Theory of Computation

Dr Mohamed BAHACHE

January 30, 2026

Computer Science Department

Contents

List of Tables

List of Figures

Chapter 1

Mathematical concepts(Sets,Relations,...)

We start with some mathematical notions that we will use in our module theory of computation , such as sets, relations, applications,. . . .

1.1 Sets

Définition:

A set is a group (collection) of objects, represented as an unit, each object called element.

- We denote by Φ to the empty set.
- The cardinality of a finite set A is the number of elements that contains; denoted by $|A|$.

1.1.1 Set specification

We can specify a set by several methods:

- listing all their elements, separated by commas and included in braces.

  Example: $E = \{White, Black, Blue\}$. , for specifying the set of colors : blue, black and yellow, we say that: $Blue \in E$, but $Yellow \notin E$. The repetition of elements in a set is useless, but if we take into account the number of appearances, in this case we call the group a **multi-set** ,instead of a set.

- formally write the rule verified by its elements.

  Example:$E = \{x | x = 2k + 1, k \in N\}$,to designate the set of odd natural numbers

1.1.2 Set operators

- **Inclusion:** Let E, F two sets if each element of E is also an element of F, and we write $E \subseteq F$. we say that E is a proper subset of F ($E \subsetneq F$) , if ($E \subseteq F$ and $E \neq F$).
- **Intersection:** $E \cap F = \{x / x \in E \, and \, x \in F\}$
- **Union:**$E \cup F = \{x / x \in E \, or \, x \in F\}$
- **Complement:**$\bar{E} = \{x / x \notin E\}$

1.1.3 Sequences and Tuples

A sequence of objects is a list of these objects in some order. We usually designate a sequence by writing the list within parentheses, as (1,2,3), the order is important unlike sets:(1,2,3) and (2,1,3),are different.

- Finite sequences often are called **tuples**.
- A sequence with k elements is a **k-tuple**
- 2-tuple is also called an **ordered pair**.

1.1.4 Cartesian product

Let A and B two sets $\Rightarrow$ the Cartesian product of A and B

$$A \times B = \{(a, b) / a \in A \wedge b \in B\}$$

1.2 Relations

Definition: A relation between two sets A and B is a subset of the Cartesian product $A \times B$

1.2.1 Few relation types

Let R be a relation defined on the set A.

- identity Relation :$R = \{(x, y) / x = y\}$
- Total relation :$R = A \times A$
- inverse Relation : $R^{-1} = \{(y, y) / (x, y) \in R\}$

1.2.2 Relation properties

Let $R \in A \times A$ be a relation and $\forall\, x, y, z \in A$, R is :

- Reflexive : if $\forall x \in A, (x, x) \in R$
- Symmetric: $(x, y) \in R \Rightarrow (y, x) \in R$
- Anti-symmetric: if $(x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$

- Transitive: si $(x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

$\Rightarrow$ if R is (reflexive and transitive) then R is a **pre-order**
$\Rightarrow$a symmetric preorder is an **equivalence**
$\Rightarrow$a preorder antisymmetric is an **order**.

1.3 Monoïd

let E be a set provided with an internal composition law. If the law is associative and admits a neutral element then we say that E has a **monoid** structure.

1.4 Homomorphism

Let E and F be two sets provided respectively with the internal composition laws $*$ et Δ si $f(x * y) = f(x) \Delta f(y)$ then we say that f is a **homomorphism** from E to F.

1.5 Function

A function is a mathematical notion, establishes a relationship with inputs and outputs see figure(1.1).

Fig. 1.1: Principle of a function

. we write: $f(a) = b$ formally we write: $f : D \to R$

- D : The set of possible inputs to the function is called its **domain**,
- R : The set of possible outputs of a function come from a set called its **range**.

If the domain of f est : $A_1 \times A_2 \times ... \times A_k$ then the elements of f are **k-tuple**$(a_1, a_2, ..., a_k)$ (a_i): (argument number i).

Function with k arguments is called k-ary function , k is the arity. If $k = 2 \Rightarrow f$ is binary function.

1.6 Graphs

Definition: A graph G=(V, E) is a pair where :

- V is a set of **vertices** (or points or nodes)
- E is a set of **edges**(or lines)
- A graph with p vertices and q edges is called a **(p, q)** graph.
- Edge joining vertex to itself is called a **loop**.
- A graph may be **directed** or **undirected**, as shown by figure(1.2)

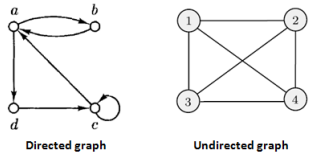

Fig. 1.2: Example of a graph

.

1.6.1 Terminology

- A graph H = (V1, X1) is called a **sub graph** of G (V, X) if $V1 \subseteq V$. and $X1 \subseteq X$.
- A graph H = (V1, X1) is called a **partial graph** of G (V, X) if V1= V. and $X1 \subseteq X$.
- A **path** in a graph is a sequence of vertices connected by edges.
- A graph is **connected** if every pair of vertices have a path between them.
- A path where the start vertex and the end vertex are identical is called a **cycle/-circuit**.see figure(1.3)

.

