

Lectures notes on Theory of automata and Formal languages

Computer Science Department
Malaga University

PGF version: 3.0.1a

October 1, 2019

Contents

1	Preamble to discrete mathematics	9
1.1	Reminder	10
1.2	Finite and Infinite Sets	19
2	Languages and grammars	23
2.1	Languages	24
2.1.1	Concept of language	24
2.1.2	Language representation	31
2.2	Grammatical representation systems	32
2.2.1	Definition and functioning of grammars	32
2.2.2	Classification of rules	35
2.2.3	Classification of grammars	37
2.2.4	Classification of languages	38
2.3	Questions about languages	39
2.4	Operations and closures of languages	40
2.5	Closures for the types of languages	43
3	Regular expressions	45
3.1	Definition of the regular expressions	46

3.2	Some properties of regular expressions	50
4	Finite automata	53
4.1	Deterministic finite automata	54
4.2	Nondeterministic finite automata	60
4.3	Minimum deterministic finite automaton	66
4.4	Equivalences	66
5	Regularity conditions	67
5.1	Myhill-Nerode	68
5.2	Pumping lemma	72
6	Context-free languages	75
6.1	Derivations and Ambiguity	76
6.2	Recursion	81
6.3	CFG Simplification	83
6.4	Normal forms	85
6.5	Closure properties	85
6.6	Non-Deterministic Pushdown Automata	86
6.7	Pumping	91
7	The Turing machine	93

7.1	Formal definition	94
7.2	Computability, decidability, enumerability	104
8	Recursive functions	107
8.1	Formal definition	108
8.2	Enumerability, decidability, computability	116
9	The WHILE language	119
9.1	Formal definition	120
9.2	Computability, decidability, enumerability	127
10	Turing completeness	129
10.1	Extended While language	130
10.2	Recursive \Rightarrow While-computable	132
11	Universality	137
11.1	Program numbering	138
11.1.1	Encoding of While programs	142
11.1.2	Decoding While programs	145
11.2	Universal function	149
11.3	Universal program	151
12	Theoretical limits of computing	155

12.1 Halting problem	156
12.2 The busy beaver function	159

References	163
-------------------	------------

Foreword

Writing just another textbook in theory of automata, formal languages or computability looks like pointless: there exist so many now, that we can hardly do better. However, every lecturer knows what is to be taught, and the lectures benefit from a particular nomenclature in a given subject, or a specific proof. In the end, you miss that textbook that satisfies your needs, that fits your communication skills or your expertise in the area. Such is the motivation behind compiling these lectures notes in the subject of Theory of automata and formal languages, in the traditional sense, but also, to be an ever-evolving document, that integrates what others have expressed before, the examples and exercises that were designed by other authors in distant parts of the planet, unsubscribing from a particular terminology, and unifying different forms of representing the same concepts. It is the sign of the XXI century: collaboration, contribution to common initiatives. The resulting manuscript has the structure of [Ramos and Morales, 2011], which is the reference book in Spanish for the subject *Teoría de autómatas y lenguajes formales* of our Computer Science pro-

gram. Much of the writing in \LaTeX has been done by students taking the subject, and curated by myself, as professor of Computer Science and Artificial Intelligence at the University of Malaga.

List of contributors: D. Arroyo Torres, A. Burgueño Romero, R. Ferreira Garcés, P. Gutierrez Ruiz, M. López Reviriego, M. Mejía Jiménez, D. Mérida Granados, J. Morales Joya, R. Moret Galán, J. Pérez Bravo, A. Román Navas and M. Guerrero Ramírez.

Additionally, a library of open-source software that implements the main concepts in this manuscript is also offered for the programming language GNU/Octave, and it has been made public under CC0 license. This is just version 1.0, coming years will see it develop, enrich and accommodate new knowledge, teaching expertise and students' feedback.

Francisco Vico
November 2018

Notation

\mathbb{N}	the natural/whole numbers set, i.e. $\{1, 2, \dots\}$ (U+2115)
\mathbb{N}_0	the natural numbers set with zero , i.e. $\{0, 2, \dots\}$
\mathbb{R}	the real numbers set (U+211d)
\mathbb{Z}	the integer numbers set (U+2124)
\mathbb{Q}	the rational numbers set (U+211a)
\mathbb{R}	the real numbers set (U+211d)
\mathbb{Z}	the integer numbers set (U+2124)
\mathbb{Q}	the rational numbers set (U+211a)
\emptyset	the empty set (U+2205)
\in	belongs to (U + 2208)
\notin	does not belong to (U + 2209)

\forall	universal quantification, for all; for any; for each (U+2200)
\exists	existential quantification, there exists (U+2203)
$\exists!$	uniqueness quantification, there exists exactly one (U+2203 U+0021)
\subseteq	the (infix) subset relation between sets (U+2286)
\subset	the (infix) proper subset relation between sets (U+228a)
\cup	the infix union operation on sets (U+222a)
\cap	the infix intersection operation on sets (U+2229)
\wedge	logical conjunction (U+2227)
\vee	logical (inclusive) disjunction (U+2228)
$-$	the infix set difference operation on sets (U+2212)
\times	the infix cartesian product of sets (U+00d7)
A^n	the postfix n -fold cartesian product of A , i.e. $A \times \cdots \times A$
$\mathcal{P}(A)$	the power set of A (also represented by 2^A)
\vdash	turnstile symbol to denote transition in automata (U+22A2)
ε	empty string (U+03B5)
\mathcal{L}	set of languages (U+2112)
\mathcal{L}	language generated by a metastring (U+1D4DB)
$:=$	assignment in While language (U+2254)

Chapter 1

Preamble to discrete mathematics

1.1	Reminder	10
1.2	Finite and Infinite Sets	19

No one shall expel us from the paradise that Cantor has created.
David Hilbert

1.1 Reminder

Definition 1.1.1. Power set (*conjunto potencia*)

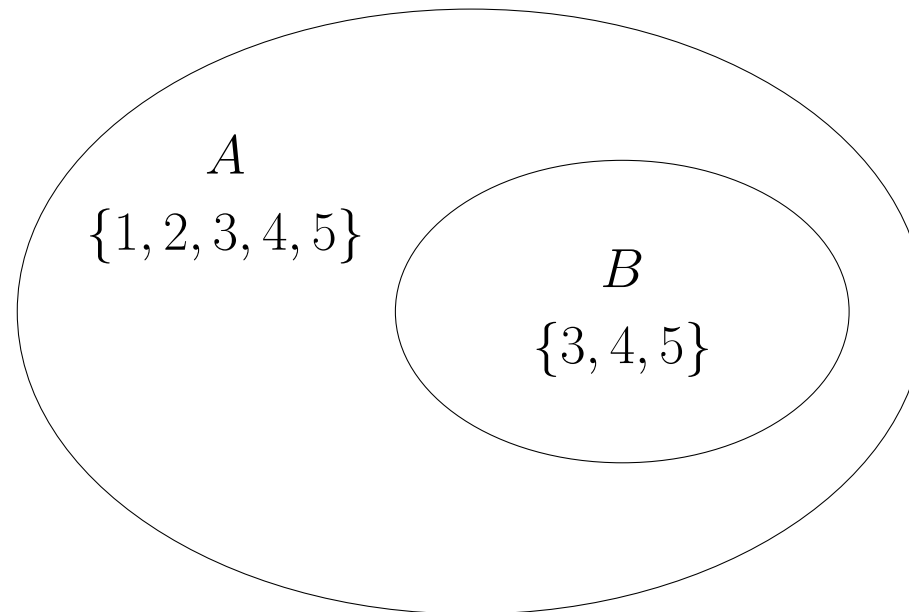
$$\mathcal{P}(A) = \{B : B \subseteq A\}$$

Remark. $\emptyset, A \in \mathcal{P}(A)$

Definition 1.1.2. Proper subset (*subconjunto propio*)

B is a proper subset of A iff

$$B \subset A \wedge B \neq \emptyset$$



Definition 1.1.3. Partition (*partición*)

$\Pi = \{A_1, A_2, \dots\} \subseteq \mathcal{P}(A)$ is a partition of A iff

- $\bigcup_{A_i \in \Pi} A_i = A$
- $A_i \cap A_j = \emptyset, \forall A_i \neq A_j$
- $A_i \neq \emptyset, \forall A_i \in \Pi$

Definition 1.1.4. Cartesian product (*producto cartesiano*)

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

Remark. (a, b) is an ordered pair (*par ordenado*).

Definition 1.1.5. Relation (*relación*)

$R \subseteq A \times B$ is a relation from A to B .

Definition 1.1.6. Binary relation (*relación binaria*)

$R \subseteq A \times A$ is a binary relation on A .

Definition 1.1.7. Properties of binary relations

- R is Reflexive iff $(a, a) \in R \quad \forall a \in A$
- R is Symmetric iff $(a, b) \in R \Rightarrow (b, a) \in R$
- R is Antisymmetric iff $(a, b) \in R \wedge a \neq b \Rightarrow (b, a) \notin R$
- R is Transitive iff $(a, b) \in R \wedge (b, c) \in R \Rightarrow (a, c) \in R$

Definition 1.1.8. Equivalence relation (*relación de equivalencia*)
 R is an equivalence relation iff it is reflexive, symmetric and transitive.

Definition 1.1.9. Equivalence class (*clase de equivalencia*)
Being A a set, R an equivalence relation on A and $a \in A$, $[a]$ is an equivalence class defined as

$$[a] = \{b \in A : (a, b) \in R\}$$

Remark. An equivalence relation of a set defines a partition for that set (quotient set) where each element of the partition is an equivalence class.

Definition 1.1.10. Identity relation I (*relación identidad*)

$$I = \{(a, a) : a \in A\}.$$

Definition 1.1.11. Inverse relation R^{-1} (*relación inversa*)

$$R^{-1} = \{(a, b) : (b, a) \in R\}$$

Definition 1.1.12. Power of a relation R^n (*potencia de una relación*)

Given $R \subseteq A \times A$,

$$R^n = \begin{cases} R & n = 1 \\ \{(a, b) : \exists x \in A, (a, x) \in R^{n-1} \wedge (x, b) \in R\} & n > 1 \end{cases}$$

Definition 1.1.13. Closure of relations (*cierre de relaciones*)

The closure of a relation under a certain property is the smallest relation which contains the first one and satisfies that property.

- Reflexive closure of R is $R \cup I$
- Symmetric closure of R is $R \cup R^{-1}$
- Transitive closure of R is $R^\infty = \bigcup_{n=1}^{\infty} R^n$

Example 1.1.1. Given $A = \{a, b, c\}$, and the binary relation $R = \{(b, c), (c, a), (a, c)\}$

$$R^2 = \{(b, a), (c, c), (a, a)\}$$

$$R^3 = \{(b, c), (c, a), (a, c)\} = R$$

$$R^\infty = \{(b, c), (c, a), (a, c), (b, a), (c, c), (a, a)\}$$

Definition 1.1.14. Function (*función*)

$f : A \rightarrow B$ is a function iff $\forall a \in A \exists! (a, b) \in f$.

Remark. "!" denotes "one and only one".

Definition 1.1.15. Domain and range of a function (*dominio y rango*)

Given $f : A \rightarrow B$,

$$Dom(f) = \{a \in A : f(a) = b, b \in B\}$$

$$Rg(f) = \{f(a) \in B : a \in A\}$$

Definition 1.1.16. Injective function (*función inyectiva*)

$f : A \rightarrow B$ iff $f(x) = f(z) \Rightarrow x = z$

Remark. This also means that $x \neq z \Rightarrow f(x) \neq f(z)$

Definition 1.1.17. Surjective function (*función sobreyectiva*)

$f : A \rightarrow B$ iff $Rg(f) = B$

Definition 1.1.18. Bijective function (*función biyectiva*)

$f : A \leftrightarrow B$ iff it is injective and surjective.

Example 1.1.2.

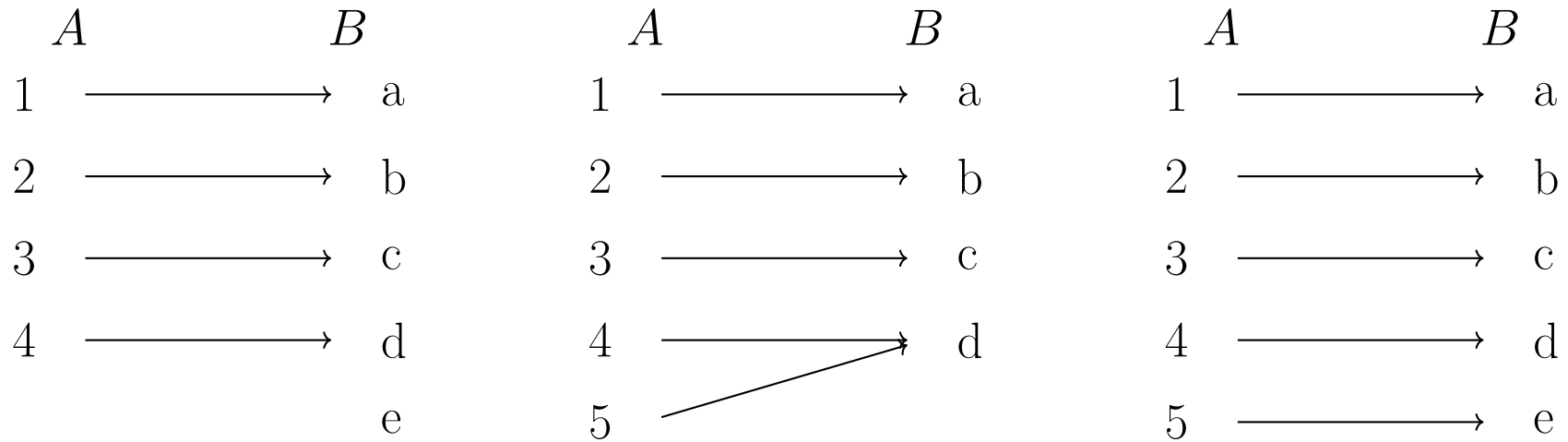


Figure 1.1: Three examples of functions: injective function (left), surjective function (center), and bijective function (right).

Definition 1.1.19. Closure of a set under an operation

A set is closed under an operation iff this operation can be defined as a function

$$f : A \times A \rightarrow A$$

Definition 1.1.20. Associative property (*propiedad asociativa*)

Being A a closed set for the operation \bullet , $(\bullet : A \times A \rightarrow A)$, \bullet is associative iff

$$(x \bullet y) \bullet z = x \bullet (y \bullet z) \forall x, y, z \in A$$

Definition 1.1.21. Semigroup (*semigrupo*)

A semigroup is a pair (A, \bullet) where A is a set closed under \bullet , which is an associative operation.

Definition 1.1.22. Neutral element (*elemento neutro*)

Also called identity element, $e \in A$ is the neutral element for an operation \bullet iff

$$\forall a \in A \quad a \bullet e = e \bullet a = a$$

Definition 1.1.23. Monoid (*monoide*)

It is a semigroup with a neutral element for the operation.

Definition 1.1.24. Strict closure of a set under an operation (*cierre estricto*)

Being \bullet an operation on A and $B \subseteq A$, B^\bullet is the strict closure of B for \bullet defined as:

- $x \in B \Rightarrow x \in B^\bullet$
- $x, y \in B^\bullet \Rightarrow x \bullet y \in B^\bullet$
- no other element belongs to B^\bullet .

