightarrow cc$

$E \rightarrow b)$

Remark: compare the two grammars (initial and the clean)

Chapter 5: Algebraic languages

5.3 Chomsky Normal Form FNC (Noam Chomsky)

A grammar $G=(V_t, V_n, S, R)$ is said to be in Chomsky Normal Form (FNC) if and only if all its derivation rules are in the form:

$A \rightarrow BC$ or $A \rightarrow a$ with $A, B, C \in V_n$ and $a \in V_t$

For any algebraic (context-free) grammar, there exists an equivalent grammar in Chomsky Normal Form.

The practical advantage of FNC is that derivation trees are binary trees, which facilitates the application of tree exploration algorithms.



Noam Chomsky born in 1928

Chapter 5: Algebraic languages

5.3 Transformation of a grammar into Chomsky normal form.

To obtain a Chomsky normal form grammar equivalent to an algebraic grammar G , the following steps are required:

1. Transform the grammar into a clean grammar
2. For each terminal a , introduce the non-terminal C_a , then add the rule $C_a \rightarrow a$
3. For each rule $A \rightarrow \alpha$, with $|\alpha| \leq 2$, replace each terminal with the non-terminal associated with it;
4. For each rule $A \rightarrow \beta$, with $|\beta| \geq 3$, ($\beta = \beta_1\beta_2...\beta_n$), create the non-terminals D_i , then replace the rule with the following rules: $A \rightarrow \beta_1D_1$, $D_1 \rightarrow \beta_2D_2$, ..., $D_{n-2} \rightarrow \beta_{n-1}D_{n-1}$, $D_{n-1} \rightarrow \beta_n$, where $D_i = \beta_{i+1}\beta_{i+2}...\beta_n$, with i varying from 1 to $n - 2$.

Chapter 5: Algebraic languages

5.3 Example of transformation into FNC

$$G = (\{a, b, c\}, \{S, A, B, D\}, S, R)$$

$$R = (S \rightarrow aSB \quad (1) \quad S \rightarrow DcBb \quad (2) \quad A \rightarrow bc \quad (3) \quad B \rightarrow aAb \quad (4)$$

$$D \rightarrow b \quad (5))$$

G : is clean

$$(1) \quad S \rightarrow aSB \quad : \quad R' = (\quad S \rightarrow X_1X_2 \quad X_1 \rightarrow a \quad X_2 \rightarrow SB$$

$$(2) \quad S \rightarrow DcBb \quad : \quad S \rightarrow X_3X_4 \quad X_3 \rightarrow DX_5 \quad X_5 \rightarrow c \quad X_4 \rightarrow BX_6 \quad X_6 \rightarrow b$$

$$(3) \quad A \rightarrow bc \quad : \quad A \rightarrow X_6X_5$$

$$(4) \quad B \rightarrow aAb \quad : \quad B \rightarrow X_1X_7 \quad X_7 \rightarrow AX_6$$

$$(5) \quad D \rightarrow b \quad : \quad D \rightarrow b \quad)$$

$$G' = (\{a, b, c\}, \{S, A, B, D, X_1, X_2, X_3, X_4, X_5, X_6, X_7\}, S, R')$$

Chapter 5: Algebraic languages

5.4 Left recursion:

A grammar is left-recursive if it contains a production of the form:

$$B \rightarrow B w \quad \text{with } B \in V_n \quad \text{and} \quad w \in (V_n \cup V_t)^*$$

- Left recursion causes infinite loops...

By eliminating it, we can transform it into right-recursion, which is not problematic.

- Let's consider a subset of productions:

$$B \rightarrow B \alpha \quad B \rightarrow \beta \quad \text{with} \quad B \in V_n, \quad \alpha, \beta \in (V_n \cup V_t)^*$$

and β doesn't start with $B \rightarrow$ It generates the language: $\beta \alpha^*$

- This language is also generated by:

$$B \rightarrow \beta \mid \beta Z$$

$$Z \rightarrow \alpha \mid \alpha Z$$

where $Z \notin (V_n \cup V_t)^*$.

Chapter 5: Algebraic languages

5.4 Left recursion (example)

$G = (\{a, b, c\}, \{S, A, B, D\}, S, R)$

$R = (\begin{array}{l} S \rightarrow aS \quad (1) \quad S \rightarrow BA \quad (2) \quad A \rightarrow AbD \quad (3) \quad A \rightarrow b \quad (4) \quad B \rightarrow bc \\ (5) \quad D \rightarrow BA \quad (6) \quad D \rightarrow Da \quad (7) \quad D \rightarrow b \quad (8) \end{array})$

We have two left-recursive rules. $A \rightarrow AbD \quad (3)$ et $D \rightarrow DA \quad (7)$

The A-rules will be replaced by the 4 rules:

$A \rightarrow AbD \quad (3) \quad A \rightarrow b \quad (4) : \quad A \rightarrow \mathbf{b} \quad A \rightarrow \mathbf{bZ} \quad Z \rightarrow \mathbf{bD} \quad Z \rightarrow \mathbf{bDZ}$

The D-rules will be replaced by the 6 rules:

$D \rightarrow BA \quad (6) \quad D \rightarrow DA \quad (7) \quad D \rightarrow b \quad (8) : \quad D \rightarrow \mathbf{BA} \quad D \rightarrow \mathbf{b} \quad D \rightarrow \mathbf{BAZ}_1$
 $D \rightarrow \mathbf{bZ}_1 \quad Z_1 \rightarrow \mathbf{a} \quad Z_1 \rightarrow \mathbf{aZ}_1$

Chapter 5: Algebraic languages

5.4 Left recursion (example continuation)

The grammar without left recursion. $G' = (\{a, b, c\}, \{S, A, B, D, Z_1, Z_2\}, S, R')$

$R' = ($

$$\begin{array}{l} S \rightarrow aS \quad S \rightarrow BA \\ A \rightarrow b \quad A \rightarrow bZ \quad Z \rightarrow bD \quad Z \rightarrow bDZ \\ B \rightarrow bc \\ D \rightarrow BA \quad D \rightarrow b \quad D \rightarrow BAZ_1 \quad D \rightarrow bZ_1 \quad Z_1 \rightarrow a \quad Z_1 \rightarrow aZ_1 \end{array}$$

$)$

Chapter 5: Algebraic languages

5.5 Greibach normal form (FNG)

An algebraic grammar is in Greibach normal form (FNG) if and only if all its production rules are in the form:

$$A \rightarrow x\alpha \quad \text{or} \quad S \rightarrow \varepsilon, \text{ with } x \in V_t, \alpha \in V_n^*$$

and S is the axiom.



Sheila Adele Greibach born in 1939
New York

Chapter 5: Algebraic languages

5.5 Greibach normal form (FNG)

Proposition :

For any algebraic grammar G_1 , there exists an equivalent grammar G_2 in Greibach normal form such that $L(G_1) = L(G_2)$.

The practical interest of FNG is that at each derivation, we can determine a longer and longer prefix consisting only of terminal symbols. This makes it easier to construct stack automata from the grammars, and therefore, syntax analyzers can be easily implemented.

Chapter 5: Algebraic languages

5.5 Greibach normal form (FNG)

Let G be a type 2 grammar.

Step1: Transform the grammar into Chomsky Normal Form (FNC) and eliminate left-recursive rules.

$A \rightarrow BC$ or $A \rightarrow a$ with $A, B, C \in V_n$ and $a \in V_t$

With an (arbitrary) order on the non-terminals: $V_n = \{ A_1, A_2, A_3, \dots, A_m \}$

$|V_n| = m$

Step2: Modify the rules so that they satisfy the following condition (C):

$A_i \rightarrow A_j \omega$ with $j > i$ and $\omega \in V_n^+$ (C)

Step3: Transform the rules into FNG starting with the non-terminal that has the highest order..

Chapter 5: Algebraic languages

5.5 Greibach normal form (FNG) (example)

Transform the following algebraic grammar into FNG: $G=(\{a,b,c,d\}, \{S, A,B\} , S, R)$

$$R=(\quad S \rightarrow cABdc \quad (1) \quad A \rightarrow Bb \quad (2) \quad A \rightarrow aA \quad (3) \quad B \rightarrow Bd \quad (4) \\ B \rightarrow a \quad (5) \quad)$$

1- Eliminate the left-recursion:

$$\mathbf{B} \rightarrow \mathbf{B}d \quad (4) \quad B \rightarrow a \quad (5) : \quad \mathbf{B} \rightarrow \mathbf{a} \quad \mathbf{B} \rightarrow \mathbf{a} \mathbf{Z} \quad \mathbf{Z} \rightarrow \mathbf{d} \quad \mathbf{Z} \rightarrow \mathbf{d} \mathbf{Z}$$

$$V_n' = \{A,B,C,D,Z\}$$

$$R' = (S \rightarrow cABdc \quad A \rightarrow Bb \quad A \rightarrow aA \quad B \rightarrow a \quad B \rightarrow aZ \quad Z \rightarrow d \\ Z \rightarrow dZ \quad)$$

Chapter 5: Algebraic languages

5.5 Greibach normal form (FNG) (example)

2- Transform into Chomsky Normal Form (FNC)

$$R'' = (S \rightarrow X_1 X_2$$

$$X_7 \rightarrow b$$

$$X_1 \rightarrow c$$

$$B \rightarrow a$$

$$X_2 \rightarrow A X_3$$

$$B \rightarrow X_6 Z$$

$$X_3 \rightarrow B X_4$$

$$Z \rightarrow d$$

$$X_4 \rightarrow X_5 X_1$$

$$Z \rightarrow X_5 Z \quad)$$

$$X_5 \rightarrow d$$

Ordonner les nouveaux non terminaux:

$$A \rightarrow X_6 A$$

$$V_{n''} = \{S, A, B, Z, X_1, X_2, X_3, X_4, X_5, X_6, X_7\}$$

$$X_6 \rightarrow a$$

$$\text{Ordre} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$$

$$A \rightarrow B X_7$$

Chapter 5: Algebraic languages

5.5 Greibach normal form (FNG) (example)

3- Scheduling.