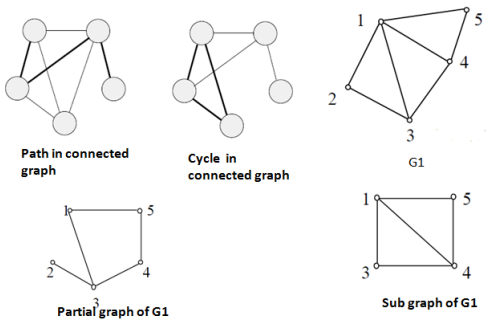

Fig. 1.3: Terminology of graph

1.6.2 Tree

Definition:A tree is an acyclic connected graph.See figure(1.4)

Theorem 1 *The following assertion are equivalents for a graph T:*

- *T is a tree*
- *Any two vertices of T are linked by a unique path in T.*
- *T is minimally connected, T is connected but (T-e is disconnected for every edge* $e \in T$.
- *T is maximally acyclic, T contains no cycle bat* $T+xy$ *does, for any two no-adjacent vertices* $x, y \in T$

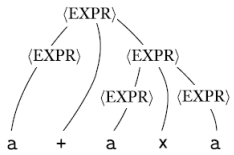

Fig. 1.4: Example of tree

.

1.7 Mathematical proofs

- *Theorems, proofs, and definitions are essential components of every mathematical subject.*
- *These three notions represent the heart and the spirit of the mathematics.*
- *We do proofs to confirm the correctness of statements.*
- *Proofs are achieved through steeps using entities that are confirmed true such as, theorems, lemmas, propositions, corollaries.*

- *The only way to determine the truth or falsity of a mathematical statement is with a mathematical proof.*
- *A proof is a convincing logical argument that a statement is true.*
- *the proof may contain within it several different sub proofs.*

1.7.1 Types of proofs

Several types of arguments used in mathematical proofs. Here, we Present three types have relation and may be used in the theory of computation.

1. ***Proof by construction:*** *Usually we need to proof that a specific type on an object is exists, one way to proof that is by demonstrate how we can construct this object, this is what s called proof by construction.*
2. ***Proof by induction:*** *Proof by induction is an advanced method used to show that all elements of an infinite set have a specified property.*
3. ***Proof by contradiction:*** *we assume that the theorem is false and then show that this assumption yields a contradiction.*

Chapter 2

Alphabets,Words and Languages

the building blocs fothe the theory of computation, are presented in this chapter, such as alphabets,words and language.

2.1 Alphabets

Alphabets are the basic concept for thr theory of computation set of elementary entities).

Definition(Alphabet):

An alphabet, denoted by A, is a finite set (not empty) of symbols (letters,digits,operators,common characters,...). Note: these symbols are used to generate lexical entities(tokens).

Example

- $A_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $A_2 = \{a, b, c, ..., z\}$
- $A_3 = \{if, then, else, id, =, nb\}$

2.2 The word (string)

Definition: *A word defined on the set of alphabets A, is a finite string from elements of A.*

Example

- *The word 2023 on alphabet* A_1
- *The word thl on* A_2
- *The word* $id = nb$ *on* A_3

2.3 Operation on words

2.3.1 Length of a word

Definition: *The length of a word u over an alphabets A, denoted by* $|u|$*,is the number of*

symbols that contains.u.

Définition(Empty word): *The empty word denoted by ε, is defined on all alphabets A. knowing that:* $|\varepsilon| = 0$

Definition:

The set of words having length greater than or equals to 1 constructed on the alphabet A, is denoted by A^+.

Definition: *The set of words can be constructed on an alphabet set A,including the empty word is denoted by:* $A^* = A^+ \cup \{\varepsilon\}$

Then we can define the function f as follow :

$f : A^* \to N$.

Definition: *We denote by* $|u|_s$ *, the number of appearances of a symbols in the word u.*

Example: Let the alphabet :$A_2 = \{a, b, c, ..., z\}$ *and the word* $u = langage$

Then : $|u| = 7$,$|u|_a = 2$, $|u|_c = 0$

2.3.2 Mirror of word

The mirror of a wrd v is denoted by w^R *or* $\overline{w}$*, obtained by inverting the symbols of v. Example:* $(acac)^R$=*caca*

Definition: *Let u and v be two words defined on the alphabet set A. the concatenation of u and v, written uv, is the string obtained by appending v to the end of u.*

Example: Let u=abcd, and v =aacc then uv=abcdaacc.