Definition 1.1.25. Wide closure of a set under an operation (*cierre amplio*)

Being (A, \bullet) a monoid and $B \subseteq A$, the wide closure of B under \bullet is

$$B^{\bullet e} = B^\bullet \cup \{e\}$$

1.2 Finite and Infinite Sets

Definition 1.2.1. Equipotent sets (*conjuntos equipotenciales*)

$A \sim B$ (are equipotent) iff $\exists f : A \leftrightarrow B$

Definition 1.2.2. Cardinality of a set (*cardinalidad de un conjunto*)

$$|A| = \begin{cases} 0 & \text{if } A = \emptyset \\ n \in \mathbb{N} & \text{if } A \sim \{1, 2, \dots, n\} \\ \aleph_0 & \text{if } A \sim \mathbb{N} \\ \aleph_i & \text{if } A \sim \mathcal{P}(\aleph_{i-1}), i > 0 \end{cases}$$

Remark.

$\emptyset = \{ \}$ is the empty set (*conjunto vacío*)

$|\mathbb{N}| = \aleph_0$ (*aleph-null* (*álef cero*))

$|\mathbb{R}| = \aleph_1$ (*aleph-one* (*álef uno*)), according to the continuum hypothesis.

Proposition 1.2.1. $\aleph_1 - \aleph_0 = \aleph_1$

Definition 1.2.3. Finite set (*conjunto finito*)

A is a finite set iff $|A| \in \mathbb{N}_0$.

Definition 1.2.4. Infinite set (*conjunto infinito*)

A is an infinite set iff $|A|$ is not finite.

Remark. A is infinite iff $|A| \in \{\aleph_i : i \in \mathbb{N}_0\}$.

Definition 1.2.5. Countable infinite set (*conjunto infinito numerable*)

A is a countable infinite set iff $A \sim \mathbb{N}$.

Definition 1.2.6. Countable set (*conjunto numerable*)

A is countable if it is finite or countable infinite.

Definition 1.2.7. Uncountable set (*conjunto no numerable*)

A is uncountable if it is not countable.

Remark. \mathbb{R} is an uncountable set.

Proposition 1.2.2. Any subset of a countable set is countable.

Proposition 1.2.3. The finite union of countable sets is countable.

Proposition 1.2.4. Countable infinite union of countable sets is countable.

Proposition 1.2.5. The cartesian product of two countable sets is countable.

Proposition 1.2.6. The finite power of a countable set is countable.

Theorem 1.2.1. Canthor's theorem (*teorema de Cantor*)

$$|A| < |\mathcal{P}(A)|$$

Remark. $|\mathbb{N}| < |\mathcal{P}(\mathbb{N})| \Rightarrow \aleph_0 < \aleph_1 \Rightarrow |\mathbb{N}| < |\mathbb{R}|$

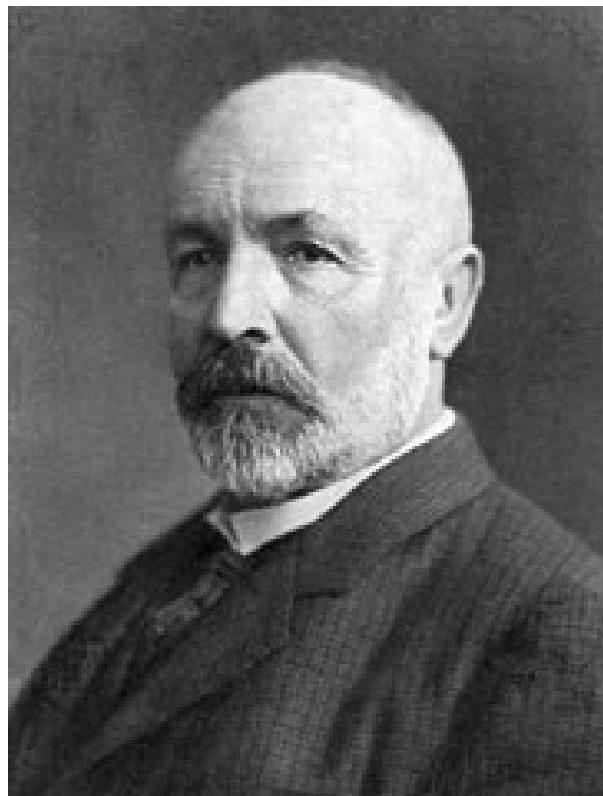


Figure 1.2: [Georg Cantor](#) defined the transfinite numbers by the end of the XIX century.

Chapter 2

Languages and grammars

2.1	Languages	24
2.2	Grammatical representation systems	32
2.3	Questions about languages	39
2.4	Operations and closures of languages	40
2.5	Closures for the types of languages	43

2.1 Languages

2.1.1 Concept of language

Definition 2.1.1. Alphabet (*alfabeto*)

A non empty, finite set of symbols.

$$\Sigma = \{s_1, \dots, s_k\}, k \geq 1$$

Definition 2.1.2. String over an alphabet (*cadena sobre un alfabeto*)

A finite sequence of symbols taken from a given alphabet,

$$w = (a_1, \dots, a_n), a_i \in \Sigma$$

Remark. Defined as a vector (ordered set), a string (or word) will be represented by the sequence of symbols: $w = a_1 \dots a_n$.

Definition 2.1.3. Empty string (*cadena vacía*)

$$\varepsilon = ()$$

Definition 2.1.4. Length of a string

$$|w| = \begin{cases} 0 & \text{if } w = \varepsilon \\ n \in \mathbb{N} & \text{if } w = (a_1, \dots, a_n) \end{cases}$$

Definition 2.1.5. Occurrence (*ocurrencia*)

Given $w = (a_1, \dots, a_n) \in \Sigma^+$, and $1 \leq j \leq n$,

$$w(j) = a_j$$

Definition 2.1.6. Number of occurrences (*número de ocurrencias*)

$$|w|_a = |\{j : w(j) = a\}|, \forall a \in \Sigma$$

Remark. $|\varepsilon|_a = 0, \forall a \in \Sigma$

Definition 2.1.7. String concatenation (*concatenación de cadenas*)

$x \cdot y$ or simply xy is the string verifying

$$|xy| = |x| + |y|$$
$$xy(j) = \begin{cases} x(j) & \text{if } 0 < j \leq |x| \\ y(j - |x|) & \text{if } |x| < j \leq |xy| \end{cases}$$

Remark. \cdot is associative and ε is its neutral element, hence (Σ^*, \cdot) is a monoid.

Definition 2.1.8. $\Sigma^+ \Sigma^*$ (*sigma más, sigma estrella*)

Σ^+ is the strict closure of Σ under the string concatenation operation, and Σ^* is its wide closure, defined as

$$\Sigma^+ = \{w = a_1 \dots a_k : a_i \in \Sigma \wedge k \geq 1\}$$
$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$$

Definition 2.1.9. Prefix (*prefijo*)

$$v \sqsubseteq w \text{ iff } \exists x : w = vx.$$

Definition 2.1.10. Suffix (*sufijo*)

$$v \sqsupseteq w \text{ iff } \exists x : w = xv.$$

Definition 2.1.11. Substring (*subcadena*)

$$v \prec w \text{ iff } \exists x, y : w = xvy.$$

Remark. ε is prefix, suffix and a substring of any string.

Definition 2.1.12. Power of a string (*potencia de una cadena*)

$$w^n = \begin{cases} \varepsilon & \text{if } n = 0 \\ w^{n-1}w & \text{if } n > 0 \end{cases}$$

Definition 2.1.13. Reversed string (*cadena inversa*)

$$w^R = \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ au^R & \text{if } w = ua, u \in \Sigma^*, a \in \Sigma \end{cases}$$

Proposition 2.1.1. $\forall x, w \in \Sigma^*, (wx)^R = x^R w^R$

Definition 2.1.14. Language over an alphabet

L is a language over Σ iff $L \subseteq \Sigma^*$

If a language is finite it can be defined by *extension* (e.g. $L = \{ac, bca, caaad\}$).

If a language is infinite it can only be defined by *compression* as

$$L = \{w \in \Sigma^* : P(w)\}$$

where P is a property verified by all the strings in L .

Proposition 2.1.2. $|\Sigma^*| = \aleph_0$

Proof. This means that $\Sigma^* \sim \mathbb{N}$. To prove it, let's assume an order on $\Sigma = \{a_1, \dots, a_n\}$, then all strings of Σ^* can be enumerated, first by length and secondly by alphabetical order:

<i>length</i>	<i>strings</i>		
0	ε		
1	a_1	a_2	\dots
2	a_1a_1	a_1a_2	\dots
3	\dots		

so a bijective function can be defined with \mathbb{N} . □

Example 2.1.1. On the use of alphabets, strings and its operations, and languages
 $\Sigma = \{a, b, c\}$

$$\begin{array}{ll} v = bbbcbba & |v| = 8 \\ w = cbbca & |w| = 5 \end{array}$$

$$\begin{array}{ll} x = vw = bbbcbbaacbbca & \\ x(4) = c & |x|_b = 8 \\ x(9) = c & |x|_c = 3 \end{array}$$

$$\begin{array}{ll} bbc \sqsubseteq x & bacb \prec x \\ acbbca \sqsupseteq x & acbbca \prec x \end{array}$$

$$\begin{array}{l} w^3 = cbbcacbbcacbbca \\ w^R = acbbc \end{array}$$

$$\begin{array}{ll} L = \{\varepsilon, b, acc, ababc\} & |L| = 4 \\ L = \{w \in \Sigma : abc \sqsubseteq w\} & |L| = \aleph_0 \\ L = \{w \in \Sigma : |w|_a = 3n \in \mathbb{N}_0\} & |L| = \aleph_0 \end{array}$$

Definition 2.1.15. Representation system

Let Σ_M be a meta-alphabet (i.e. the alphabet of the natural language, including all the mathematical symbols)

$$\Sigma_M = \{A, \dots, Z, a, \dots, z, 0, \dots, 9, \forall, \exists, \Rightarrow, \wedge, \vee, \dots\}$$

A representation system is a function from strings over a meta-alphabet (Σ_M) to languages over an alphabet (Σ), namely

$$\mathcal{R} : \Sigma_M^* \rightarrow \mathcal{P}(\Sigma^*)$$

Remark. \mathcal{R} establishes an unambiguous correspondence of strings and languages:

$$(r, L) \in \mathcal{R} \Rightarrow \nexists (r, L') \in \mathcal{R} : L' \in \mathcal{P}(\Sigma^*) \wedge L' \neq L$$

Definition 2.1.16. Set of the representable languages, $\mathcal{L}.\text{REP}$

$$\mathcal{L}.\text{REP} = \{L \in \mathcal{P}(\Sigma^*) : \exists (r, L) \in \mathcal{R}\}$$

Remark.

Σ	is a finite set
Σ^*	is a countable infinite set
L	is a countable set (either finite or countable infinite)
$\mathcal{L}.\text{REP}$	is a countable infinite set
$\mathcal{P}(\Sigma^*)$	is an uncountable set

Proposition 2.1.3. There are uncountable non representable languages

Proof. The set of all language representations for a given \mathcal{R} will be $\text{REP} \subseteq \Sigma_M^*$, and each representation can only represent one language (which is then representable), so

$$|\mathcal{L}.\text{REP}| \leq |\text{REP}|$$

and since $|\Sigma_M^*| = \aleph_0$,

$$|\text{REP}| \leq \aleph_0$$

Also, every finite language is representable by extension, and since there is a countable infinite number of finite languages

$$\aleph_0 \leq |\mathcal{L}.\text{REP}|$$

and then

$$\aleph_0 \leq |\mathcal{L}.\text{REP}| \leq |\text{REP}| \leq \aleph_0 \Rightarrow |\mathcal{L}.\text{REP}| = |\text{REP}| = \aleph_0$$

By definition, $L \subseteq \Sigma^*$ and the set of all possible languages over Σ is $\mathcal{P}(\Sigma^*)$, then

$$|\Sigma^*| = \aleph_0 \Rightarrow |\mathcal{P}(\Sigma^*)| = \aleph_1$$

The set of non representable languages is formed by the languages not in $\mathcal{L}.\text{REP}$.

$$\mathcal{L}.\text{NOREP} = \mathcal{P}(\Sigma^*) - \mathcal{L}.\text{REP}$$

And given the cardinality of $\mathcal{P}(\Sigma^*)$ and $\mathcal{L}.\text{REP}$,

$$\aleph_1 - \aleph_0 = \aleph_1 \Rightarrow |\mathcal{L}.\text{NOREP}| = \aleph_1$$

□

2.1.2 Language representation

Definition 2.1.17. Conclusive algorithm (*algoritmo conclusivo*)

Finite sequence of finite, precise, unambiguous instructions, that, in a finite time, returns an output for a given input.

Definition 2.1.18. Language recognizer (*reconocedor de lenguajes*)

A conclusive algorithm designed for a given language L that determines whether or not a string belongs to L .

Definition 2.1.19. Language generator (*generador de lenguajes*)

A system of conclusive algorithms that are able to produce all, and only the strings of a given language L .

2.2 Grammatical representation systems

2.2.1 Definition and functioning of grammars

Definition 2.2.1. Grammar

A grammar is a language generator defined as a quadruple $G = (N, T, P, S)$ where

- N is the *non-terminal alphabet* (non terminals are capital Latin letters)
- T is the *terminal alphabet* (terminals are lower case Latin letters)
 $N \cap T = \emptyset$
 $N \cup T = V$
- $P \subset V^+ \times V^*$ is the *production system* or *ruleset* and it is finite
 $(\alpha, \beta) \in P$ is a production rule, noted $\alpha \rightarrow \beta$
- $S \in N$ is the *axiom*, or *starting symbol*, of the grammar.

Definition 2.2.2. Produce directly (*producir directamente*)

Given $G = (N, T, P, S)$, $x \in V^+$ and $y \in V^*$, then x produces y directly, $x \Rightarrow y$, if $\exists u, v \in V^*$ satisfying

$$x = uzv$$

$$y = u\beta v$$

$$\exists z \rightarrow \beta \in P$$

Definition 2.2.3. Produce in n steps (*producir en n pasos*)

x produces y in 0 steps: $x \Rightarrow^0 y$ iff $x = y$

x produces y in 1 steps: $x \Rightarrow^1 y$ iff $x \Rightarrow y$

x produces y in n steps: $x \Rightarrow^n y$ iff

$$\exists z_1, z_2, \dots, z_{n-1} \in V^+ : x \Rightarrow z_1, z_1 \Rightarrow z_2, \dots, z_{n-1} \Rightarrow y$$

Definition 2.2.4. Produce in at least one step

$$x \Rightarrow^+ y \text{ iff } \exists n > 0 : x \Rightarrow^n y$$

Definition 2.2.5. Produce (*producir*)

$$x \Rightarrow^* y \text{ iff } \exists n \geq 0 : x \Rightarrow^n y$$

\Rightarrow is a binary relation over the strings in V^*

\Rightarrow^+ is its transitive closure

\Rightarrow^* is its reflexive and transitive closure

Definition 2.2.6. Derivation, length of a derivation (*derivación*)

Given $G = (N, T, P, S)$, a derivation of G is a finite sequence of strings over V , $(w_0, w_1, w_2 \dots w_n)$, such that

$$w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n$$

where $w_0 = S$ and $n \in \mathbb{N}_0$ is the length of the derivation.

Definition 2.2.7. Sentential form (*forma sentencial*)

$\alpha \in V^*$ is a sentential form of a grammar G iff

$$S \Rightarrow^* \alpha$$

Definition 2.2.8. String generated by a grammar

$y \in T^*$ is generated by G iff

$$S \Rightarrow^* y$$

Definition 2.2.9. Language generated by a grammar

$G = (N, T, P, S)$ generates the language

$$\mathcal{L}(G) = \{y \in T^* : S \Rightarrow^* y\}$$

2.2.2 Classification of rules

In what follows, a rule of P will be classified as type n if it meets, at least, the constraints up to type n (e.g., if a rule satisfies the constraints of type 1, but not those of type 2, then it is classified as type 1, even if it also fits -and it does- with type 0).