$R'' = (S \rightarrow X_1 X_2$	(C)	$X_7 \rightarrow b$	(FNG)
$X_1 \rightarrow c$	(FNG)	$B \rightarrow a$	(FNG)
$X_2 \rightarrow A X_3$	(non C)	$B \rightarrow X_6 Z$	(C)
$X_3 \rightarrow B X_4$	(non C)	$Z \rightarrow d$	(FNG)
$X_4 \rightarrow X_5 X_1$	(C)	$Z \rightarrow X_5 Z$	(C)
$X_5 \rightarrow d$	(FNG)		
$A \rightarrow X_6 A$	(C)		
$X_6 \rightarrow a$	(FNG)		
$A \rightarrow B X_7$	(C)		

Chapter 5: Algebraic languages

5.5 Greibach normal form (FNG) (example)

3- Scheduling

$X_2 \rightarrow AX_3$ (non C) replaced by : $X_2 \rightarrow X_6A X_3$ (C)
 $X_2 \rightarrow BX_7 X_3$ (non C)

$X_2 \rightarrow BX_7 X_3$ (non C) replaced by : $X_2 \rightarrow aX_7 X_3$ (FNG)
 $X_2 \rightarrow X_6ZX_7 X_3$ (C)

$X_3 \rightarrow BX_4$ (non C) replaced by : $X_3 \rightarrow aX_4$ (FNG)
 $X_3 \rightarrow X_6ZX_4$ (C)

Chapter 5: Algebraic languages

5.5 Greibach normal form (FNG) (example)

4- FNG: We start with the non-terminals of higher priority. (11,10,....,1)

$$\begin{aligned}
 R''' = (& \quad X_7 \rightarrow b \quad \quad X_6 \rightarrow a \quad \quad X_5 \rightarrow d \quad \quad X_4 \rightarrow dX_1 \\
 & X_3 \rightarrow aX_4 \\
 & \quad X_3 \rightarrow aZX_4 \quad X_2 \rightarrow aAX_3 \quad X_2 \rightarrow aX_7X_3 \quad X_2 \rightarrow aZX_7X_3 \\
 & \quad X_1 \rightarrow c \quad \quad Z \rightarrow d \quad \quad Z \rightarrow dZ \quad \quad B \rightarrow a \quad \quad B \rightarrow aZ \\
 & \quad A \rightarrow aA \quad \quad A \rightarrow aX_7 \quad \quad A \rightarrow aZX_7 \quad \quad S \rightarrow cX_2 \quad)
 \end{aligned}$$

Chapter 5: Algebraic languages

5.5 Greibach normal form (FNG) (example)

5- Verify the reduction of the new grammar.

The symbols B and X_5 are unreachable

$$R''' = (\begin{array}{llll} X_7 \rightarrow b & X_6 \rightarrow a & X_4 \rightarrow dX_1 & X_3 \rightarrow aX_4 \\ X_3 \rightarrow aZX_4 & X_2 \rightarrow aAX_3 & X_2 \rightarrow aX_7X_3 & X_2 \rightarrow aZX_7X_3 \\ X_1 \rightarrow c & Z \rightarrow d & Z \rightarrow dZ & A \rightarrow aA \\ A \rightarrow aX_7 & A \rightarrow aZX_7 & S \rightarrow cX_2 &) \end{array}$$

$$V_n''' = \{ S, A, Z, X_1, X_2, X_3, X_4, X_6, X_7 \}$$

Chapter 5: Algebraic languages

5.6 Other normal forms

Quadratic normal form:

A grammar is in Greibach's quadratic normal form if all its rules are of the form

$A \rightarrow aV$ where V is composed of at most two non-terminals,

$$S \rightarrow aSS \mid b$$

Backus-Naur Form (BNF)

is a notation used to describe the syntax rules of programming languages. It was designed by John Backus and Peter Naur when creating the grammar for the Algol 60 language.