- *We denote by* u^n*(the word u is concatenated with itself n times).*
- *the concatenation operation is not commutative.*$(u.v \neq v.u)$
- $u^0 = \varepsilon, u^n = u.u^{n-1}, n \geq 1$.
- $\forall u \in A^*, u^n.u^m = u^{n+m}$
- $\forall u \in A^*, (u^m)^n = u^{mn}$
- $\forall u \in A^* u.\varepsilon = \varepsilon.u = u$
- $\forall u, v, w \in A^* u.(v.w) = (u.v).w$
- $\mid u.v \mid = \mid u \mid + \mid v \mid$

Proposition: *Let A be a set of alphabets,the structure$(A^*, ., \varepsilon)$ is a monoid.*

2.3.3 Factorization of word

Let A be an alphabet set , and u, v two words on A^*. u is a factor of v iff: $\exists w_1, w_2 \in A^*$ such sucg as :$v = w_1 u w_2$

- *if $w_1 = \varepsilon$, then u is a left factor(prefixe) of v*
- *si $w_2 = \varepsilon$, then u is a right factor(suffixe) of v*
- *We say that u is a proper prefix of v if $\exists w \in A^+$ sucg as: $v = u.w$*
- *We say that u is a proper suffix of v if $\exists w \in A^+$ such as :$v = w.u$*

Example: let u=abba be a word on the alphabet set A=a,b, the following sub strings: $\varepsilon, a, ab, abb, abba$ are prefixes of u, and $\varepsilon, a, ba, bba, abba$ are suffixes of u.

Levi's Lemma

Let A be an alphabet and let $a, b, c, d \in A^*$ be (four words) such as : $ab = cd$.

It exist three states define the relation between a,b,c and d :

- *if $|a| < |c|$ then $\exists w \in A^+$ such as $c = aw$ and $b = wd$*
- *if $|a| = |c|$ then $a = c$ and $b = d$*
- *if $|a| > |c|$ then $\exists w \in A^+$ such as $a = cw$ and $d = wb$*

2.4 Quotient of word

2.4.1 Right quotient

The right quotient of a word u by the word v denoted by uv^{-1} or $u_{/v}$

- *w if u=wv*
- *not defined if v not suffix of u*

Example:

$abcde(cde)^{-1} = ab$

We use the same fashion to define the left quotient.

2.5 Languages

Definition: *Let A an alphabet ; a language L defined on this set of alphabets A, is a set of words (strings) that can be built by a specific rule. Soit $A = \{0, 1\}$, on peut définir le langage L=$\{00, 01, 10, 11\}$ sur A.*

- *The empty language* $L = \Phi$
- *the language that contains the empty word.* $L = \{\varepsilon\}$.
- $A^0 = \{\varepsilon\}$.
- $A^1 = A$ *(the language where the words is the alphabet symbols.*$\mid u \mid = 1$*).*
- $A^+ = A^1 \cup A^2 \cup ...$
- $A^* = A^0 \cup A^1 \cup ...$
- *Then a language L is a subset of* A^* $(L \subset A^*)$

2.5.1 Operations on languages

Let L_1 and L_2 defined respectively on A_1 and A_1 , then:

- *the union of L_1 and L_2 is the language defined on $A_1 \cup A_2$, contains all words hold either to L_1, or to L_2:*
  $L_1 \cup L_2 = \{u / u \in L_1 \, or \, u \in L_2\}$.
- *intersection of L_1 and L_2 is the language defined on $A_1 \cap A_2$ Contains all words that hold to L_1 and L_1 in the same time.* $L_1 \cap L_2 = \{u \in A^* / u \in L_1 \wedge u \in L_2\}$
- *concatenation:* $L_1.L_2 = \{u.v \in A^* / u \in L_1 \wedge v \in L_2\}$. *(The concatenation is not commutative).*
- *the complement:* $\bar{L} = \{u \in A^* / u \notin L\}$.
- *difference:* $L1 - L2 = \{u \in A^* / u \in L1 \wedge u \notin L2\}$
- *Example: Let* $X = \{a, b\}$, *and* $L1 = \{aa, ab, ba\}$ *and* $L2 = \{ab, bb\}$ *then:*
  1. $L1 \cap L2 = \{ab\}$
  2. $L1 \cup L2 = L1 + L2 = \{aa, ab, ba, bb\}$.
  3. $L1.L2 = \{aaab, aabb, abab, abbb, baab, babb\}$
  4. $L2.L1 = \{abaa, abab, abba, bbaa, bbab, bbba\}$
- *Iteration:*
  1. $L^i = L.L^{i-1}, i \geq 2$.
  2. *Star of language:* $L^* = \cup_{i \geq 0} L^i$.

3. $L^0 \neq \Phi$
4. $L^* = L^0 + L^+$
5. $L^+ = L.L^* = L^*.L$

- *let L and R be two languages*

1. $L^* = (L^*)^*$
2. $L^* = L^*.L^*$
3. $(L + R)^* = (L^*.R^*)^*$

2.5.2 Arden's equation

Theorem 2 *Let A and B be two languages related by the following equation:$X = AX + B$, then :*

1. *A^*B is the minimal solution of the equation.*
2. *if $\epsilon \in A$, then this solution is unique.*

Note:this theorem is used to determine regular expression(Language type3).