Definition 2.2.10. Type 0 (phrase structure) rule (*regla con estructura de frase*)

$$\alpha \rightarrow \beta$$

with $\alpha \in V^+, \beta \in V^*$.

Remark. Every rule is type 0.

Definition 2.2.11. Type 1 (context-sensitive) rule (*regla sensible al contexto*)

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

with $\alpha, \beta \in V^*, \gamma \in V^+, A \in N$.

Remark. α and β constitute the *context*.

Definition 2.2.12. Type 2 (context free) rule (*regla independiente del contexto*)

$$A \rightarrow \alpha$$

Definition 2.2.13. Left-regular rule (*regla regular izquierda*)

$$A \rightarrow aB$$

Definition 2.2.14. Right-regular rule (*regla regular derecha*)

$$A \rightarrow Ba$$

Definition 2.2.15. Terminal-regular rule (*regla regular terminal*)

$$A \rightarrow a$$

Definition 2.2.16. Type 3 (or regular) rule (*regla regular*)

A rule is regular iff it is left-regular, or it is right-regular, or it is terminal-regular.

Proposition 2.2.1. Rules hierarchy

$$\text{Type 3 rule} \subset \text{Type 2 rule} \subset \text{Type 1 rule} \subset \text{Type 0 rule}$$

Definition 2.2.17. Epsilon rule

$$A \rightarrow \varepsilon$$

2.2.3 Classification of grammars

Definition 2.2.18. Type 0 (phrase structure) grammar
 G is type 0 iff r is type 0, $\forall r \in P$.

Remark. Every grammar is type 0.

Definition 2.2.19. Type 1 (context-sensitive) grammar
 G is type 1 iff r is type 1, $\forall r \in P$.

Definition 2.2.20. Type 2 (context free) grammar
 G is type 2 iff r is type 2, $\forall r \in P$.

Definition 2.2.21. Left-regular type grammar
 G is left-regular type iff r is left-regular or terminal-regular type, $\forall r \in P$.

Definition 2.2.22. Right-regular type grammar
 G is right-regular type iff r is right-regular or terminal-regular type, $\forall r \in P$.

Definition 2.2.23. Type 3 (regular) grammar
 G is a regular grammar iff it is a left-regular grammar or a right-regular grammar.

Proposition 2.2.2. Grammars hierarchy

Type 3 grammar \subset Type 2 grammar \subset Type 1 grammar \subset Type 0 grammar

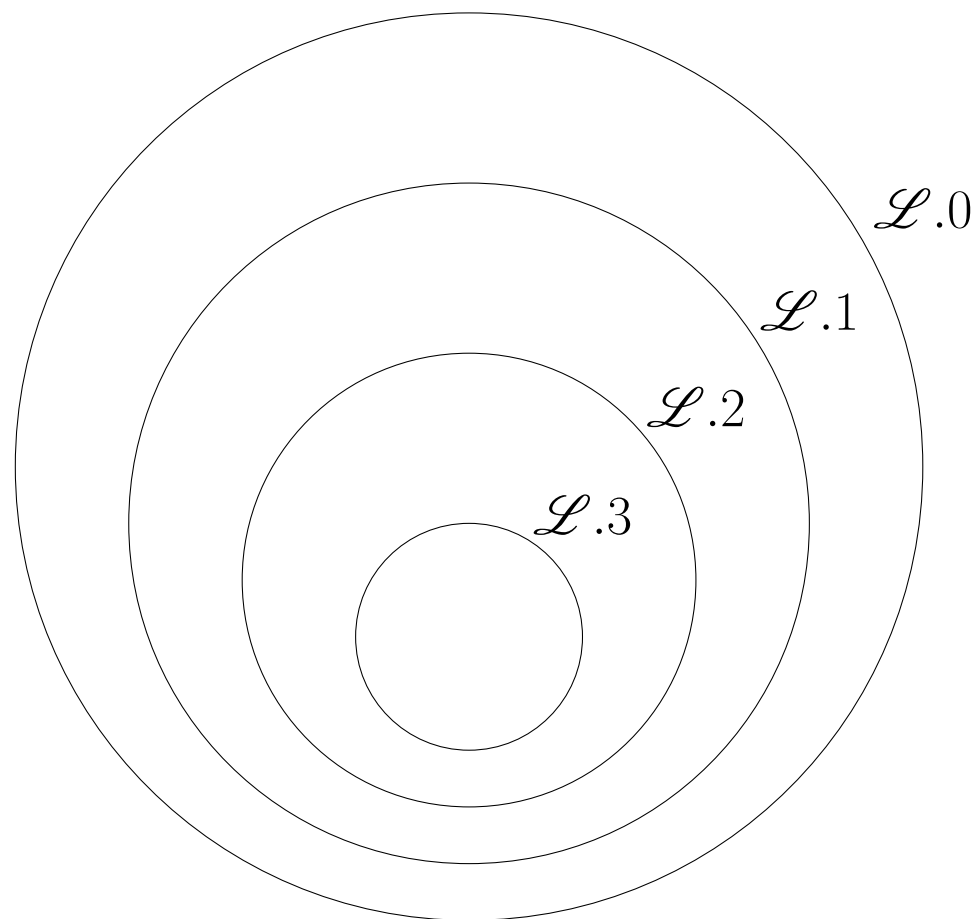
2.2.4 Classification of languages

Definition 2.2.24. Type i ($\mathcal{L}.i$) language

A language L is type $i \in \{0 \dots 3\}$ ($L \in \mathcal{L}.i$) if there exists a type i grammar such that $\mathcal{L}(G) = L - \{\varepsilon\}$.

Proposition 2.2.3. Chomsky's formal languages hierarchy

$$\mathcal{L}.3 \subset \mathcal{L}.2 \subset \mathcal{L}.1 \subset \mathcal{L}.0$$



2.3 Questions about languages

Definition 2.3.1. Equivalent grammars: $G_1 \equiv G_2 \Rightarrow \mathcal{L}(G_1) = \mathcal{L}(G_2)$

The following table shows whether or not it exists a conclusive algorithm to answer these questions according to the type of grammar:

	phrase structure	context-sensitive	context-free	regular
Equivalence $G_1 \equiv G_2$	no	no	no	yes
Inclusion $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$	no	no	no	yes
Pertenence $x \in \mathcal{L}(G)$	no	yes	yes	yes
Emptiness $\mathcal{L}(G) = \emptyset$	no	no	yes	yes
Finiteness $ \mathcal{L}(G) \in \mathbb{N}$	no	no	yes	yes
Regularity $\mathcal{L}(G) \in \mathcal{L}.3$	no	no	no	yes

2.4 Operations and closures of languages

Since languages are sets of strings, regular sets operations like *union*, *intersection*, *difference* and *complement* are also aplicable to them. Besides those mentioned, there are some other operations that are specific to languages, like the following.

Definition 2.4.1. Concatenation of languages

$$L_1 \cdot L_2 = \{xy \in \Sigma^* : x \in L_1, y \in L_2\}$$

where $L_1, L_2 \subseteq \Sigma^*$.

Remark. Sometimes the symbol '.' might be omitted, leaving L_1L_2 .

Language concatenation verifies these properties:

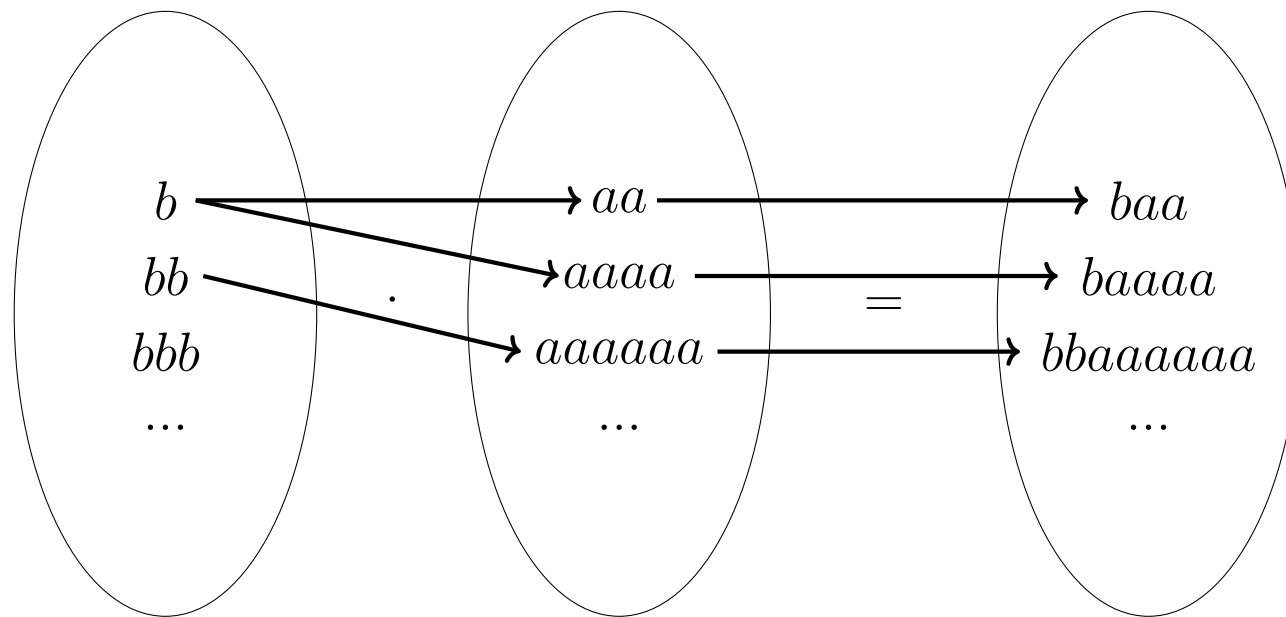
$$L \cdot \emptyset = \emptyset \cdot L = \emptyset$$

$$L \cdot \{\varepsilon\} = \{\varepsilon\} \cdot L = L$$

$$L_1 \cdot (L_2 \cdot L_3) = (L_1 \cdot L_2) \cdot L_3$$

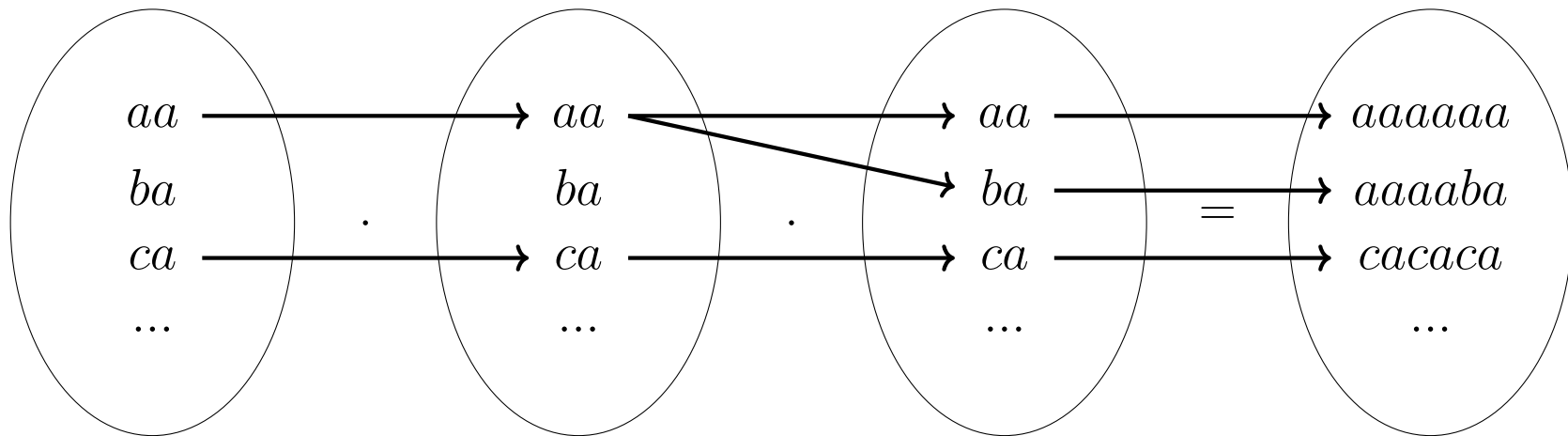
$$L_1 \cdot (L_2 \cup L_3) = L_1 \cdot L_2 \cup L_1 \cdot L_3$$

$$(L_1 \cup L_2) \cdot L_3 = L_1 \cdot L_3 \cup L_2 \cdot L_3$$



Definition 2.4.2. Power of a language (*potencia de un lenguaje*)

$$L^n = \begin{cases} \{\varepsilon\} & \text{if } n = 0 \\ L^{n-1} \cdot L & \text{otherwise} \end{cases}$$



Definition 2.4.3. Kleene closure (Kleene star) (*cierre o estrella de Kleene*)

$$L^* = \bigcup_{i \in \mathbb{N}_0} L^i$$

Remark. $\emptyset^* = \{\varepsilon\}$

Definition 2.4.4. Language reversal (*inverso de un lenguaje*)

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

2.5 Closures for the types of languages

The following table shows whether or not each language type is closed to a certain operation:

	phrase structure	context-sensitive	context-free	regular
Union	yes	yes	yes	yes
Intersection	yes	yes	no	yes
Intersection with $\mathcal{L}.3$	yes	yes	yes	yes
Complement	no	yes	no	yes
Concatenation	yes	yes	yes	yes
Power	yes	yes	yes	yes
Kleene closure	yes	yes	yes	yes
Reverse	yes	yes	yes	yes

Proposition 2.5.1. Language types hierarchy

$$\mathcal{L}.finite \subset \mathcal{L}.3 \subset \mathcal{L}.2 \subset \mathcal{L}.1 \subset \mathcal{L}.0 \subset \mathcal{L}.REP \subset \mathcal{P}(\Sigma^*)$$

Remark. $\mathcal{L}.0$ not being complement-closed $\Rightarrow \mathcal{L}.0 \neq \mathcal{L}.REP$.

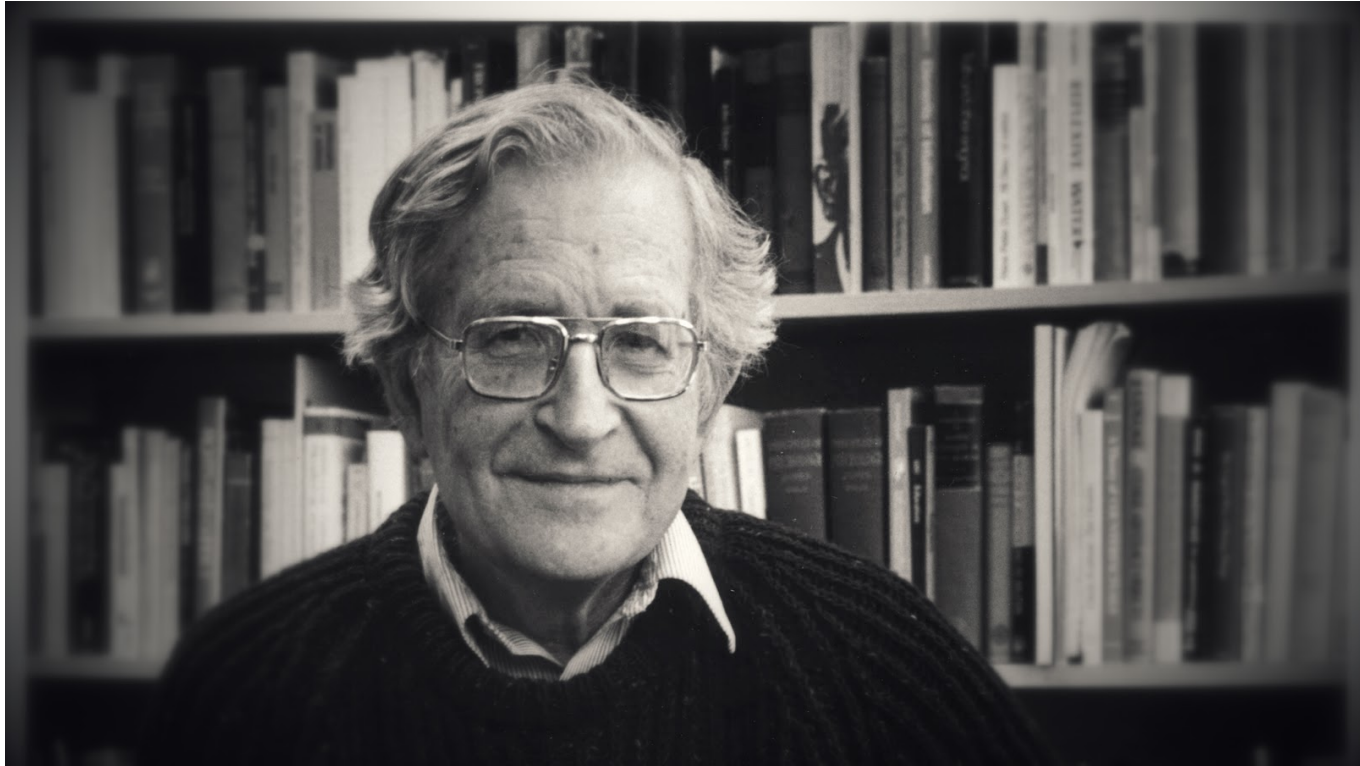


Figure 2.1: [Noam Chomsky](#) proposed the generative grammar in 1950.