Chapter 5: Algebraic languages

5.6 Autres Formes normales

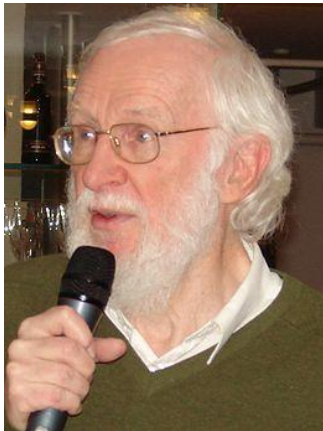
Backus-Naur Form (BNF) :

Let's take an example that defines the structure of the "if" statement in the C programming language:

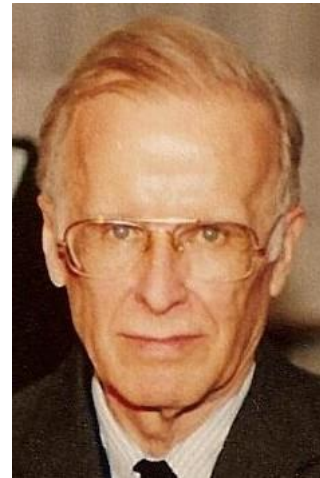
`<structure_if> ::= if "(" <condition> ")" "{" <instructions> "}"`

`<structure_if>`, `<condition>` and `<instructions>` : are non-terminals..

`::=` is a meta-symbol meaning "is defined by". `if`, `"("`, `)"`, `"{"` and `"}"` are terminals.



né en 1928 à Frederiksberg (Danemark)

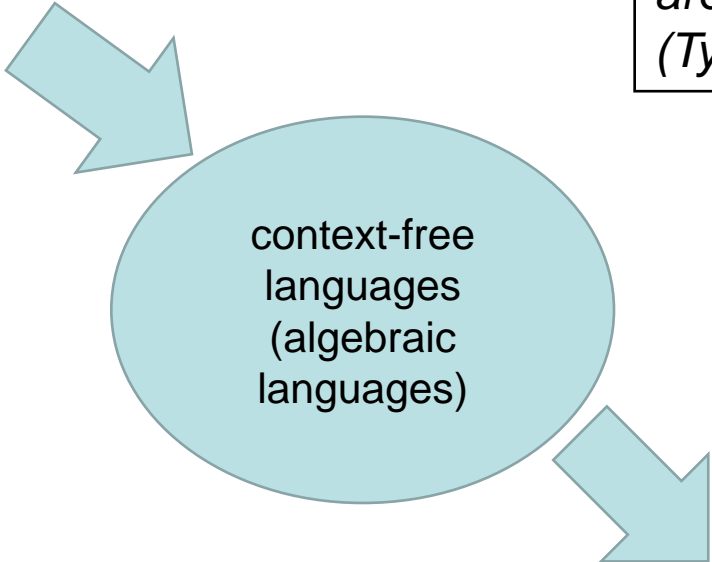


(né en 1924 à Philadelphie)

Chapter 5: Algebraic languages

Context-free grammars
generate...

Like FSAs, pushdown automata (PDA) are abstract machines that determine whether a word belongs to a language or not. The languages recognized by PDAs are the context-free languages (Type 2).



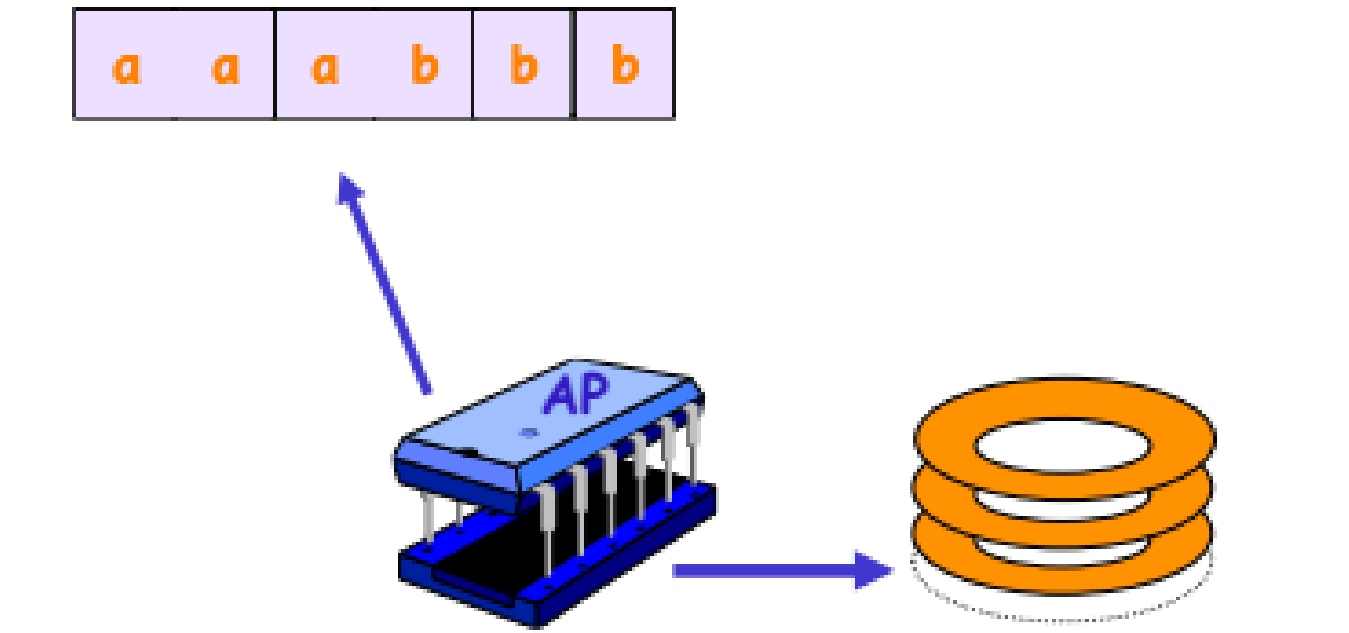
context-free
languages
(algebraic
languages)