Chapter **3**

Generator system of language(Grammar)

Here, we present the notion of the grammar as a generator system for languages,or a rewriting system , their component, types,classification,...

3.1 Grammar

Definition:

- *A grammar is a system that generates words in a language, via the application of production (or rewriting) rules.*
- *Formally, We can define a grammar by a quadruplet* $G = (N, T, S, P)$ *tel que :*
  - *T:Set of alphabets of the language generated by G, these symbols are called (terminal symbol set)*
  - *N: Set of no terminal symbols.*
  - *S:* $S \in N$*,the start symbol (axiom).*
  - *P: the set of rewriting rules. each rewriting rule has the form;* $\alpha \to \beta$*, where* $\alpha \in (T \cup N)^+$ *and* $\beta \in (T \cup N)^*$

Example : *Let* $G = (N, T, S, P)$ *be a grammar*

- $T = \{0, 1\}$
- $N = \{S\}$
- $P = (S \to 0S|1S|\varepsilon)$

Remark:

-If the left side for several production rules is the same then we can use factorization,as in the above example.

-There is another representation for these rules called BNF (Backus Normal Form). $\alpha \to \beta$ *is represented by :* $< \alpha >::=< \beta >$

3.2 The derivation in a grammar

3.2.1 The derivation in one step

Let $G = (N, T, P, S)$ be a grammar u is a no empty string $u \in (T \cup N)^+$ v is a string (possibly be empty) $v \in (T \cup N)^*$ the derivation in one step of v from u is denoted by $u \Rightarrow v$) iff:

- *$u = xu'y$ (u can be decomposed in x, u' and y ; x and y may be empty),*
- *$v = xv'y$ (v can be decomposed in x, v' and y),*
- *$u' \to v'$ is a rule in P.*

3.2.2 Successive derivation

the side v can be derived from the side u en successive steps:

- $u \dot{\Rightarrow} v$: if v is obtained from u by 1 or several derivations in one step,

- $u \overset{*}{\Rightarrow} v$: if v is obtained from u by 0, 1 or several in one step.

3.2.3 Generated language by a grammar

The language generated by a gammar $G = (N, T, P, S)$ is the set of words(strings) on the set T that can be derived from S, formally we write:

$$L(G) = \left\{ w \in T^* / S \overset{*}{\Rightarrow} w \right\}$$

Example :

Let $G = (N, T, P, S)$ be a grammar where: $P = (S \to aSb|ab)$

By derivation we get: $S \overset{*}{\Rightarrow} a^n b^n, n > 0$

Note : One grammar can generates one language. One language can be generated by multiple grammars.

3.3 Chomsky hierarchy

Chomsky identifies a hierarchy of grammar classes, each class corresponding to a particular constraint on the form of the production(rewriting) rules.

3.3.1 Grammar of type0

- *No restrictions on rules.*
- *All grammar is therefore type 0.Type 0 is the most general type.*

Theorem 3 *Recursively enumerable languages are languages generated by a type 0 grammar.*

3.3.2 Grammar of type 1

- *All the rules of this grammar are of the form* $\alpha \to \beta$

  $\alpha \in (T \cup N)^+ \ \alpha \in (T \cup N)^* \mid \alpha \mid \leq \mid \beta \mid \ \beta \neq \varepsilon$,*if* $\alpha \neq S$

- *Contextual ,contextual sensitive grammar . The rules are of the form:*

  $uAv \to uwv \ A \in N, u, v \in (T \cup N)^* \ w \in (T \cup N)^+$

  In other words, the non-terminal symbol A is replaced by w if we have the contexts u on the left and v on the right. So all contextual grammar is monotonous because $\beta \neq \varepsilon$.

Theorem 4 *For any language L generated by a monotone grammar G, there exists a contextual grammar generates L.*

Example :

$S \overset{1}{\to} aSQ; S \overset{2}{\to} abc; cQ \overset{3}{\to} Qc; bQc \overset{4}{\to} bbcc;$

1-$S \overset{2}{\Rightarrow} abc$ *2-*$S \overset{1+2+3+4}{\Rightarrow} aabbcc$ *3-*$S \overset{1+1+2+3+4+3+3+4}{\Rightarrow} aaabbbccc$ *the generated language is:* $a^n b^n c^n$

Definition: *We call contextual language (or language sensitive to context, a language generated by a contextual grammar.*

3.3.3 Context free Grammar(type 2)

An additional constraint compared to that imposed for contextual grammars consists of requiring that all contextual productions have an empty context. This leads to the following definition.

Definition: *We call type 2 grammar or a context-free grammar, or algebraic grammar, if all the rules have the form:* $A \to \beta$

$A \in N, \beta \in (T \cup N)^*$

Definition: *We call context-free language (or algebraic language, A language generated by an algebric grammar.*

Example : $G = (T, N, S, P) \ N = \{S\} \ T = \{a, b\} \ P = (S \to aSb|\varepsilon)$

Definition: *A context-free grammar is in Chomsky normal form if each rule has the form:* $A \to BC; A \to a$

$a \in T, and \ A \neq \varepsilon, B \neq \varepsilon$

And with the possibility of $S \to \varepsilon$.