Chapter 3

Regular expressions

3.1	Definition of the regular expressions	46
3.2	Some properties of regular expressions	50

Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.

Jamie Zawinski

3.1 Definition of the regular expressions

Regular expressions constitute a language representation method, that can only represent regular languages.

Definition 3.1.1. Set of regular expressions (\mathcal{R})

Given $\Sigma_{\mathcal{R}} = \Sigma \cup \{), (, \emptyset, +, ^*\}$

$$\mathcal{R} \subseteq \Sigma_{\mathcal{R}}^*$$

verifying

1. $\emptyset \in \mathcal{R}$
2. $a \in \mathcal{R}, \forall a \in \Sigma$
3. $(\alpha\beta) \in \mathcal{R}, \forall \alpha, \beta \in \mathcal{R}$
4. $(\alpha + \beta) \in \mathcal{R}, \forall \alpha, \beta \in \mathcal{R}$
5. $\alpha^* \in \mathcal{R}, \forall \alpha \in \mathcal{R}$
6. No other string over $\Sigma_{\mathcal{R}}$ (that cannot be generated with the previous rules) is an element of \mathcal{R} .

Definition 3.1.2. Language defined by a regular expression

$$\begin{aligned}\mathcal{L} : \mathcal{R} &\rightarrow 2^{\Sigma^*} \\ r &\rightarrow \mathcal{L}(r)\end{aligned}$$

verifying

1. $\mathcal{L}(\emptyset) = \emptyset$
2. $\mathcal{L}(a) = \{a\}, \forall a \in \Sigma$
3. $\mathcal{L}((\alpha\beta)) = \mathcal{L}(\alpha) \cdot \mathcal{L}(\beta), \forall \alpha, \beta \in \mathcal{R}$
4. $\mathcal{L}((\alpha + \beta)) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta), \forall \alpha, \beta \in \mathcal{R}$
5. $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*, \forall \alpha \in \mathcal{R}$

Remark.

\cup, \cdot and $*$ are the operations of union, concatenation and Kleene's star for languages.

From now on, we will assume that a regular expression is, actually, the language that it represents, i.e. $(10)^* = \mathcal{L}((10)^*) = \{\varepsilon, 10, 1010, \dots\}$

Notational conventions

Removing the brackets

1. Pairs of brackets can be replaced by square brackets for readability.
2. First and last brackets of a regular expression can be removed.
3. To avoid cluttering of parenthesis, we adopt the following conventions.

Order of operations (*orden de prioridad*): $* > \cdot > +$
 $(R + ((S^*)T)) = R + S^*T$

Associativity:

$$(R + (S + T)) = ((R + S) + T) = R + S + T$$
$$(R(ST)) = ((RS)T) = RST$$

Example 3.1.1. Languages and their representation with regular expressions

Strings that have 001 as a substring

$$(0 + 1)^*001(0 + 1)^*$$

Strings where the number of 1s is a multiple of 3

$$0^* + (0^*10^*10^*10^*)^*$$

Strings that consist of alternating 0s and 1s

$$(10)^* + (01)^* + 0(10)^* + 1(01)^* = (\varepsilon + 1)(01)^*(\varepsilon + 0)$$

Strings that do not have two consecutive 0s

$$(0 + \varepsilon)(1 + 10)^*$$

3.2 Some properties of regular expressions

Any $\alpha, \beta, \gamma \in \mathcal{R}$ verify the following properties:

$$1) \alpha + \emptyset = \alpha$$

$$2) \alpha + \alpha = \alpha$$

$$3) \alpha + \beta = \beta + \alpha$$

$$4) \alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$$

$$5) \alpha(\beta\gamma) = (\alpha\beta)\gamma$$

$$6) \alpha\varepsilon = \varepsilon\alpha = \alpha$$

$$7) (\alpha + \beta)\gamma = \alpha\gamma + \beta\gamma$$

$$8) \gamma(\alpha + \beta) = \gamma\alpha + \gamma\beta$$

$$9) \alpha^*\alpha^* = \alpha^*$$

$$10) (\alpha^*)^* = \alpha^*$$

$$11) \alpha\alpha^* = \alpha^*\alpha$$

$$12) \alpha^* = \varepsilon + \alpha + \alpha^2 + \alpha^3 + \dots + \alpha^k + \alpha^{k+1}\alpha^* \quad \forall k \geq 0$$

$$13) \alpha^* = \varepsilon + \alpha\alpha^*$$

$$14) (\alpha^* + \beta^*)^* = (\alpha + \beta)^*$$

$$15) (\alpha^*\beta^*)^* = (\alpha + \beta)^*$$

$$16) (\alpha\beta)^*\alpha = \alpha(\beta\alpha)^*$$

$$17) (\alpha^*\beta)^*\alpha^* = (\alpha + \beta)^*$$

$$18) \alpha^*(\beta\alpha^*)^* = (\alpha + \beta)^*$$

$$19) (\alpha^*\beta)^* = (\alpha + \beta)^*\beta + \varepsilon$$

$$20) f(\alpha_1, \alpha_2, \dots, \alpha_n) \subseteq (\alpha_1 + \alpha_2 + \dots + \alpha_n)^* \text{ where the images obtained by } f \text{ are combinations of } \alpha_1, \alpha_2, \dots, \alpha_{n-1} \text{ and } \alpha_n, \text{ using only the operations of union } (+), \text{ concatenation } (\cdot) \text{ and Kleene star } (^*)$$



Figure 3.1: [Stephen Kleene](#) proposed the regular expressions model in 1951.

Chapter 4

Finite automata

4.1	Deterministic finite automata	54
4.2	Nondeterministic finite automata	60
4.3	Minimum deterministic finite automaton	66
4.4	Equivalences	66

4.1 Deterministic finite automata

Definition 4.1.1. Deterministic finite automaton (*autómata finito determinista*)

A deterministic finite automaton (DFA) is a 5-tuple $(K, \Sigma, \delta, s, F)$, where

K is a non-empty set of states

Σ is an alphabet

$s \in K$ is the initial state

$F \subseteq K$ is a set of final states

$\delta : K \times \Sigma \rightarrow K$ is the transition function

Definition 4.1.2. Configuration of a DFA (*configuración*)

A configuration of $(K, \Sigma, \delta, s, F)$ is any $(q, w) \in K \times \Sigma^*$.

Remark. q represents the current state of the DFA, and w is the string that is still to be processed by the DFA.

Definition 4.1.3. Initial configuration of a DFA

An initial configuration of $(K, \Sigma, \delta, s, F)$ is any pair (s, w) , with $w \in \Sigma^*$.

Definition 4.1.4. Final configuration of a DFA

A final configuration of $(K, \Sigma, \delta, s, F)$ is any pair (q, ε) , with $q \in K$.

Definition 4.1.5. Direct transition of a DFA (*transición directa*)

$(K, \Sigma, \delta, s, F)$ performs a direct transition from configuration (q, w) to (q', w') , noted $(q, w) \vdash (q', w')$, iff

$$\exists \sigma \in \Sigma : (w = \sigma w') \wedge (\delta(q, \sigma) = q')$$

Remark. A direct transition of a DFA is the function

$$\begin{aligned} \vdash : K \times \Sigma^+ &\rightarrow K \times \Sigma^* \\ (q, \sigma w') &\rightarrow (\delta(q, \sigma), w') \end{aligned}$$

Definition 4.1.6. Transiting in n steps (*transitar*)

Let (q, w) and (q', w') be configurations of a DFA.

We say that (q, w) transits in n steps to (q', w') , noted $(q, w) \vdash^n (q', w')$, with $n > 1$, iff $\exists C_1, C_2, \dots, C_{n-1}$ configurations of that DFA such that

$$(q, w) \vdash C_1, C_1 \vdash C_2, \dots, C_{n-1} \vdash (q', w')$$

We say that (q, w) transits in 1 step to (q', w') , noted $(q, w) \vdash^1 (q', w')$, iff

$$(q, w) \vdash (q', w')$$

We say that (q, w) transits in 0 step to (q', w') , noted $(q, w) \vdash^0 (q', w')$, iff

$$(q, w) = (q', w')$$

Definition 4.1.7. Transiting in at least one step

Let (q, w) and (q', w') be configurations of a DFA.

We say that (q, w) transits in at least one step to (q', w') , noted $(q, w) \vdash^+ (q', w')$, iff

$$\exists n > 0 : (q, w) \vdash^n (q', w')$$

Definition 4.1.8. Transiting

Let (q, w) and (q', w') be configurations of a DFA.

We say that (q, w) transits to (q', w') , noted $(q, w) \vdash^* (q', w')$, iff

$$\exists n \geq 0 : (q, w) \vdash^n (q', w')$$

Definition 4.1.9. Computation. Length of a computation

Let M be a DFA, a computation of M is any sequence of configurations $C_0, C_1, C_2, \dots, C_n$ such that $C_i \vdash C_{i+1} \forall i \in \{0, \dots, n-1\}$ with $n \geq 0$.

It will be noted as $C_0 \vdash C_1 \vdash C_2 \vdash \dots \vdash C_n$, and the length of the computation is n .

Definition 4.1.10. Well started/terminated/complete computation

A well started computation is any computation in which its first configuration is an initial configuration.

A terminated computation is any computation in which its last configuration is a final configuration.

A computation is complete if it is well started and terminated.

Definition 4.1.11. Inaccessible state

$q \in K$ is an inaccessible state of $(K, \Sigma, s, F, \delta)$ iff $\nexists x \in \Sigma^* : (s, x) \vdash^* (q, \varepsilon)$

Definition 4.1.12. Accepted string

Let $M = (K, \Sigma, \delta, s, F)$ and $w \in \Sigma^*$. w is accepted by M iff

$$\exists q \in F : (s, w) \vdash^* (q, \varepsilon).$$

Definition 4.1.13. Accepted language

$$\mathcal{L}(M) = \{w \in \Sigma^* : w \text{ is accepted by } M\}$$

Definition 4.1.14. Set of languages accepted by DFA

The set of the languages accepted by DFA, denoted as $\mathcal{L}(DFA)$, is defined as:

$$\mathcal{L}(DFA) = \{L : \exists M \text{ DFA}, \mathcal{L}(M) = L\}$$

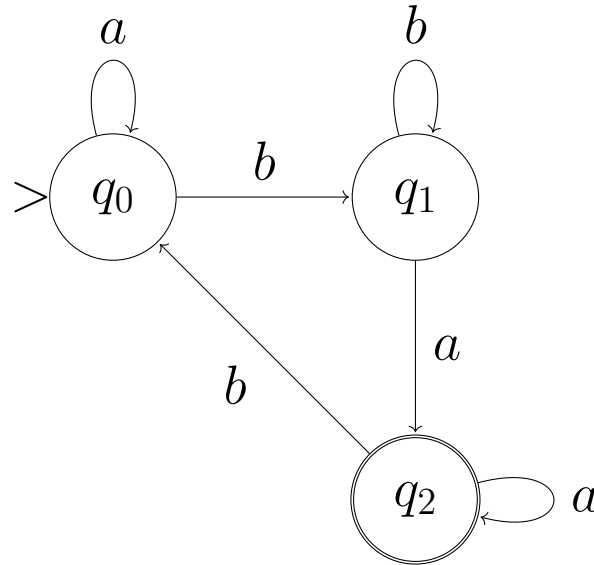
Definition 4.1.15. States diagram

The state diagram of a DFA is a directed graph in which each node represents a state, and each edge represents a transition for a certain input between the states the edge connects (i.e. if $\delta(q, \sigma) = q'$ then there will be an edge from q to q' labeled as σ).

Final states are represented with double circles and the initial state is indicated with " \triangleright ".

Example 4.1.1. Let $M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$ be a DFA with:

$\delta(q, \sigma)$	a	b
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_0



4.2 Nondeterministic finite automata

Nondeterministic finite automaton can transit from a given state to none, one, or more than one state, consuming zero (ε), one, or more than one symbol.

Definition 4.2.1. Nondeterministic finite automaton

A nondeterministic finite automaton (NFA) is a 5-tuple $(K, \Sigma, \Delta, s, F)$, where

K is a non-empty set of states

Σ is an alphabet

$s \in K$ is the initial state

$F \subseteq K$ is a set of final states

$\Delta \subseteq K \times \Sigma^* \times K$ is a transition relation

Definition 4.2.2. Configuration of an NFA

A configuration of $(K, \Sigma, \Delta, s, F)$ is any $(q, w) \in K \times \Sigma^*$.

Definition 4.2.3. Initial configuration of an NFA

An initial configuration of $(K, \Sigma, \Delta, s, F)$ is any pair (s, w) , with $w \in \Sigma^*$.

Definition 4.2.4. Final configuration of an NFA

A final configuration of $(K, \Sigma, \Delta, s, F)$ is any pair (q, ε) , with $q \in K$.

Definition 4.2.5. Direct transition of an NFA

An NFA $(K, \Sigma, \Delta, s, F)$ performs a direct transition from configuration (q, w) to configuration (q', w') , noted $(q, w) \vdash (q', w')$, iff

$$\exists u \in \Sigma^* : (w = uw') \wedge ((q, u, q') \in \Delta)$$

Remark. Transiting directly is not a function since for some configurations (q, w) there can be more than one or none configuration (q', w') such that $(q, w) \vdash (q', w')$.

Definition 4.2.6. Blocked configuration

A blocked configuration of an NFA $M = (K, \Sigma, \Delta, s, F)$ is any non final configuration (q, w) that verifies that $\nexists (q', w') \in K \times \Sigma^* : (q, w) \vdash (q', w')$.

Definition 4.2.7. Transiting in n steps

Let (q, w) and (q', w') be configurations of an NFA.

We say that (q, w) transits in n steps to (q', w') , noted $(q, w) \vdash^n (q', w')$, with $n > 1$, iff $\exists C_1, C_2, \dots, C_{n-1}$ configurations of that NFA :

$$(q, w) \vdash C_1, C_1 \vdash C_2, \dots, C_{n-1} \vdash (q', w')$$

We say that (q, w) transits in 1 step to (q', w') , noted $(q, w) \vdash^1 (q', w')$, iff

$$(q, w) \vdash (q', w')$$

We say that (q, w) transits in 0 step to (q', w') , noted $(q, w) \vdash^0 (q', w')$, iff

$$(q, w) = (q', w')$$

Definition 4.2.8. Transiting in at least one step

Let (q, w) and (q', w') be configurations of an NFA.