which are recognized by pushdown
automata.

Chapter 5: Algebraic languages

5.7 Pushdown automata

The pushdown automaton PDA will attempt to read the word. **aaabbb** :



Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

Why a stack?

- Finite automata (FSA) have no memory other than their states
- They cannot "count" beyond their number of states
- A stack provides additional unbounded memory
- The stack is accessed only from its top
- The number of symbols used in the stack is finite
- An empty stack can be a criterion for accepting words.

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

Preliminary example:

The steps for recognizing the language $\{a^n b^n, n \geq 1\}$ by a PDA could be the following:

Read the a's, push them onto the stack and stay in the same state.

Upon encountering the first b, pop an a from the stack and change state.

For each subsequent b encountered, pop an a from the stack.

If the a's in the stack are exhausted at the same time as all the b's have been read, then the word belongs to the language.

Chapitre 5: Les langages algébriques

5.7 pushdown automata :

Preliminary example:

The steps for recognizing the language $\{a^n b^n, n \geq 1\}$ by a PDA could be the following:

1. *Read the a's, push them onto the stack and remain in the same state.*
2. *Upon encountering the first b, pop an a from the stack and change state.*
3. *For each subsequent b encountered, pop an a from the stack.*
4. *If the a's in the stack are exhausted at the same time as all the b's have been read, then the word belongs to the language.*

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

Formal definition:

A PDA is formally defined by a septuple $(V_t, W, Q, q_0, Z_0, f, F)$ where :

- V_t is the input alphabet, finite and non-empty
- W is the stack vocabulary, finite and non-empty
- Q is the set of states, finite and non-empty
- q_0 is the initial state that belongs to Q
- Z_0 is the initial symbol of the stack (bottom of the stack), $Z_0 \in W$
- F is the set of final (acceptance) states, $F \subset Q$
- f is the transition function.

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

Formal definition:

f is defined as : $Q \times (V \cup \{\varepsilon\}) \times (W \cup \{\varepsilon\}) \rightarrow Q \times W^*$

Operation:

A transition rule $f(q, a, p) = (q', \chi)$ considers:

- the current state q of the automata
- the character read a on the tape (or maybe not: ε)
- the symbol p on the top of the stack (or maybe not: ε)

A rule indicates:

- the next state q' of the automata
- the sequence of symbols χ to be pushed on top of the stack in place of the top symbol.

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

$$f(q, a, p) = (q', \chi)$$

- 1- if $|\chi| = 1 \Rightarrow f(q, a, A) = (q', B) : \text{replace } A \text{ by } B$
- 2- if $|\chi| = 2 \Rightarrow f(q, a, A) = (q', BA) : \text{push on } B$
- 3- if $|\chi| = 0 \Rightarrow f(q, a, A) = (q', \varepsilon) : \text{pop } A$
- 4- if $a = \varepsilon \Rightarrow f(q, \varepsilon, A) = (q', \chi) : \text{we do not change the input symbol.}$
- 5- if $a = \varepsilon$ and $\chi = \varepsilon \Rightarrow f(q, \varepsilon, A) = (q', \varepsilon) : \text{Emptying the contents of the stack.}$
- 6- A word belongs to the language if it is **fully read** and **the automata is in a final state** or **the stack is empty.**

Chapitre 5: Les langages algébriques

5.7 Les automates à pile: *Recognition modes*:

There are 2 recognition modes for PDAs depending on whether F equals \emptyset or not:

- *Final state recognition*
- *Empty stack recognition*
- **Final state recognition:**

$L_F(A) = \{ w \in Vt^*, \text{ the stack contains } Z_0 \text{ Initially, there is a reading of } w \text{ from } q_0 \text{ to } q_f, q_f \in F \}$

- **Empty stack recognition**

$L_\emptyset(A) = \{ w \in Vt^*, \text{ the stack contains } Z_0 \text{ Initially, there is a reading of } w \text{ from } q_0 \text{ which ends with an empty stack} \}$

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : *Initial configuration*:

A configuration of the PDA at a certain moment, is given by the content of the stack, the current state of the PDA and the remaining word to read:
(stack content, current state, word remaining to read).