3.3.3.1 Ambiguity

We say that a context-free grammar G is ambiguous if a string can be generated by more than one left derivation tree.

Example:

Let G be the a grammar has the following rules:

$E \rightarrow E + E|E * E|(E)|a$

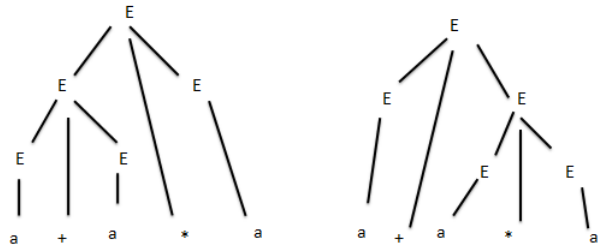

Fig. 3.1: Ambiguous grammar

.

3.3.4 Regular grammar(type 3)

regular grammars adds constraints on the form of the grammatical rule. The rules of a regular grammar can either be of the form of:

$A \rightarrow \alpha B$ *ou* $A \rightarrow \alpha$ *avec* $A, B \in N$ *et* $\alpha \in T^*$ *(Linear on the right)*

Or :

$A \rightarrow B\alpha$ *ou* $A \rightarrow \alpha$ *avec* $A, B \in N$ *et* $\alpha \in T^*$ *(Linear on the left)*

Definition: *We call regular language a language generated by a regular grammar.*

3.3.4.1 K-grammar

The grammar G(N,T,S,P) is a k-grammaire (Kleene's grammar) if the grammatical rules have the form:

- $A \rightarrow xB$ *ou* $A \rightarrow x$ *with* $A, B \in N$ *et* $x \in T$
- *All the rules of the form :* $A \rightarrow \epsilon$ *where* $(A \neq S)$ *are not accepted.*

Transformation of linear on the right grammar to K-grammar

Input : Linear on the right grammar $G\,(N, T, S, P)$

Output : K-grammar $G'\,(N', T', S', P')$.

- *Replace each rule has the form :* $A \to (a_1 a_2 ... a_n)\,B$
  by $A \to a_1 A_1, A_1 \to a_2 A_2, ... A_n \to a_n B$
- *Replace each rule has the form :* $A \to a_1 a_2 ... a_n$
  by $A \to a_1 A_{1;} A_1 \to a_2 A_2; ...; A_n \to a_n$
- *Empty word*
  1. $\epsilon \in L(G) \Rightarrow \varepsilon \in L(G')$
  2. *If* $A \to \varepsilon$ *et* $A \neq S$ *then replace* A *by epsilon in the right side of the rules.*

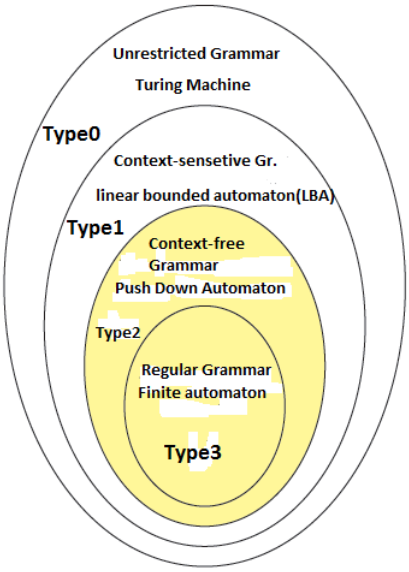

Fig. 3.2: Inclusion relation between formal grammars.

.

$Type3 \subset Type2 \subset Type1 \subset Type0$

Chapter 4

Regular languages and regular expressions

In this chapter we present the regular languages, generated by the regular grammars, and the different operation achieved on this kind of languages, from one side and the regular expressions as a specification tool from another side

4.1 Regular languages

4.1.1 Defintion

A language is regular if and only if there is a regular grammar generating this language

4.1.2 Reminder-Regular grammar(type 3)

The rules of a regular grammar can either be of the form of:

$A \to \alpha B$ *ou* $A \to \alpha$ *avec* $A, B \in N$ *et* $\alpha \in T^*$ *(Linear on the right)*

Or :

$A \to B\alpha$ *ou* $A \to \alpha$ *avec* $A, B \in N$ *et* $\alpha \in T^*$ *(Linear on the left)*

4.1.2.1 K-grammar

The grammar G=(N,T,S,P) is a k-grammaire (Kleene's grammar) if the grammatical rules have the form:

- $A \to xB$ *ou* $A \to x$ *with* $A, B \in N$ *et* $x \in T$
- *All the rules of the form :* $A \to \epsilon$ *where* $(A \neq S)$ *are not accepted.*

Transformation of linear on the right grammar to K-grammar

Input : Linear on the right grammar $G\,(N, T, S, P)$

Output : K-grammar $G'\,(N', T', S', P')$.