We say that (q, w) transits in at least one step to (q', w') , noted $(q, w) \vdash^+ (q', w')$, iff

$$\exists n > 0 : (q, w) \vdash^n (q', w')$$

Definition 4.2.9. Transiting

Let (q, w) and (q', w') be configurations of an NFA.

We say that (q, w) transits to (q', w') , noted $(q, w) \vdash^* (q', w')$, iff

$$\exists n \geq 0 : (q, w) \vdash^n (q', w')$$

Definition 4.2.10. Computation. Length of a computation

Let M be an NFA, a computation of M is any sequence of configurations $C_0, C_1, C_2, \dots, C_n$ such as $C_i \vdash C_{i+1} \forall i \in \{0, \dots, n-1\}$ with $n \geq 0$.

It will be noted as $C_0 \vdash C_1 \vdash C_2 \vdash \dots \vdash C_n$, and the length of the computation is n .

Definition 4.2.11. Well started/terminated/complete/blocked computation

A well started computation is any computation in which its first configuration is an initial configuration.

A terminated computation is any computation in which its last configuration is a final configuration.

A computation is complete if it is well started and terminated.

A computation is blocked if its last configuration is blocked.

Definition 4.2.12. Inaccessible state

$q \in K$ is an inaccessible state $(K, \Sigma, s, F, \Delta)$ iff $\nexists x \in \Sigma^* : (s, x) \vdash^* (q, \varepsilon)$

Definition 4.2.13. Accepted string

Let $M = (K, \Sigma, \Delta, s, F)$ an NFA and $w \in \Sigma^*$. w is accepted by M iff $\exists q \in F$: $(s, w) \vdash^* (q, \varepsilon)$.

Definition 4.2.14. Accepted language

Let M an NFA, the language accepted by M is defined as:

$$\mathcal{L}(M) = \{w \in \Sigma^* : w \text{ is accepted by } M\}$$

Definition 4.2.15. Set of languages accepted by NFA

The set of the languages accepted by NFA is defined as:

$$\mathcal{L}(NFA) = \{L : \exists M \text{ NFA}, \mathcal{L}(M) = L\}$$

Definition 4.2.16. States diagram

The state diagram of an NFA is a directed graph in which each node represents a state, and each edge represents a transition for a certain input between the states the edge connects (i.e. if $(q, w, q') \in \Delta$ then there will be an edge from q to q' labeled as w). Final states are represented with double circles and the initial state is indicated with " \triangleright ".

Example 4.2.1. Let $M = (K, \Sigma, \Delta, s, F)$ be an NFA with:

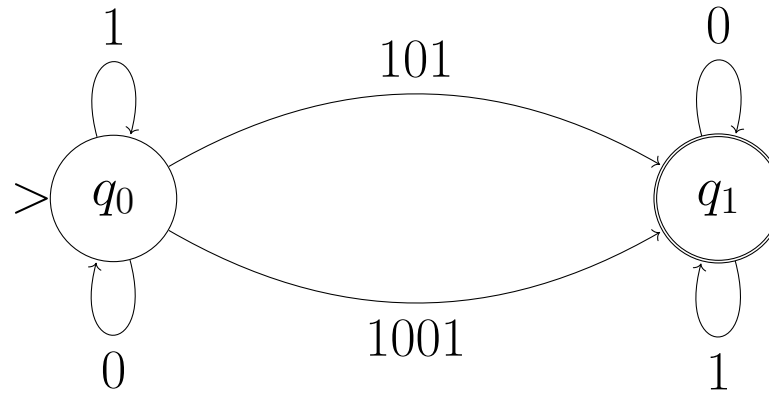
$$K = \{q_0, q_1\}$$

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

$$\Delta = \{(q_0, 0, q_0), (q_0, 1, q_0), (q_0, 101, q_1), (q_0, 1001, q_1), (q_1, 0, q_1), (q_1, 1, q_1)\}$$

$$s = q_0$$

$$F = \{q_1\}$$



4.3 Minimum deterministic finite automaton

Definition 4.3.1. Equivalent finite automata ($M_1 \equiv M_2$)

Two finite automata, M_1 and M_2 are equivalent iff:

$$L(M_1) = L(M_2)$$

Definition 4.3.2. Minimum deterministic finite automaton (MDFA)

For a DFA $M = (K, \Sigma, \delta, s, F)$, M is a MDFA iff it verifies:

$$M' \equiv M \Rightarrow |K'| \geq |K|, \forall M' = (K', \Sigma, \delta', s', F')$$

Proposition 4.3.1. $\forall M$ DFA $\exists M'$ MDFA : $M \equiv M'$

4.4 Equivalences

Definition 4.4.1. Set of the languages represented by finite automata

$$\mathcal{L}(\mathcal{F}) = \mathcal{L}(DFA) = \mathcal{L}(NFA)$$

Theorem 4.4.1. $L \in \mathcal{L}(\mathcal{R}) \Leftrightarrow L \in \mathcal{L}(\mathcal{F})$

Theorem 4.4.2. $L \in \mathcal{L}.3 \Leftrightarrow L \in \mathcal{L}(\mathcal{F})$

Corollary 4.4.1. $\mathcal{L}.3 = \mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{F})$

Chapter 5

Regularity conditions

5.1	Myhill-Nerode	68
5.2	Pumping lemma	72

5.1 Myhill-Nerode

Definition 5.1.1. Indistinguishable strings (*cadena indistinguibles*)

$$x \approx_L y \Leftrightarrow \forall z \in \Sigma^* (xz \in L \wedge yz \in L) \vee (xz \notin L \wedge yz \notin L)$$

If the condition meets, $x, y \in \Sigma^*$ are indistinguishable strings with respect to $L \subseteq \Sigma^*$.

Remark. Conversely, $x, y \in \Sigma^*$ are distinguishable strings with respect to $L \subseteq \Sigma^*$ iff

$$x \not\approx_L y \Leftrightarrow \exists z \in \Sigma^* : (xz \in L \wedge yz \notin L) \vee (xz \notin L \wedge yz \in L)$$

Definition 5.1.2. Indistinguishability relation (*relación de indistinguibilidad*)

$$I_L = \{(x, y) \in \Sigma^* \times \Sigma^* : x \approx_L y\}$$

Proposition 5.1.1. I_L is an equivalence relation over Σ^* .

Remark. Since I_L is an equivalence relation, it establishes a partition over Σ^* where each element is an equivalence class, this partition is noted Π_L .

Theorem 5.1.1. Myhill-Nerode theorem: $L \in \mathcal{L}.3 \Leftrightarrow |\Pi_L| \in \mathbb{N}$

Proof. It has to be proven in both directions.

a) $L \in \mathcal{L}.3 \Rightarrow |\Pi_L| \in \mathbb{N}$

$L \in \mathcal{L}.3 \Rightarrow L \in \mathcal{L}(\text{AFD}) \Rightarrow \exists M \text{ AFD without inaccessible states} : \mathcal{L}(M) = L.$

Let $M = (K = \{q_0, \dots, q_n\}, \Sigma, \delta, q_0, F)$ and $\delta^* : K \times \Sigma^* \rightarrow K$ defined as

$$\delta^*(q_i, x) = q_j \Leftrightarrow (q_i, x) \vdash^* (q_j, \varepsilon)$$

Let $A_i = \{x \in \Sigma^* : \delta^*(q_0, x) = q_i\}$ with $i = 0, \dots, n$, and $\Pi = \{A_0, \dots, A_n\}$.

M is a DFA, so Π is a partition of Σ^* and

$$\forall x, y \in A_i \ (x, y) \in I_L, i = 0, \dots, n$$

Let $\Pi' = \{B \subseteq \Sigma^* : B = A_{i_1} \cup \dots \cup A_{i_k}, A_{i_j} \in \Pi \quad \wedge$

$$\forall x, y \in B \ (x, y) \in I_L \quad \wedge$$

$$(x \in B \wedge y \notin B) \Rightarrow (x, y) \notin I_L\}$$

By definition, we have that Π' is a partition of Σ^* , meeting the criteria:

$$(\Pi' = \Pi_L \wedge |\Pi'| \leq |\Pi| = n + 1) \Rightarrow |\Pi_L| \leq n + 1 \Rightarrow |\Pi_L| \in \mathbb{N}$$

b) $|\Pi_L| \in \mathbb{N} \Rightarrow L \in \mathcal{L}.3$

$$|\Pi_L| \in \mathbb{N} \Rightarrow |\Pi_L| = m \geq 1 \Rightarrow \Pi_L = \{A_0, \dots, A_{m-1}\}$$

Let $\Sigma = \{a_1, \dots, a_n\}$ the alphabet over which L is defined.

Let $M = (K, \Sigma, \delta, s, F)$ be a DFA with:

$$K = \{q_0, \dots, q_{m-1}\}$$

$$\Sigma = \{a_1, \dots, a_n\}$$

$$\delta(q_i, a_k) = q_j \Leftrightarrow x \in A_i \Rightarrow xa_k \in A_j$$

$$s = q_j \Leftrightarrow \varepsilon \in A_j$$

$$F = \{q_i \in K : x \in A_i \Rightarrow x \in L\}$$

Constructively we have that $\mathcal{L}(M) = L \Rightarrow L \in \mathcal{L}.3$

□

Remark.

This is a necessary and sufficient condition for a language to be regular.

M is a *MDFA* that represents L .

If infinite strings that are distinguishable with respect to L can be found, then L is not a regular language.

Example 5.1.1. Determine that $L = \{0^n 1^n : n \in \mathbb{N}\} \notin \mathcal{L}$.3

Given $S = \{0^n : n \in \mathbb{N}\}$, any two strings in S , say 0^k and 0^l (with $k \neq l$), are distinguishable with respect to L , since $(0^k 1^k \in L) \wedge (0^l 1^k \notin L)$.

5.2 Pumping lemma

Theorem 5.2.1. Pumping (*bombeo*)

Let $(K, \Sigma, \delta, s, F)$ DFA with $|K| = n$, and $L = \mathcal{L}(M)$.

Then, $\forall x \in L : |x| \geq n, \exists u, v, w \in \Sigma^* :$

1. $x = uvw$
2. $|uv| \leq n$
3. $|v| > 0$
4. $\forall m \geq 0 \ uv^m w \in L$

Definition 5.2.1. Regular pumping condition (RPC)

L verifies the RPC iff $\exists n \in \mathbb{N} : \forall x \in L, |x| \geq n, \exists u, v, w \in \Sigma^*$ with

1. $x = uvw$
2. $|uv| \leq n$
3. $|v| > 0$
4. $\forall m \geq 0 \ uv^m w \in L$

Pumping lemma for $\mathcal{L}.3$:

$L \in \mathcal{L}.3 \Rightarrow L$ verifies the regular pumping condition.

Example 5.2.1. Determine that $L = \{0^n 1^n : n \in \mathbb{N}\} \notin \mathcal{L}$.3

It must be shown that

$\forall n \in \mathbb{N} \exists x \in L \ |x| \geq n : \forall u, v, w \in \Sigma^*$ they do not meet the RPC.

Let $x \in L$ and $x = 0^n 1^n \in L$, then $|x| = 2n \geq n$. Now, let assume that $\exists u, v, w \in \Sigma^* : \text{they verify the RPC. Then}$

1. $x = uvw$
2. $|uv| \leq n \xRightarrow{1} uv = 0^k, k \leq n \xRightarrow{1} w = 0^{n-k} 1^n$
3. $|v| > 0 \xRightarrow{2} v = 0^j, 0 < j < k$
4. $\forall m \geq 0 \ uv^m w \in L \Rightarrow \forall m > 1 \ uv^m w \in L$

But we can see that

$$uv^m w = (uv)v^{m-1}w = 0^k(0^j)^{m-1}0^{n-k}1^n = 0^{k+j(m-1)+(n-k)}1^n \notin L$$

because $j(m-1) > 0$, since $j > 0$ and $m > 1$.

This means that any selection of u, v, w will not meet the RPC, and then, the language is not regular.

Chapter 6

Context-free languages

6.1	Derivations and Ambiguity	76
6.2	Recursion	81
6.3	CFG Simplification	83
6.4	Normal forms	85
6.5	Closure properties	85
6.6	Non-Deterministic Pushdown Automata	86
6.7	Pumping	91

6.1 Derivations and Ambiguity

Definition 6.1.1. Derivation A -subtree (*Subárbol- A de derivación*)

For a context free grammar $G = (N, T, P, S)$, a tree is a derivation A -subtree associated to G , with $A \in N$, if:

1. Each node is labeled with a symbol from V .
2. The root is labeled as A
3. If a node is labeled as B and its children nodes from left to right are labeled as X_1, X_2, \dots, X_k , then:

$$B \rightarrow X_1 X_2 \dots X_k \in P$$

Definition 6.1.2. Product of a derivation A -subtree

The product of a derivation A -subtree, is the string obtained from reading its leaves from left to right.

Definition 6.1.3. A -derivation

An A -derivation associated to a derivation A -subtree is the finite sequence of direct productions starting from A used to obtained the A -subtree.

Definition 6.1.4. Derivation tree

A derivation tree is a derivation S -subtree (S is the grammar's axiom)

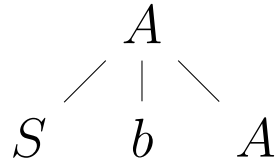
Definition 6.1.5. Derivation

A derivation is an S -derivation (S is the grammar's axiom).

Definition 6.1.6. Left-most derivation. Right-most derivation

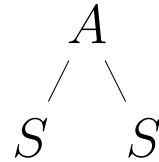
If in every step of a derivation it is applied a rule of production to the non terminal character most to the left/right then the derivation is a left/right-most derivation.

Example 6.1.1. Let $G = (\{S, A\}, \{a, b\}, \{S \rightarrow aAS \mid a, A \rightarrow SbA \mid SS \mid ba\}, S)$. These are some examples of derivation A -subtrees, their products and A -derivations:



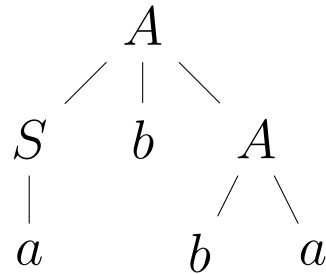
Product: SbA

A -derivation: $A \Rightarrow SbA$



Product: SS

A -derivation: $A \Rightarrow SS$



Product: $abba$

A -derivations:

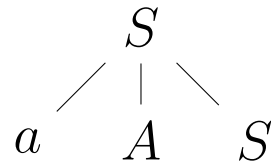
$A \Rightarrow Sb\mathbf{A} \Rightarrow \mathbf{S}bba \Rightarrow abba$

$A \Rightarrow \mathbf{S}bA \Rightarrow ab\mathbf{A} \Rightarrow abba$

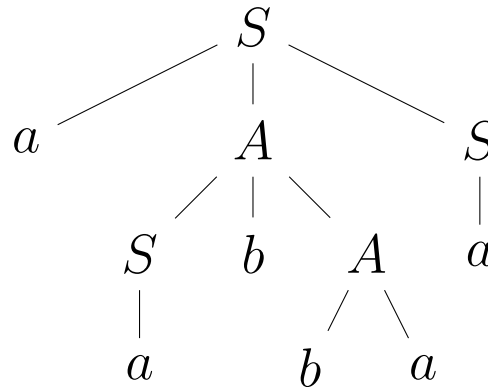
These are examples of derivation trees (S -subtrees), their products and derivations:



Product: a Derivation: $S \Rightarrow a$



Product: aAS Derivation: $S \Rightarrow aAS$



Product: $aabbaa$

Derivations:

$$S \Rightarrow a\mathbf{A}S \Rightarrow aSb\mathbf{A}S \Rightarrow aSbba\mathbf{S} \Rightarrow a\mathbf{S}bbaa \Rightarrow aabbaa$$

$$S \Rightarrow a\mathbf{A}S \Rightarrow a\mathbf{S}bAS \Rightarrow aab\mathbf{A}S \Rightarrow aabba\mathbf{S} \Rightarrow aabbaa \text{ (left-most derivation)}$$

$$S \Rightarrow aA\mathbf{S} \Rightarrow a\mathbf{A}a \Rightarrow aSb\mathbf{A}a \Rightarrow a\mathbf{S}bbaa \Rightarrow aabbaa \text{ (right-most derivation)}$$

Definition 6.1.7. Ambiguous grammar

Let G be a CFG. It is said that G is ambiguous iff

$\exists w \in \mathcal{L}(G) : w$ is a product of more than one derivation tree.