The initial configuration is (Z_0, q_0, ω) , where q_0 is the initial state of the PDA and ω is the word submitted to A (to be recognized).

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : *Equivalence between PDAs.*

- The final state recognition mode is equivalent to the empty stack recognition mode.
- A PDA recognizes one and only one language, but the same language can be recognized by multiple PDAs.
- We say that two PDAs A_1 and A_2 are equivalent if and only if they recognize the same language, $L(A_1) = L(A_2)$.

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

Example: Construct the PDA that recognizes the following language.:

$$L(A) = \{ w / w \in V_t^* \text{ and } w = x c x^R \text{ tel que } x \in \{ 0,1 \}^* \} \quad V_t = \{ 0,1,c \}$$

(x^R means "x reversed" or "x mirrored".)

example of recognized words:

0110c0110 , c, 0c0, 1c1 , 00c00, 11c11

Solution 1: Empty stack recognition $F = \phi$

$$A = (V_t, W, Q, q_0, Z_0, f, \phi)$$

$$V_t = \{ 0,1,c \} \quad W = \{ Z_0, A, B \} \quad Q = \{ q_0, q_1, q_2 \}$$

We push an A for the symbol 0 and a B for the symbol 1.

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

$f(q_0, c, Z_0) = (q_0, \varepsilon)$: Pop Z_0 , stop with an empty stack

$f(q_0, 0, Z_0) = (q_1, AZ_0)$: push A and modify the state

$f(q_0, 1, Z_0) = (q_1, BZ_0)$: push B and modify the state

 $f(q_1, 0, A) = (q_1, AA)$: push A *Push loop*

$f(q_1, 1, A) = (q_1, BA)$: push B *Push loop*

$f(q_1, 1, B) = (q_1, BB)$: push B *Push loop*

$f(q_1, 0, B) = (q_1, AB)$: push A *Push loop*

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

$f(q_1, c, A) = (q_2, A)$: Replace A with A, move to c, and change the state.

$f(q_1, c, B) = (q_2, B)$: Replace B with B, move to c, and change the state.

$f(q_2, 0, A) = (q_2, \varepsilon)$: Pop A in case of symbol match.

$f(q_2, 1, B) = (q_2, \varepsilon)$: Pop B in case of symbol match.

$f(q_2, \varepsilon), Z_0) = (q_2, \varepsilon)$: pop Z_0 empty stack stop the automata

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

Example: Construct the PDA that recognizes the following language:

$$L(A) = \{ w \mid w \in Vt^* \text{ and } w = x c x^R \text{ such as } x \in \{ 0,1 \}^* \} \quad Vt = \{ 0,1,c \}$$

(x^R x mirrored)

example of recognized words:

0110c0110 , c, 0c0, 1c1 , 00c00, 11c11

Solution 2: Final state recognition $F \neq \phi$

$$A = (Vt, W, Q, q_0, Z_0, f, F)$$

$$Vt = \{ 0,1,c \} \quad W = \{ Z_0, A, B \} \quad Q = \{ q_0, q_1, q_2, q_3 \}, \quad F = \{ q_3 \}$$

We push an A for the symbol 0 and a B for the symbol 1.

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

$f(q_0, c, Z_0) = (q_3, \varepsilon)$: Pop Z_0 stop with q_3 final state

$f(q_0, 0, Z_0) = (q_1, AZ_0)$: push A and modify the state

$f(q_0, 1, Z_0) = (q_1, BZ_0)$: push B and modify the state

$f(q_1, 0, A) = (q_1, AA)$: push A *Push loop*

$f(q_1, 1, A) = (q_1, BA)$: push B *Push loop*

$f(q_1, 1, B) = (q_1, BB)$: push B *Push loop*

$f(q_1, 0, B) = (q_1, AB)$: push A *Push loop*

Chapitre 5: Les langages algébriques

5.7 pushdown automata :

$f(q_1, c, A) = (q_2, A)$: Replace A with A, move to c, and change the state

$f(q_1, c, B) = (q_2, B)$: Replace B with B, move to c, and change the state

$f(q_2, 0, A) = (q_2, \epsilon)$: Pop A *in case of symbol match*

$f(q_2, 1, B) = (q_2, \epsilon)$: Pop B *in case of symbol match*

$f(q_2, \epsilon, Z_0) = (q_3, \epsilon)$: Pop Z_0 , q_3 *final state* *stop the automata*

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : *Deterministic and non-deterministic PDA*

There are two cases of non-determinism for Pushdown Automata (PDA):