- *Replace each rule has the form :* $A \to (a_1 a_2 ... a_n)\, B$
  by $A \to a_1 A_1, A_1 \to a_2 A_2, ... A_n \to a_n B$
- *Replace each rule has the form :* $A \to a_1 a_2 ... a_n$
  by $A \to a_1 A_{1;} A_1 \to a_2 A_2; ...; A_n \to a_n$
- *Empty word*
  1. $\epsilon \in L(G) \Rightarrow \varepsilon \in L(G')$
  2. *If* $A \to \varepsilon$ *et* $A \neq S$ *then replace* A *by epsilon in the right side of the rules.*

Example

Let $G = (N, T, S, P)$ *be a regular grammar (right linear):*

- $N = \{S, U\}$
- $T = \{a, b\}$
- $P = (S \to aS \mid aU; U \to bU \mid b)$

Generates the language $L(G) = \{a^n b^m / n > 0, m > 0\}$

- *Regular grammars and languages are the basis of lexicography. All the keywords, identifiers, numeric constants, ... of a programming language such as C++ is a regular language and can be described by a regular grammar.*
- *The interest in distinguishing regular grammars on the right or on the left appears during the analysis*
- *if we reads the symbols of the word to be analyzed from left to right, then*
- *a linear grammar on the right will be used for a descending analysis, from the axiom towards the word ;*
- *a linear left grammar will be used for an ascending analysis, from the word to the axiom.*

4.1.3 Definition:

Let $\sum$ *be an alphabet(finite), Rational languages on* $\sum$ *are defined inductively by:*

- Φ*; is a regular language*
- $\{\varepsilon\}$*: is a regular language.*

- $\forall a \in \sum, \{a\}$: *is aregular language.*
- *if* L_1 *and* L_2 *are regulars then* $L_1.L_2$ *is regular.*
- *if* L *is regylar then* L^* *is regular.*
- *If* L_1 *and* L_2 *are regulars then* $L_1 \cup L_2$ *is regular.*

Theorem 5 *The operations union, intersection, complement, product, star, transpose and homomorphism are closure properties for regular languages. (Démonstration.)*

4.1.4 Definition:

Let a language L *be on an alphabet* $\sum$*, let* $w \in \sum^*$ *be a word. We call derived of the language* L *with respect to a* w *the set of words* z *such that* $wz \in L$ $(z \in \sum^*)$ *denoted by :* Lw *and formally we write:*

$Lw = \{z \in \sum^* / wz \in L\}$. *Example:* $\sum = \{0,1\}, L = \{010, 11, 01\}$ $L \parallel 0 = \{10, 1\}, L \parallel 01 = \{0, \varepsilon\}$

Theorem 6 *A language* L *on* X *is regular iff the number of its derivatives with respect to the words of* X^* *is finite.*

4.2 Pumping lemma

There is a property always verified by a regular language, this property known as pumping lemma Usually we use this lemma for proving that a given language is not regular. This pumping property is verified bu any regular language . If we can prove that a given language does not verify this property, then it is not regular. The property states that all strings in the language can be "pumped" if they are at least as long as a certain special value, called the pumping length. That means each such string contains a section that can be repeated any number of times with the resulting string remaining in the language.

Theorem 7 *If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:*

1. $\forall i \geq 0, xy^iz \in A$,
2. $\mid y \mid > 0, and$
3. $\mid xy \mid \leq p$.

Remark:

1. *The pumping lemma is not a sufficient condition, which guarantees that a language is regular (means there exist languages which are not regular, but for which this lemma is valid.)*
2. *if the star lemma (another name) is not verified by a language, then this latter cannot be regular.*

Example:

prove that $L(G) = \{a^m b^m / m \geq 0\}$ does not a regular language

- *Suppose $L(G)$ is regular, then $L(G)$ verify the pimping lemma.*
- *let's take n its value not defined, but each word belongs to L, its length is greater than or equals n verifies the conditions of pumping lemma.*
- *let $s = 0^m 1^m$, by definition $s \in L$, and $\mid s \mid = 2n \geq n$*
- *Applying the lemma s can be rewritten $s = xyz$, such as*
  $\mid xy \mid \leq n, \mid y \mid \geq 1, \forall i \geq 0, xy^iz \in L$
- *Consider $\mid xy \mid = k, k \leq n$(hypothesis).$\Rightarrow$*
- *xy is a prefix of s, and necessary of the form 0^k.*
- *We take $i = 0$, we must have $xy^0z = xz \in L$.*

- $xz = 0^{n-|y|}1^n$, *and* $\mid y \mid \geq 1$, $\Rightarrow s \notin L$ *contradiction with start hypotheis, then L does not regular.*

.

4.3 Regular Expressions

The major challenge in the theory of computation is how specify(describe) a language especially when this language is infinite set where listing all its words is not possible, here a formal description is possible, another useful tool is through Regular expression.