Remark. Ambiguity is an undecidable property, i.e. there is no conclusive algorithm that for any given CFG it can answer in a finite time whether such CFG is ambiguous or not.

Example 6.1.2. Let $G = (\{S, A, B\}, \{a\}, \{S \rightarrow A \mid B, A \rightarrow a, B \rightarrow a\}, S)$.

$$\begin{array}{cc} S & S \\ | & | \\ A & B \\ | & | \\ a & a \end{array}$$

The string a is the product of two distinct trees, which means that G is ambiguous.

Definition 6.1.8. Inherently ambiguous language (*lenguaje inherentemente ambiguo*)

A context-free language L is said to be inherently ambiguous iff:

$$L = \mathcal{L}(G) \Rightarrow G \text{ is ambiguous}$$

6.2 Recursion

Definition 6.2.1. Recursion (*recursividad*)

A CFG (N, T, P, S) is directly left-recursive iff:

$$\exists A \in N : A \Rightarrow A\alpha, \text{ with } \alpha \in V^*.$$

G is directly right-recursive iff:

$$\exists A \in N : A \Rightarrow \alpha A, \text{ with } \alpha \in V^*.$$

G is left-recursive iff:

$$\exists A \in N : A \Rightarrow^+ A\alpha, \text{ with } \alpha \in V^*.$$

G is right-recursive iff:

$$\exists A \in N : A \Rightarrow^+ \alpha A, \text{ with } \alpha \in V^*.$$

G is recursive iff:

$$\exists A \in N : A \Rightarrow^+ \alpha A\beta, \text{ with } \alpha, \beta \in V^*.$$

Proposition 6.2.1. Let G be a CFG, therefore there is a G' CFG that is not directly left-recursive such that $G \equiv G'$.

Proposition 6.2.2. Let G be a CFG, therefore there is a G' CFG that is not left-recursive such that $G \equiv G'$.

6.3 CFG Simplification

Definition 6.3.1. Useful symbol, useless symbol (*símbolo útil*)

Let $G = (N, T, P, S)$ be a CFG and let $X \in V$. X is an useful symbol iff

$$\exists \alpha, \beta \in V^* : S \Rightarrow^* \alpha X \beta \Rightarrow^* w \in T^+$$

Otherwise, X is said to be a useless symbol.

Definition 6.3.2. Generating symbol, non-generating symbol (*símbolo terminable*)

X is a generating symbol iff

$$\exists w \in T^+ : X \Rightarrow^* w$$

Otherwise X is a non-generating symbol.

Definition 6.3.3. Reachable symbol, unreachable symbol (*símbolo alcanzable*)

X is a reachable symbol iff

$$\exists \alpha, \beta \in V^* : S \Rightarrow^* \alpha X \beta$$

Otherwise, X is an unreachable symbol.

Remark. X being both generating and reachable does not imply that it is useful.

Definition 6.3.4. Unit production (*regla unitaria*)

A unit production is any rule of the form $A \rightarrow B : A, B \in N$.

Definition 6.3.5. Proper grammar (*gramática propia*)

A CFG is said to be proper iff it is not left-recursive and it does not have useless symbols.

Proposition 6.3.1. Let G be a CFG that generates a language different than the empty language, therefore there exists a proper CFG G' such that $G \equiv G'$

Proposition 6.3.2. Let G be a CFG, therefore there exists a CFG G' without unit productions such that $G \equiv G'$.

6.4 Normal forms

Definition 6.4.1. Chomsky normal form (*forma normal de Chomsky*)

Let $G = (N, T, P, S)$ be a CFG. G is in Chomsky normal form (CNF) iff

$$\forall (A \rightarrow \alpha) \in P \quad \alpha = BC \vee \alpha = a \quad \text{with } B, C \in N \text{ and } a \in T$$

Proposition 6.4.1. $\forall G \text{ CFG} \quad \exists G' \text{ in CNF} : G \equiv G'$.

Definition 6.4.2. Greibach normal form (*forma normal de Greibach*)

Let $G = (N, T, P, S)$ be a CFG. G is in Greibach normal form (GNF) iff

$$\forall (A \rightarrow \alpha) \in P \quad \alpha = a\beta \quad \text{with } a \in T \text{ and } \beta \in N^*$$

Proposition 6.4.2. $\forall G \text{ CFG} \quad \exists G' \text{ in GNF} : G \equiv G'$

6.5 Closure properties

Theorem 6.5.1. \mathcal{L}_2 is closed for the operations of union, concatenation, and Kleene star.

Proposition 6.5.1. \mathcal{L}_2 is not closed for the operations of intersection and complement.

6.6 Non-Deterministic Pushdown Automata

Definition 6.6.1. Non-deterministic pushdown automaton (*autómata con pila*)

A non-deterministic pushdown automaton (NDPA) is a sextuple $M = (K, \Sigma, \Gamma, \Delta, s, F)$ where:

K is a finite set of states.

Σ is an alphabet (input symbols).

Γ is an alphabet (stack symbols).

$s \in K$ is the start state.

$F \subseteq K$ is the set of final states.

Δ is a transition relation defined as a subset of $(K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$.

Definition 6.6.2. Transition.

Let $M = (K, \Sigma, \Gamma, \Delta, s, F)$ be a NDPA. Each pair $((p, u, \beta), (q, \gamma)) \in \Delta$ is called a transition of M .

Hence, Δ is a finite set of transitions.

Definition 6.6.3. Configuration.

C is a configuration of $M = (K, \Sigma, \Gamma, \Delta, s, F)$ iff $C \in K \times \Sigma^* \times \Gamma^*$.

Definition 6.6.4. Initial configuration.

Given a NDPA $M = (K, \Sigma, \Gamma, \Delta, s, F)$, an initial configuration of M is defined as any configuration (s, w, ε) .

Definition 6.6.5. Final configuration.

Given a NPDA $M = (K, \Sigma, \Gamma, \Delta, s, F)$, a final configuration of M is defined as any configuration $(q, \varepsilon, \varepsilon)$.

Definition 6.6.6. Blocked configuration.

Let $M = (K, \Sigma, \Gamma, \Delta, s, F)$ be a NDPA and (q, u, α) a configuration of M . (q, u, α) is a blocked configuration if it is not final and

$$\nexists (q', u', \alpha') \in K \times \Sigma^* \times \Gamma^* : (q, u, \alpha) \vdash (q', u', \alpha')$$

Definition 6.6.7. Direct transition.

Let (q, u, α) and (q', u', α') be two configurations of a NPDA $M = (K, \Sigma, \Gamma, \Delta, s, F)$.
 (q, u, α) directly transits to (q', u', α') , noted as $(q, u, \alpha) \vdash (q', u', \alpha')$, iff

$$\begin{aligned} & \exists v \in \Sigma^* \text{ and } \beta, \beta', \gamma \in \Gamma^* : \\ & (u = vu') \wedge (\alpha = \beta\gamma) \wedge (\alpha' = \beta'\gamma) \wedge ((q, v, \beta), (q', \beta')) \in \Delta \end{aligned}$$

Definition 6.6.8. Transition in n steps.

Let M be a NDPA and (q, u, α) and (q', u', α') configurations of M .
 (q, u, α) transits in n steps to (q', u', α') , $n > 1$, noted as $(q, u, \alpha) \vdash^n (q', u', \alpha')$, iff

$$\exists C_1, \dots, C_{n-1} : (q, u, \alpha) \vdash C_1, C_1 \vdash C_2, \dots, C_{n-1} \vdash (q', u', \alpha')$$

Definition 6.6.9. To transit in at least one step.

Let M be a NDPA and (q, u, α) and (q', u', α') configurations of M .
 (q, u, α) transits in at least one step to (q', u', α') , noted as $(q, u, \alpha) \vdash^+ (q', u', \alpha')$, iff

$$\exists n > 0 : (q, u, \alpha) \vdash^n (q', u', \alpha')$$

Definition 6.6.10. To transit

Let M be a NDPA and (q, u, α) and (q', u', α') configurations of M .
 (q, u, α) transits to (q', u', α') , noted as $(q, u, \alpha) \vdash^* (q', u', \alpha')$, iff

$$\exists n \geq 0 : (q, u, \alpha) \vdash^n (q', u', \alpha')$$

Definition 6.6.11. Computation and length of a computation

Let M be a NDPA, a computation of M is any finite sequence of configurations $C_0, C_1, \dots, C_n : C_i \vdash C_{i+1}, i \in \{0 \dots n - 1\}$, noted as

$$C_0 \vdash C_1 \vdash \dots \vdash C_n$$

The length of such computation is n .

Definition 6.6.12. Well-initiated, terminal, complete and blocked computations

A well-initiated computation is any computation whose first configuration is initial.

A terminal computation is any computation whose last configuration is final.

A computation is complete when it is both well-initiated and terminal.

A configuration is blocked when its last configuration is a blocking configuration.

Definition 6.6.13. Accepted string.

Let $w \in \Sigma^*$ and M be a NDPA. It is said that w is accepted by M iff

$$\exists q \in F : (s, w, \varepsilon) \vdash^* (q, \varepsilon, \varepsilon).$$

Definition 6.6.14. Accepted language

Let M be a NDPA. The language accepted by M , noted as $\mathcal{L}(M)$ is defined as

$$\mathcal{L}(M) = \{w \in \Sigma^* : w \text{ is accepted by } M\}.$$

Definition 6.6.15. Set of languages accepted by NDPA.

The set of the languages accepted by NDPA, noted as $\mathcal{L}(NDPA)$, is defined as:

$$\mathcal{L}(NDPA) = \{L : \exists M \text{ NDPA with } \mathcal{L}(M) = L\}$$

Proposition 6.6.1. $L \in \mathcal{L}.2 \Leftrightarrow L \in L(NDPA)$

6.7 Pumping

Definition 6.7.1. Context free pumping condition (*condición de bombeo independiente del contexto*)

Let $L \subseteq \Sigma^*$.

L satisfies the context free pumping condition (CFPC) iff

$\exists n \in \mathbb{N} : \forall x \in L \text{ with } |x| \geq n, \exists u, v, w, y, z \in \Sigma^*$ verifying:

1) $x = uvwyz$

2) $|uwy| < n$

3) $|vy| > 0$

4) $\forall m \geq 0 \quad uv^mwy^mz \in L$

Pumping lemma for $\mathcal{L}.2$:

$L \in \mathcal{L}.2 \Rightarrow L$ satisfies the CFPC.

Chapter 7

The Turing machine

7.1	Formal definition	94
7.2	Computability, decidability, enumerability	104

7.1 Formal definition

Definition 7.1.1. Turing Machine (TM)

A TM is a 5-tuple $(K, q_0, \Sigma, \gamma, \delta)$, where

K is a non-empty, finite set of states,

$q_0 \in K$ is the initial state,

Σ is an alphabet, and if $\Sigma_R = \{a_0, l, r, h\}$ are reserved words, then $\Sigma \cap \Sigma_R = \emptyset$,

$\gamma : K \times \Sigma_T \rightarrow \Sigma'$ is the instruction function,

$\delta : K \times \Sigma_T \rightarrow K$ is the transition function

where $\Sigma_T = \Sigma \cup \{a_0\}$, $\Sigma' = \Sigma \cup \Sigma_R$ and a_0 is often referred as blank symbol.

Remark. The instructions set is Σ' , consequently the TM is able to:

- write a symbol of Σ or a blank symbol (a_0) in a cell,
- move its head to the neighbouring left cell (l) or right cell (r),
- halt the computation (h).

If a TM is in a state $q \in K$ and reads $a \in \Sigma$, it will execute the instruction $\gamma(q, a) \in \Sigma'$ and will transit to the state $\delta(q, a) \in K$.

A TM can be represented by a matrix of four columns and $|K| \cdot |\Sigma_T|$ rows.

$$\begin{bmatrix} q_0 & a_0 & \gamma(q_0, a_0) & \delta(q_0, a_0) \\ q_0 & a_1 & \gamma(q_0, a_1) & \delta(q_0, a_1) \\ \vdots & \vdots & \vdots & \vdots \\ q_0 & a_n & \gamma(q_0, a_n) & \delta(q_0, a_n) \\ \\ q_1 & a_0 & \gamma(q_1, a_0) & \delta(q_1, a_0) \\ q_1 & a_1 & \gamma(q_1, a_1) & \delta(q_1, a_1) \\ \vdots & \vdots & \vdots & \vdots \\ q_1 & a_n & \gamma(q_1, a_n) & \delta(q_1, a_n) \\ \\ \vdots & \vdots & \vdots & \vdots \\ q_{|K|-1} & a_0 & \gamma(q_{|K|-1}, a_0) & \delta(q_{|K|-1}, a_0) \\ q_{|K|-1} & a_1 & \gamma(q_{|K|-1}, a_1) & \delta(q_{|K|-1}, a_1) \\ \vdots & \vdots & \vdots & \vdots \\ q_{|K|-1} & a_n & \gamma(q_{|K|-1}, a_n) & \delta(q_{|K|-1}, a_n) \end{bmatrix}$$

Definition 7.1.2. Tape expression of a TM

A tape expression of a TM is any function $E : \mathbb{Z} \rightarrow \Sigma_T$, satisfying

$$|\{z \in \mathbb{Z} : E(z) \neq a_0\}| \in \mathbb{N}$$

Remark. The (infinite) tape contains a finite number of symbols of Σ .

Definition 7.1.3. Configuration of a TM

A configuration of a TM $M = (K, q_0, \Sigma, \gamma, \delta)$ is any $C = (q, E, z)$, with $q \in K$, E is a tape expression of M , and $z \in \mathbb{Z}$.

Remark. q represents the current state of the TM, E represents the current content of the complete tape and z is the cell where M is reading.

Definition 7.1.4. Initial configuration of a TM

An initial configuration of a TM $M = (K, q_0, \Sigma, \gamma, \delta)$ is any configuration (q_0, E, z)

Definition 7.1.5. Terminal and non-terminal configuration of a TM

Given a TM M configuration (q, E, z) , (q, E, z) is terminal $\iff \gamma(q, E(z)) = h$

Otherwise, the configuration would be non-terminal.

Remark. A configuration could be initial and terminal.