1- For the same stack state, same state, and the same input symbol, there exist at least two transitions:

$$f(q_1, a, A) = (q_1, \chi_1)$$

$$f(q_1, a, A) = (q_2, \chi_2)$$

2. For the same stack state and same state, we have the choice to read or not to read from the tape.

$$f(q_1, a, A) = (q_1, \chi_1)$$

$$f(q_1, \varepsilon, A) = (q_1, \chi_1)$$

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : *Remarks*

- There exist algebraic languages for which there is no deterministic pushdown automaton (DPDA) that recognizes them.
- If a language L is recognized by a DPDA, then there exists an unambiguous algebraic grammar that generates L .
- For each algebraic language L , there exists a pushdown automaton (PDA) A such that $L(A)=L$.
- The languages recognized by PDAs are algebraic languages.

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

Example: Construct the PDA that recognizes the following language:

$$L(A) = \{ w / w \in V_t^* \text{ et } w = x x^R \text{ such as } x \in \{ 0,1 \}^* \} \quad V_t = \{ 0,1 \}$$

example of recognized words:

01100110 , 00, 11 , 0000, 1111

Solution 1: Finite-state pushdown automaton $F \neq \emptyset$

$$A = (V_t, W, Q, q_0, Z_0, f, F)$$

$$V_t = \{ 0,1 \} \quad W = \{ Z_0, A, B \} \quad Q = \{ q_0, q_1, q_2, q_3 \}, \quad F = \{ q_0, q_2 \}$$

We push an A onto the stack for the symbol 0 and a B for the symbol 1.

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

$f(q_0, 0, Z_0) = (q_0, AZ_0)$: push A

$f(q_0, 1, Z_0) = (q_0, BZ_0)$: push B

$f(q_0, 0, A) = (q_0, AA)$: push B *Stacking loop of symbols from x.*
 $= (q_1, \varepsilon)$: pop A *and change the state, the first symbol of x^R ,*

$f(q_0, 1, A) = (q_0, BA)$: push B *Push loop*

$f(q_0, 0, B) = (q_0, AB)$: push B *Push loop*

$f(q_0, 1, B) = (q_0, BB)$: push A *Push loop*
 $= (q_1, \varepsilon)$: pop A *and change the state, the first symbol of x^R ,*

Chapitre 5: Les langages algébriques

5.7 Pushdown automata :

$f(q_1, 0, A) = (q_1, \varepsilon)$: pop A *in case of symbol match*

$f(q_1, 1, B) = (q_1, \varepsilon)$: pop B *in case of symbol match*

$f(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$: pop Z_0 , q_2 final state *stop the automata*

Note: This is a non-deterministic automata, we cannot know the first symbol of x^R .

When we have two consecutive equal symbols (00 or 11), we have to do two operations, either push or pop.

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : Grammar and PDA

- For each algebraic language L , there exists a pushdown automata (PDA) A such that $L(A)=L$.
- The languages recognized by PDAs are algebraic languages.

Method: grammar PDA

For any algebraic grammar $G = (V_t, V_n, S, R)$, there exists a pushdown automata (PDA) that recognizes the language generated by G .

$A = (V_t, W, Q, q_0, Z_0, f, F)$ such as $L(G)=L(A)$

Given an algebraic grammar G , the goal is to find a pushdown automaton (PDA) A that recognizes the language $L(G)$ with an empty stack. ($F = \phi$)

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : Grammar PDA

1. First, we transform G into Greibach normal form.
2. Then, we define the parameters of the PDA as follows:

$$G = (V_t, V_n, S, R) \quad \rightarrow \quad A = (V_t', W, Q, q_0, Z_0, f, F)$$

$$V_t' = V_t$$

$$W = V_n$$

$$Q = \{q_0\};$$

$$Z_0 = S;$$

$$B \rightarrow aA_1A_2\dots A_n \quad \rightarrow \quad f(q_0, a, B) = (q_0, A_n \dots A_2A_1)$$

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : Example grammar PDA

$$G = (V_t, V_n, S, R) \rightarrow V_t = \{a, b, c\} \quad V_n = \{S, A, B\}$$

$$R = (S \rightarrow aSA \quad S \rightarrow bSB \quad S \rightarrow c \quad A \rightarrow a \quad B \rightarrow b)$$

We want to construct the equivalent pushdown automaton with an empty stack.

$$A = (V_t', W, Q, q_0, Z_0, f, F)$$

$$V_t' = V_t \quad W = V_n \quad Q = \{q_0\} \quad Z_0 = S \quad F = \emptyset$$

$$S \rightarrow aSA \rightarrow f(q_0, a, S) = (q_0, AS)$$

$$S \rightarrow bSB \rightarrow f(q_0, b, S) = (q_0, BS)$$

$$S \rightarrow c \rightarrow f(q_0, c, S) = (q_0, \varepsilon)$$

$$A \rightarrow a \rightarrow f(q_0, a, A) = (q_0, \varepsilon)$$

$$B \rightarrow b \rightarrow f(q_0, b, B) = (q_0, \varepsilon)$$

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : Grammar PDA

Method: from PDA to grammar

For every PDA $A = (V_t, W, Q, q_0, Z_0, f, F)$ There exists an algebraic grammar $G = (V_t, V_n, S, R)$ such as $L(G) = L(A)$