Here we advance the notions a little to give an example, to move from the normal specification to the specification with regular expressions:

Example:

- *Let* $A = \{a, b\}$ *, an alphabet. L:Language of words hast two non consecutive occurrences of 1,the first and second of which are not consecutive . This language can be described as:* $0^*100^*10^*$.
- *Regular expressions allow us to describe regular languages in a simpler way.*

Definition:

Let A be an alphabet the expression R is a regular expression on A iff: We say that R is a regular expression if R is:

1. *a where* $a \in A$,
2. ε,
3. ϕ,
4. $(R_1 \mid R_2)$, *where R1 and R2 are regular expressions.*
5. $(R_1.R_2)$,*where R1 and R2 are regular expressions,*
6. $(R_1)^*$, *where R1 and is a regular expressions.*
    - $\{a\}$*: represents the language* $\{a\}$.
    - ε*; represents the language* $\{\varepsilon\}$,
    - ϕ*: represents the empty language,*
    - $(R_1 \mid R_2)$*: language obtained by the union of R1 and R2,*
    - $(R_1.R_2)$*: represents the language obtained by the concatenation of R1 and R2,*
    - $(R_1)^*$: *Represents the star of language R1.*

4.3.1 Definition:

The language $\mathcal{L}(\mathcal{R})$, specified by the regular expression $\mathcal{R}$, defined on the alphabet A is :

- $\mathcal{L}(\mathcal{R}) = \phi$, *if* $\mathcal{R} = \phi$,
- $\mathcal{L}(\mathcal{R}) = \varepsilon$, *if* $\mathcal{R} = \varepsilon$
- $\mathcal{L}(\mathcal{R}) = \{a\}$, *if* $\mathcal{R} = a$
- $\mathcal{L}(R) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$, *if* $R = R_1 \mid R_2$
- $\mathcal{L}(R) = \mathcal{L}(R_1).\mathcal{L}(R_2)$, *if* $R = R_1.R_2$.
- $\mathcal{L}(R)^* = \mathcal{L}(R_1)^*$, *if* $R = R_1^*$
  (WHERE R_1 and R_2 are regular expressions on $\mathcal{A}$).

Example:

$R = a^*bbc^*$, specifies the language, $\mathcal{L}(R) = \{a^n bbc^p \not| n, p \geq 0\}$

Theorem 8 *A language $\mathcal{L}$, is regular iff it specified by a regular expression.*

4.3.2 Few algebraic laws

Let, e, f, g three regular expressions on the alphabet $\mathcal{A}$, the following algebraic laws are verified:

1. $\phi e = e\phi = \phi$.
2. $\phi^* = \varepsilon$.
3. $e\varepsilon = \varepsilon e = e$
4. $\varepsilon^* = \varepsilon$.
5. $e + f = f + e$.
6. $e + e = e$.
7. $e + \phi = e$.
8. $(e^*)^* = e^*$.
9. $e(f + g) = ef + eg$. $(e + f)g = eg + fg$.
10. $(ef)^*e = e(fe)^*$.
11. $(e + f)^* = e^*(e + f)^* = (e^* + f)^*$.
12. $(e + f)^* = (e^*f^*)^* = (e^*f)^*e^*$

Chapter 5

Finite Automata

In this chapter we present the Finite Automata(FA) as a tool to recognize regular languages, how to transfer Non Deterministic Finite Automata(NFA) to Deterministic Finite Automata(DFA), then the minimized (DFA),...

5.1 Introduction

- *Automata theory bases on mathematical concepts,*
- *Automata allow to create models.*
- *One model, called the (Finite automaton)->Regular grammar: is used in text processing, compilers, and hardware design.*
- *Another model, called the (Push down automaton)->Context-free grammar , is used in programming languages and artificial intelligence.*

5.2 Theoretical Review

- *As shown in the previous chapters we have used the formal grammar* $G = (N, T, S, P)$
- *To generate languages using productions or rewriting rules, each type for specific class of language as shown by figure(5.1) .*