Definition 7.1.6. Direct transition

A TM $M = (K, q_0, \Sigma, \gamma, \delta)$ performs a transition from (q, E, z) to (q', E', z') , noted as $(q, E, z) \vdash (q', E', z')$, iff M satisfies all these conditions:

$$1. \delta(q, E(z)) = q'$$

$$(z' = z - 1 \wedge \gamma(q, E(z)) = l) \quad \vee$$

$$2. (z' = z + 1 \wedge \gamma(q, E(z)) = r) \quad \vee$$

$$(z' = z \wedge \gamma(q, E(z)) \in \Sigma_T)$$

$$3. E'(x) = \begin{cases} E(x) & \text{if } x \neq z \\ E(x) & \text{if } x = z \wedge \gamma(q, E(z)) \in \{l, r\} \\ \gamma(q, E(x)) & \text{if } x = z \wedge \gamma(q, E(z)) \in \Sigma_T \end{cases}$$

Definition 7.1.7. To transit in n steps

A TM M transits in n steps from a configuration C to C' , denoted as $C \vdash^n C'$, iff:
 $\exists C_1, C_2, \dots, C_{n-1}$ configurations of M :

$$C \vdash C_1 \vdash C_2 \vdash \dots \vdash C_{n-1} \vdash C'$$

Definition 7.1.8. To transit in one step

A TM M transits in 1 step from C to C' , iff $C \vdash C'$.

Definition 7.1.9. To transit in 0 steps

A TM M transits in 0 steps from C to C' , denoted as $C \vdash^0 C'$, iff $C = C'$.

Definition 7.1.10. To transit in at least one step

A TM M transits in at least one step from C to C' , denoted as $C \vdash^+ C'$, iff:

$$\exists n > 0 \mid C \vdash^n C'$$

Definition 7.1.11. To transit

A TM M transits from C to C' , denoted as $C \vdash^* C'$, iff:

$$\exists n \geq 0 \mid C \vdash^n C'$$

Definition 7.1.12. Computation

We define a computation of a TM, denoted as $C_0 \vdash C_1 \vdash \dots \vdash C_n$, as a finite sequence of configurations C_0, C_1, \dots, C_n , such that $C_i \vdash C_{i+1}$, where $0 \leq i < n$, with $n \geq 0$.

The length of this computation is n .

Definition 7.1.13. Well-started computation

A computation of a TM is well started, if its first configuration is initial.

Definition 7.1.14. Finished computation

A computation of a TM is finished, if its last configuration is terminal.

Definition 7.1.15. Completed computation

A computation of a TM is completed, iff it is well-started and finished.

Definition 7.1.16. Halt after a configuration

Let M be a TM, and C a configuration of M . We say that M halts after C iff:

$$\exists C' \text{ terminal configuration of } M : C \vdash^* C'$$

Remark. Conversely, M does not halt after a configuration C iff

$$\forall n \geq 0, \exists C' \text{ configuration of } M : C \vdash^n C'$$

Definition 7.1.17. Halt over or halt behind a string

Let M be a TM, C a configuration, and $C' = (q, E, z)$, a terminal one, such that $C \vdash^* C'$.

$$E(z) \neq a_0 \quad \Rightarrow \quad M, \text{ after } C, \text{ halts over } w_z$$

$$E(z-1) \neq a_0 \wedge E(z) = a_0 \quad \Rightarrow \quad M, \text{ after } C, \text{ halts behind } w_{z-1}$$

$$\forall n \leq z, E(n) = a_0 \quad \Rightarrow \quad M, \text{ after } C, \text{ halts behind } \varepsilon$$

Definition 7.1.18. Place a TM over a cell

We place a TM M over a cell $z \in \mathbb{Z}$, iff its initial configuration is (q_0, E, z) , where q_0 is its initial state, and E is a tape expression.

Definition 7.1.19. Place a TM behind a string

We place a TM M behind a string w_y , iff we place it over the cell $y + 1$, and

$$E(z) = \begin{cases} a \in \Sigma & \text{if } z \in [x, y] \\ a_0 & \text{otherwise} \end{cases}$$

where $w_y = E(x)E(x+1)\dots E(y)$, and $M = (K, q_0, \Sigma, \gamma, \delta)$.

Definition 7.1.20. Place a TM behind some strings

Let $M = (K, q_0, \Sigma, \gamma, \delta)$ be a TM, and w_1, w_2, \dots, w_n strings over Σ . We say that we place M behind w_1, w_2, \dots, w_n , iff we place it behind $w = w_1a_0w_2a_0\dots a_0w_n$.

Example 7.1.1. The predicate $a > b$ is Turing decidable by a Turing machine defined by the table:

q_0	*	l	q_1	right-most symbol of w_2
q_0	l	l	q_0	void
q_1	*	l	q_2	skip separator
q_1		l	q_1	skip w_2
q_2	*	l	q_2	skip further separators
q_2		*	q_3	erase right-most symbol of $w_1 \Rightarrow w_1 := w_1 - 1$
q_3	*	l	q_4	shift to left to right-most symbol of w_1
q_3			q_3	void
q_4	*	h	q_4	stop and decide $w_1 \leq w_2$
q_4		r	q_5	w_1 not empty yet
q_5	*	r	q_5	skip separators
q_5		*	q_6	erase left-most symbol of $w_2 \Rightarrow w_2 := w_2 - 1$
q_6	*	r	q_7	shift to right to left-most symbol of w_2
q_6			q_6	void
q_7	*		q_8	write as true the value
q_7		l	q_2	w_2 not empty yet
q_8	*	*	q_8	void
q_8		h	q_8	stop and decide $w_1 > w_2$

Placing it behind the strings $w_1 \star w_2$, these are computations that it can perform:

$w_1 > w_2 \Rightarrow E(z) = $	$w_1 = w_2 \Rightarrow E(z) = \star$
$(q_0, \star \star \underline{\star}, 5) \vdash$	$(q_0, \star \star \underline{\star}, 5) \vdash$
$(q_1, \star \star \star, 4) \vdash$	$(q_1, \star \star \star, 4) \vdash$
$(q_2, \star \underline{\star} \star, 3) \vdash$	$(q_2, \star \underline{\star} \star, 3) \vdash$
$(q_2, \star \star \star, 2) \vdash$	$(q_2, \star \star \star, 2) \vdash$
$(q_3, \star \underline{\star} \star \star, 2) \vdash$	$(q_3, \star \underline{\star} \star \star, 2) \vdash$
$(q_4, \star \star \star \star, 1) \vdash$	$(q_4, \underline{\star} \star \star \star, 1) \vdash$
$(q_5, \star \underline{\star} \star \star, 2) \vdash$	
$(q_5, \star \star \underline{\star} \star, 3) \vdash$	
$(q_5, \star \star \star \star, 4) \vdash$	
$(q_6, \star \star \star \underline{\star} \star, 4) \vdash$	
$(q_7, \star \star \star \star \underline{\star}, 5) \vdash$	
$(q_8, \star \star \star \star , 5) \vdash$	

7.2 Computability, decidability, enumerability

Definition 7.2.1. Turing-computable function

$f : \mathbb{N}^n \rightarrow \mathbb{N}$ is Turing-computable iff, exists a TM, such that $\forall (x_1, \dots, x_n) \in \mathbb{N}^n$:

M halts behind $f(x_1, \dots, x_n)$, if we placed it behind x_1, \dots, x_n

$F(MT)$ is the set of all Turing-computable functions.

Remark. If the function is not defined for certain values (i.e. is not a total function), then for those values the TM will not halt.

Definition 7.2.2. T-MT

$$T-MT = \{ f \in F(MT) : f \text{ is a total function} \}$$

Definition 7.2.3. Turing decidable set

$A \subseteq \mathbb{N}^n$ is Turing decidable iff exists a TM M , such that $\forall (x_1, x_2, \dots, x_n) \in \mathbb{N}^n$, M placed behind the strings x_1, x_2, \dots, x_n , it halts in a terminal configuration (q, E, z) and:

$$E(z) \neq a_0 \iff (x_1, \dots, x_n) \in A$$

DEC(MT) is the set of all Turing decidable sets.

Definition 7.2.4. Turing enumerable set

$A \subseteq \mathbb{N}^n$ is Turing enumerable iff exists a TM M , such that $\forall (x_1, x_2, \dots, x_n) \in \mathbb{N}^n$, M placed behind the strings x_1, x_2, \dots, x_n :

$$M \text{ halts in a terminal configuration } (q, E, z) \wedge E(z) \neq a_0 \iff (x_1, \dots, x_n) \in A$$

ENU(MT) is the set of all Turing enumerable sets.

Remark. DEC(MT) \subset ENU(MT)

Definition 7.2.5. Turing decidable predicate

A predicate P is Turing decidable, iff $P \in \text{PRED}(\text{T-MT})$. We define this set as:

$$\text{PRED}(\text{T-MT}) = \{P_f : f \in \text{T-MT}\}$$

Definition 7.2.6. Turing enumerable predicate

A predicate P is Turing decidable, iff $P \in \text{PRED}(\text{MT})$. We define this set as:

$$\text{PRED}(\text{MT}) = \{P_f : f \in \text{F}(\text{MT})\}$$

Remark. $\text{PRED}(\text{T-MT}) \subset \text{PRED}(\text{MT})$

Chapter 8

Recursive functions

8.1	Formal definition	108
8.2	Enumerability, decidability, computability	116

8.1 Formal definition

Definition 8.1.1. Zero function (θ) (*función cero*)

$$\theta : \mathbb{N}^0 \rightarrow \mathbb{N}$$

$$\theta() = 0$$

Definition 8.1.2. Successor function (*función sucesor*)

$$\sigma : \mathbb{N} \rightarrow \mathbb{N}$$

$$\sigma(n) = n + 1$$

Definition 8.1.3. Projection function π_i^k (*función proyección de k argumentos*)

For each pair $(k, i) \in \mathbb{N}^2$ with $k \geq 1$ and $1 \leq i \leq k$ the k -ary projection function that returns the i -th argument is defined as:

$$\pi_i^k : \mathbb{N}^k \rightarrow \mathbb{N} \quad \pi_i^k(n_1, \dots, n_k) = n_i$$

The set of all the projection functions is represented as π , that is:

$$\pi = \{\pi_i^k : (k, i) \in \mathbb{N}^2 \text{ with } k \geq 1 \wedge 1 \leq i \leq k\}$$

Definition 8.1.4. Initial functions (*funciones iniciales*)

$$\text{INI} = \pi \cup \{\theta, \sigma\}$$

Definition 8.1.5. Composition (*comopsición de funciones*)

Let $m > 0, k \geq 0$ and the functions:

$$g : \mathbb{N}^m \rightarrow \mathbb{N}$$

$$h_1, \dots, h_m : \mathbb{N}^k \rightarrow \mathbb{N}$$

If the function $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is:

$$f(\underline{n}) = g(h_1(\underline{n}), \dots, h_m(\underline{n}))$$

then f is obtained by composition of g and h_1, \dots, h_m , which is expressed as $f(\underline{n}) = g(h_1, \dots, h_m)(\underline{n})$, or simply $f = g(h_1, \dots, h_m)$.

Definition 8.1.6. Primitive recursion (*recursión primitiva*)

Let $k \geq 0$ and the functions

$$g : \mathbb{N}^k \rightarrow \mathbb{N}$$

$$h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$$

If the function $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ is:

$$f(\underline{n}, m) = \begin{cases} g(\underline{n}) & \text{if } m = 0 \\ h(\underline{n}, m-1, f(\underline{n}, m-1)) & \text{if } m > 0 \end{cases}$$

then f is obtained from g and h by primitive recursion.

We will express it as $f(\underline{n}) = \langle g \mid h \rangle (\underline{n})$, or simply $f = \langle g \mid h \rangle$.

Definition 8.1.7. Minimization (*minimización no acotada*)

Let $k \geq 0$ and the function:

$$g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$$

If the function $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is:

$$f(\underline{n}) = \begin{cases} \text{minimum}(A) & \text{if } A \neq \emptyset \wedge \forall t \leq \text{minimum}(A) \ g(\underline{n}, t) \in \mathbb{N} \\ \uparrow & \text{otherwise} \end{cases}$$

where $A = \{t \in \mathbb{N} : g(\underline{n}, t) = 0\}$ and $\underline{n} \in \mathbb{N}^k$. Then, f is obtained from g by minimization.

We will express it as $f(\underline{n}) = \mu [g] (\underline{n})$, or simply $f = \mu [g]$.

Remark. The symbol " \uparrow " means that the function is not defined (diverges) for that input.

Example 8.1.1. *add* is a recursive function

$$\begin{aligned} add &: \mathbb{N}^2 \rightarrow \mathbb{N} \\ add(x, y) &= x + y \end{aligned}$$

It can be defined like:

$$add = \langle \pi_1^1 | successor_3 \rangle$$

where

$$\begin{aligned} successor_3 &: \mathbb{N}^3 \rightarrow \mathbb{N} \\ successor_3(x, y, z) &= z + 1 \end{aligned}$$

is also a recursive function, because it can be defined as

$$successor_3 = \sigma(\pi_3^3)$$

yielding

$$add = \langle \pi_1^1 | \sigma(\pi_3^3) \rangle$$

Similarly, the *subtract* function is also a recursive function

$$subtract = \langle \pi_1^1 | \langle \theta | \pi_1^2 \rangle (\pi_3^3) \rangle$$

Example 8.1.2. Adding two numbers recursively

$$\begin{aligned} \text{add}(4, 3) &= \langle \pi_1^1 \mid \sigma(\pi_3^3) \rangle (4, 3) \\ &= \sigma(\pi_3^3)(4, 2, \langle \pi_1^1 \mid \sigma(\pi_3^3) \rangle (4, 2)) \\ &= \sigma(\pi_3^3)(4, 2, \sigma(\pi_3^3)(4, 1, \langle \pi_1^1 \mid \sigma(\pi_3^3) \rangle (4, 1))) \\ &= \sigma(\pi_3^3)(4, 2, \sigma(\pi_3^3)(4, 1, \sigma(\pi_3^3)(4, 0, \langle \pi_1^1 \mid \sigma(\pi_3^3) \rangle (4, 0)))) \\ &= \sigma(\pi_3^3)(4, 2, \sigma(\pi_3^3)(4, 1, \sigma(\pi_3^3)(4, 0, \pi_1^1(4)))) \\ &= \sigma(\pi_3^3)(4, 2, \sigma(\pi_3^3)(4, 1, \sigma(\pi_3^3)(4, 0, 4))) \\ &= \sigma(\pi_3^3)(4, 2, \sigma(\pi_3^3)(4, 1, \sigma(\pi_3^3(4, 0, 4)))) \\ &= \sigma(\pi_3^3)(4, 2, \sigma(\pi_3^3)(4, 1, \sigma(4))) \\ &= \sigma(\pi_3^3)(4, 2, \sigma(\pi_3^3)(4, 1, 5)) \\ &= \sigma(\pi_3^3)(4, 2, \sigma(\pi_3^3(4, 1, 5))) \\ &= \sigma(\pi_3^3)(4, 2, \sigma(5)) \\ &= \sigma(\pi_3^3)(4, 2, 6) \\ &= \sigma(\pi_3^3(4, 2, 6)) \\ &= \sigma(6) \\ &= 7 \end{aligned}$$

Example 8.1.3. *product* is a recursive function

$$\begin{aligned} \text{product} &: \mathbb{N}^2 \rightarrow \mathbb{N} \\ \text{product}(x, y) &= x \cdot y \end{aligned}$$

and it is a recursive function, because it can be defined like:

$$\text{product} = \langle \theta^1 \mid \text{add}_{1,3}^3 \rangle$$

being θ^1 and $\text{add}_{1,3}^3$ recursive functions as well:

$$\theta^1 : \mathbb{N}^1 \rightarrow \mathbb{N}$$

$$\theta^1(x) = 0$$

$$\theta^1(x) = \langle \theta \mid \pi_2^2 \rangle$$

$$\text{add}_{1,3}^3 : \mathbb{N}^3 \rightarrow \mathbb{N}$$

$$\text{add}_{1,3}^3(x, y, z) : x + z$$

$$\text{add}_{1,3}^3 = \text{add}(\pi_1^3, \pi_3^3) = \langle \pi_1^1 \mid \sigma(\pi_3^3) \rangle (\pi_1^3, \pi_3^3)$$

yielding

$$\text{product} = \langle \langle \theta \mid \pi_2^2 \rangle \mid \langle \pi_1^1 \mid \sigma(\pi_3^3) \rangle (\pi_1^3, \pi_3^3) \rangle$$

Example 8.1.4. *divide* is a recursive function

$$\begin{aligned} \text{divide} &: \mathbb{N}^2 \rightarrow \mathbb{N} \\ \text{divide}(x, y) &= \frac{x}{y} \end{aligned}$$

it can be defined with the minimization operator:

$$\text{divide} = \mu [\text{subtract}(\pi_1^3, \text{product}(\pi_2^3, \pi_3^3))]$$

namely,

$$\text{divide} = \mu [\langle \pi_1^1 \mid \langle \sigma \mid \pi_2^2 \rangle (\pi_3^3) \rangle (\pi_1^3, \langle \langle \sigma \mid \pi_2^2 \rangle \mid \langle \pi_1^1 \mid \sigma(\pi_3^3) \rangle (\pi_1^3, \pi_3^3) \rangle (\pi_2^3, \pi_3^3))]$$

In this approach, the quotient is found in a search process, as the minimum number which multiplied by the divisor equals (or exceeds) the dividend.