Let the pushdown automata (PDA) $A = (V_t, W, Q, q_0, Z_0, f, \phi)$ given:

We construct the grammar $G = (V_t', V_n, S, R)$ as follows

1- $V_t' = V_t$

2- $V_n = \{ [q, A, p] \mid \forall q \in Q \text{ and } \forall p \in Q \text{ and } \forall A \in W \cup \{S\} \}$

3- $R =$ 3 types of rules

Chapitre 5: Les langages algébriques

5.7 Pushdown automata le : Grammar PDA

3- R = 3 types of rules:

a) $S \rightarrow [q_0, Z_0, q] \quad \forall q \in Q$ as many arrows as there are states in A

b) if $f(q, a, A) = (q_1, B_1 B_2 \dots B_m)$ is a transition then

construct the rule:

$$[q, A, p] \rightarrow a [q_1, B_1, q_2] [q_2, B_2, q_3] \dots [q_m, B_m, q_{m+1}]$$

$$p = q_{m+1}, \quad q_1, q_2, \dots, q_{m+1} \in Q$$

c) if $f(q, a, A) = (p, \varepsilon)$ is a transition then construct

$$[q, A, p] \rightarrow a$$

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : Example Grammar PDA

Consider the pushdown automata accepting the following language:

$L = \{ w \mid w \in \{0, 1\}^*, w \text{ contains an equal number of 0's and 1's} \}$

$A = (V_t, W, Q, q_0, Z_0, f, \phi)$

$V_t = \{0, 1\}$ $W = \{Z_0, A, B\}$ $Q = \{q_0\}$,

Transition functions f :

1) $f(q_0, 0, Z_0) = (q_0, AZ_0)$ 2) $f(q_0, 1, Z_0) = (q_0, BZ_0)$

3) $f(q_0, 0, A) = (q_0, AA)$ 4) $f(q_0, 1, B) = (q_0, BB)$

5) $f(q_0, 1, A) = (q_0, \varepsilon)$ 6) $f(q_0, 0, B) = (q_0, \varepsilon)$

7) $f(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : Example Grammar PDA

The equivalent grammar is as follows: $G = (V_t, V_n, S, R)$

$$V_n = \{S\} \cup \{[q_0, Z_0, q_0], [q_0, A, q_0], [q_0, B, q_0]\}$$

$|Q| = 1$ one arrow

$$\rightarrow S \rightarrow [q_0, Z_0, q_0]$$

$$1) f(q_0, 0, Z_0) = (q_0, AZ_0) \rightarrow [q_0, Z_0, q_0] \rightarrow 0 [q_0, A, q_0] [q_0, Z_0, q_0]$$

$$2) f(q_0, 1, Z_0) = (q_0, BZ_0) \rightarrow [q_0, Z_0, q_0] \rightarrow 1 [q_0, B, q_0] [q_0, Z_0, q_0]$$

$$3) f(q_0, 0, A) = (q_0, AA) \rightarrow [q_0, A, q_0] \rightarrow 0 [q_0, A, q_0] [q_0, A, q_0]$$

$$4) f(q_0, 1, B) = (q_0, BB) \rightarrow [q_0, B, q_0] \rightarrow 1 [q_0, B, q_0] [q_0, B, q_0]$$

$$5) f(q_0, 1, A) = (q_0, \varepsilon) \rightarrow [q_0, A, q_0] \rightarrow 1$$

$$6) f(q_0, 0, B) = (q_0, \varepsilon) \rightarrow [q_0, B, q_0] \rightarrow 0$$

$$7) f(q_0, \varepsilon, Z_0) = (q_0, \varepsilon) \rightarrow [q_0, Z_0, q_0] \rightarrow \varepsilon$$

Formal languages

Chapitre 5: Les langages algébriques

5.7 Pushdown automata : Example Grammar PDA

The equivalent grammar is as follows: $G = (V_t, V_n, S, R)$

$$V_n = \{S\} \cup \{[q_0, Z_0, q_0], [q_0, A, q_0], [q_0, B, q_0]\} = \{S, X, Y, Z\}$$

$$V_t = \{0, 1\}$$

$$R = (S \rightarrow X$$

$$X \rightarrow \varepsilon / 0YX / 1ZX$$

$$Y \rightarrow 1 / 1YY$$

$$Z \rightarrow 0 / 1ZZ)$$