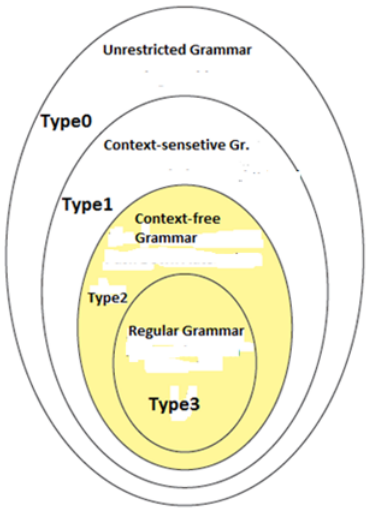

Fig. 5.1: Grammar classification

5.3 Finite Automata

- *It is Language recognition system (Accepting system) see this scheme(5.2) .*

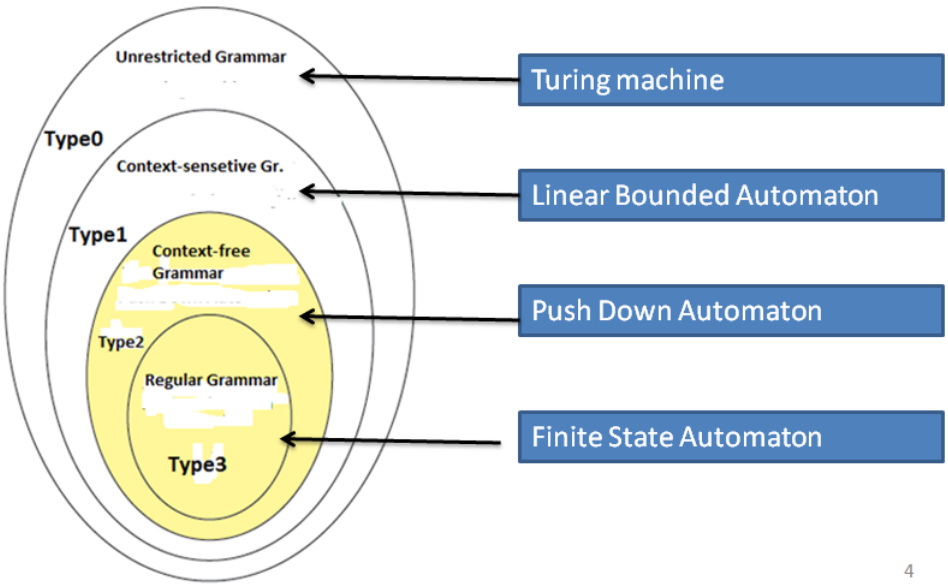

Fig. 5.2: Language recognition(Automata)

- *FINITE STATE MACHINE (FINITE AUTOMATA)*
- *The simplest scheme of automata*
- *Memory less machine*
- *Characterized by a finite input tape.*
- *Left to right reading(scanning). .*

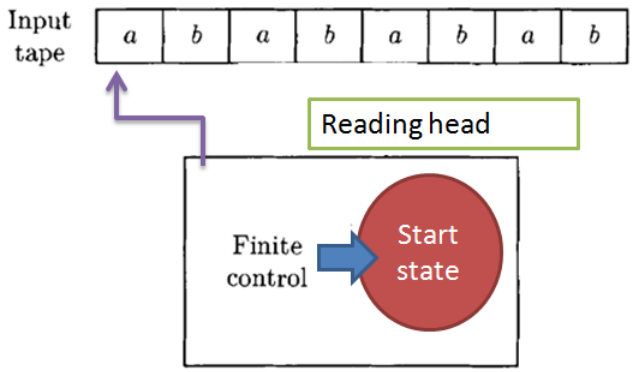

Fig. 5.3: Finite automata principle

5.3.1 Finite Automata Components

The following scheme presents the differents components of FA,

5.3.2 Defintion

A language is regular if and only if there is a regular grammar generating this language

5.3.3 Reminder-Regular grammar(type 3)

The rules of a regular grammar can either be of the form of:

$A \to \alpha B$ *ou* $A \to \alpha$ *avec* $A, B \in N$ *et* $\alpha \in T^*$ *(Linear on the right)*

Or :

$A \to B\alpha$ *ou* $A \to \alpha$ *avec* $A, B \in N$ *et* $\alpha \in T^*$ *(Linear on the left)*

5.3.3.1 K-grammar

The grammar G=(N,T,S,P) is a k-grammaire (Kleene's grammar) if the grammatical rules have the form:

- $A \to xB$ *ou* $A \to x$ *with* $A, B \in N$ *et* $x \in T$
- *All the rules of the form :* $A \to \epsilon$ *where* $(A \neq S)$ *are not accepted.*

Transformation of linear on the right grammar to K-grammar

Input : Linear on the right grammar $G\,(N, T, S, P)$

Output : K-grammar $G'\,(N', T', S', P')$.

- *Replace each rule has the form :* $A \to (a_1 a_2 ... a_n)\, B$
  by $A \to a_1 A_1, A_1 \to a_2 A_2, ... A_n \to a_n B$
- *Replace each rule has the form :* $A \to a_1 a_2 ... a_n$
  by $A \to a_1 A_{1;} A_1 \to a_2 A_2; ...; A_n \to a_n$
- *Empty word*
  1. $\epsilon \in L(G) \Rightarrow \varepsilon \in L(G')$
  2. *If* $A \to \varepsilon$ *et* $A \neq S$ *then replace A by epsilon in the right side of the rules.*

Example

Let $G = (N, T, S, P)$ *be a regular grammar (right linear):*

- $N = \{S, U\}$
- $T = \{a, b\}$
- $P = (S \to aS \mid aU; U \to bU \mid b)$

Generates the language $L(G) = \{a^n b^m / n > 0, m > 0\}$

Bibliography

Appendix **A**

Code