For example,

$$\text{divide}(6, 3) = 2, \text{ since } 6 - 3 \cdot 0 = 6, 6 - 3 \cdot 1 = 3 \text{ and } 6 - 3 \cdot 2 = 0$$

$$\text{divide}(25, 3) = 9, \text{ since } 25 - 3 \cdot 8 = 1 \text{ and } 25 - 3 \cdot 9 = 0$$

Definition 8.1.8. Total function, Partial functions

$f : \mathbb{N}^k \rightarrow \mathbb{N}, k > 0$ is a total function if $Dom(f) = \mathbb{N}^k$
 $f : \mathbb{N}^0 \rightarrow \mathbb{N}$ is a total function if $f() \in \mathbb{N}$

A function is a partial function iff it is not total.

Definition 8.1.9. Set of the recursive functions

The set REC is defined as

1. $INI \subset REC$
2. $g, h_1, \dots, h_m \in REC \wedge \exists g(h_1, \dots, h_m) \Rightarrow g(h_1, \dots, h_m) \in REC$
3. $g, h \in REC \wedge \exists \langle g|h \rangle \Rightarrow \langle g|h \rangle \in REC$
4. $g \in REC \wedge \exists \mu[g] \Rightarrow \mu[g] \in REC$
5. No other function is in REC.

f is a recursive function iff $f \in REC$.

Definition 8.1.10. Total recursive functions

$$TREC = \{f \in REC : f \text{ is total}\}$$

8.2 Enumerability, decidability, computability

Definition 8.2.1. Set of truth values for a predicate (*valores de verdad*)

The set of truth values for a predicate P , of $k \geq 0$ arguments is

$$V_P = \{\underline{x} \in \mathbb{N}^k : P(\underline{x})\}$$

Definition 8.2.2. Predicate associated to a function

The predicate associated to the function f of $k \geq 0$ arguments, is

$$P_f(\underline{x}) = \begin{cases} true & \text{if } f(\underline{x}) > 0 \\ false & \text{otherwise} \end{cases}$$

Definition 8.2.3. Characteristic function of a set (*función característica*)

The characteristic function of $V \subseteq \mathbb{N}^k$, $X_V : \mathbb{N}^k \rightarrow \mathbb{N}$, with $k \geq 0$, is

$$X_V(\underline{x}) = \begin{cases} 1 & \text{if } \underline{x} \in V \\ 0 & \text{if } \underline{x} \notin V \end{cases}$$

Definition 8.2.4. Characteristic function of a predicate

X_{V_P} is the characteristic function of predicate P .

Definition 8.2.5. Recursively decidable predicates (*predicado r.d.*)

$$\text{PRED}(\text{TREC}) = \{P_f : f \in \text{TREC}\}$$

Definition 8.2.6. Recursively enumerable predicates (*predicado r.e.*)

$$\text{PRED}(\text{REC}) = \{P_f : f \in \text{REC}\}$$

Remark. $\text{PRED}(\text{TREC}) \subset \text{PRED}(\text{REC})$ since $\text{TREC} \subset \text{REC}$.

Definition 8.2.7. Recursively decidable sets

$$\text{DEC} = \{V_P : P \in \text{PRED}(\text{TREC})\}$$

Definition 8.2.8. Recursively enumerable sets

$$\text{ENU} = \{V_P : P \in \text{PRED}(\text{REC})\}$$

Remark. $\text{DEC} \subset \text{ENU}$ since $\text{PRED}(\text{TREC}) \subset \text{PRED}(\text{REC})$.

Definition 8.2.9. Recursively generable set

V is a recursively generable set iff V is recursively enumerable.

Chapter 9

The WHILE language

9.1	Formal definition	120
9.2	Computability, decidability, enumerability	127

9.1 Formal definition

Given the alphabets

$$\Sigma_d = \{0, 1, 2, 3, 4; 5, 6, 7, 8, 9\}$$

$$\Sigma_w = \{X, :=, 1, +, -, \text{while}, \neq, 0, \text{do}, \text{od}, ;\}$$

Let $G = (N, T, P, < \text{code} >)$ be a grammar with:

$$N = \{ < \text{number} >, < \text{firstdigit} >, < \text{restnumber} >, < \text{digit} >, \\ < \text{code} >, < \text{sentence} >, < \text{assignment} >, < \text{loop} >, < \text{id} > \}$$

$$T = \Sigma_d \cup \Sigma_w$$

$$P = \{ \\ < \text{number} > \rightarrow < \text{firstdigit} > \mid < \text{firstdigit} > < \text{restnumber} >, \\ < \text{firstdigit} > \rightarrow 1|2|3|4|5|6|7|8|9, \\ < \text{digit} > \rightarrow 0|1|2|3|4|5|6|7|8|9, \\ < \text{restnumber} > \rightarrow < \text{digit} > \mid < \text{digit} > < \text{restnumber} >, \\ < \text{code} > \rightarrow < \text{sentence} > \mid < \text{code} >; < \text{sentence} >, \\ < \text{sentence} > \rightarrow < \text{assignment} > \mid < \text{loop} >, \\ < \text{id} > \rightarrow X < \text{number} >, \\ < \text{loop} > \rightarrow \text{while } < \text{id} > \neq 0 \text{ do } < \text{code} > \text{od}, \\ < \text{assignment} > \rightarrow < \text{id} > := 0 \mid < \text{id} > := < \text{id} > \mid \\ & \quad < \text{id} > := < \text{id} > + 1 \mid < \text{id} > := < \text{id} > - 1 \}$$

Definition 9.1.1. Set of While codes (CODE) (*códigos While*)

$s \in T^+$ is a While code iff $s \in \text{CODE} = \mathcal{L}(G)$.

Remark. CODE contains all the syntactically correct codes written in While language.

Definition 9.1.2. Set of While programs (WHILE) (*programas While*)

Q is a While program iff $Q \in \text{WHILE} = \{(n, s) \in \mathbb{N} \times \text{CODE}\}$.

Definition 9.1.3. Size of a While code and a While program

$size : \text{CODE} \rightarrow \mathbb{N}$

$$size(s) = \begin{cases} 1 & \text{if } \langle assignment \rangle \Rightarrow^* s \\ size(s) = size(s_1) + 2 & \text{if } s = \text{while } Xi \neq 0 \text{ do } s_1 \text{ od } \wedge \\ & i \in \Sigma_d^+ \wedge s_1 \in \text{CODE} \\ size(s) = size(s_1) + size(s_2) & \text{if } s = s_1; s_2 \wedge s_1, s_2 \in \text{CODE} \end{cases}$$

Similarly, for a While program (overloading the function):

$size : \text{WHILE} \rightarrow \mathbb{N}$

$size((n, s)) = size(s)$

Definition 9.1.4. Line function

$line : CODE \times \mathbb{N} \rightarrow T^*$

$$line(s, n) = \begin{cases} s & \text{if } \langle assignment \rangle \Rightarrow^* s \wedge n = 1 \\ \\ \text{while } Xi \neq 0 \text{ do} & \text{if } s = \text{while } Xi \neq 0 \text{ do } s_1 \wedge \\ & i \in \Sigma_d^+ \wedge s_1 \in CODE \wedge n = 1 \\ line(s_1, n - 1) & \text{if } s = \text{while } Xi \neq 0 \text{ do } s_1 \wedge \\ & i \in \Sigma_d^+ \wedge s_1 \in CODE \wedge 1 < n < size(s) \\ \text{od} & \text{if } s = \text{while } Xi \neq 0 \text{ do } s_1 \wedge \\ & i \in \Sigma_d^+ \wedge s_1 \in CODE \wedge n = size(s) \\ \\ line(s_1, n) & \text{if } s = s_1; s_2 \wedge \\ & s_1, s_2 \in CODE \wedge 0 < n \leq size(s_1) \\ line(s_2, n - size(s_1)) & \text{if } s = s_1; s_2 \wedge \\ & s_1, s_2 \in CODE \wedge size(s_1) < n \leq size(s) \\ \\ \varepsilon & \text{otherwise} \end{cases}$$

Definition 9.1.5. Control flow function

$go : \text{CODE} \times \mathbb{N} \rightarrow \mathbb{N}$

$$go(s, n) = \begin{cases} size(s) + 1 & \text{if } s = \text{while } Xi \neq 0 \text{ do } s_1 \text{ od } \wedge \\ & i \in \Sigma_d^+ \wedge s_1 \in \text{CODE} \wedge n = 1 \\ go(s_1, n - 1) + 1 & \text{if } s = \text{while } Xi \neq 0 \text{ do } s_1 \text{ od } \wedge \\ & i \in \Sigma_d^+ \wedge s_1 \in \text{CODE} \wedge 1 < n < size(s) \\ 1 & \text{if } s = \text{while } Xi \neq 0 \text{ do } s_1 \text{ od } \wedge \\ & i \in \Sigma_d^+ \wedge s_1 \in \text{CODE} \wedge n = size(s) \\ go(s_1, n) & \text{if } s = s_1; s_2 \wedge \\ & s_1, s_2 \in \text{CODE} \wedge 0 < n \leq size(s_1) \\ go(s_2, n - size(s_1)) + size(s_1) & \text{if } s = s_1; s_2 \wedge \\ & s_1, s_2 \in \text{CODE} \wedge size(s_1) < n \leq size(s) \\ 0 & \text{otherwise} \end{cases}$$

Remark. The go function returns the non-consecutive line number that could be jumped to from a given line or zero if from the specified line you can not jump to a non-consecutive line.

Definition 9.1.6. Configurations of a While program

The set of configurations of $Q = (n, s) \in \text{WHILE}$ is

$$C_Q = \{(m, \underline{x}) \in \mathbb{N}^{p+1} : 1 \leq m \leq \text{size}(Q) + 1\}$$

where $p = \max \{m \in \Sigma_d^+ : X_m \prec s\}$ (also in further definitions).

Definition 9.1.7. Initial configuration

Let $Q = (n, s) \in \text{WHILE}$.

$\underline{c} = (1, \underline{x}) \in C_Q$ is a initial configuration of Q iff $x_{n+1} = \dots = x_p = 0$.

Definition 9.1.8. Terminal configuration, non-terminal configuration

Let $Q = (n, s) \in \text{WHILE}$.

$\underline{c} = (m, \underline{x}) \in C_Q$ is a terminal configuration of Q iff $m = \text{size}(Q) + 1$.

Otherwise we say that it is a non-terminal configuration.

Remark. A configuration can not be both initial and terminal.

Definition 9.1.9. Direct transition

Let $Q = (n, s) \in \text{WHILE}$ and $\underline{c}_1 = (m, \underline{x}), \underline{c}_2 = (t, \underline{z}) \in C_Q$.

Q transits directly from c_1 to c_2 , $c_1 \vdash c_2$, iff

$$\begin{array}{lll}
\text{line}(s, m) = \text{Xi} := 0 & \Rightarrow & t = m + 1 \wedge z_i = 0 \quad \wedge z_r = x_r, \forall r \neq i \\
\text{line}(s, m) = \text{Xi} := \text{Xj} & \Rightarrow & t = m + 1 \wedge z_i = x_j \quad \wedge z_r = x_r, \forall r \neq i \\
\text{line}(s, m) = \text{Xi} := \text{Xj} + 1 & \Rightarrow & t = m + 1 \wedge z_i = x_j + 1 \quad \wedge z_r = x_r, \forall r \neq i \\
\text{line}(s, m) = \text{Xi} := \text{Xj} - 1 & \Rightarrow & t = m + 1 \wedge z_i = x_j - 1 \quad \wedge z_r = x_r, \forall r \neq i \\
\text{line}(s, m) = \text{while } \text{Xi} \neq 0 \text{ do} & \Rightarrow & (x_i \neq 0 \Rightarrow t = m + 1) \quad \wedge \\
& & (x_i = 0 \Rightarrow t = \text{go}(s, m)) \quad \wedge \underline{z} = \underline{x} \\
\text{line}(s, m) = \text{od} & \Rightarrow & t = \text{go}(s, m) \quad \wedge \underline{z} = \underline{x}
\end{array}$$

where $i, j \in \Sigma_d^+$.

Remark. Assignments update the variables, loop headers do not alter the variables, and loop tails redirect to its header.

Definition 9.1.10. Next configuration function (*función siguiente configuración*)

Let $Q = (n, s) \in \text{WHILE}$ (also in further definitions).

$$\text{next}_Q : \mathbb{N}^{p+1} \rightarrow \mathbb{N}^{p+1}$$

$$\text{next}_Q(\underline{c}) = \begin{cases} \underline{c}' & \text{if } \underline{c}, \underline{c}' \in C_Q \wedge \underline{c} \vdash \underline{c}' \\ \underline{c} & \text{if } \underline{c} \notin C_Q \vee \underline{c} \in C_Q \wedge \nexists \underline{c}' \in C_Q : \underline{c} \vdash \underline{c}' \end{cases}$$

Definition 9.1.11. Calculation function (*función cálculo*)

$$\text{cal}_Q : \mathbb{N}^{n+1} \rightarrow \mathbb{N}^{p+1}$$

$$\text{cal}_Q(\underline{x}, t) = \begin{cases} (1, \underline{x}, \underline{0}) & \text{if } t = 0 \\ \text{next}_Q(\text{cal}_Q(\underline{x}, t - 1)) & \text{if } t > 0 \end{cases}$$

where $t \in \mathbb{N}, \underline{x} \in \mathbb{N}^n \wedge \underline{0} \in \mathbb{N}^{p-n}$.

Definition 9.1.12. Temporal complexity function (*función complejidad temporal*)

$$T_Q : \mathbb{N}^n \rightarrow \mathbb{N}$$

$$T_Q(\underline{x}) = \begin{cases} \text{minimum}(A) & \text{if } A \neq \emptyset \\ \uparrow & \text{if } A = \emptyset \end{cases}$$

where $A = \{t \in \mathbb{N} : \pi_1^{p+1}(\text{cal}_Q(\underline{x}, t)) = \text{size}(Q) + 1\}$

Definition 9.1.13. Calculated function (*función calculada*)

$$F_Q : \mathbb{N}^n \rightarrow \mathbb{N}$$

$$F_Q(\underline{x}) = \pi_2^{p+1}(\text{cal}_Q(\underline{x}, T_Q(\underline{x})))$$

9.2 Computability, decidability, enumerability

Definition 9.2.1. While-computable function (*función While-calculable*)

f is a While-computable function iff $\exists Q \in \text{WHILE} : F_Q = f$.

$F(\text{WHILE})$ is the set of all While-computable functions.

Definition 9.2.2. Total While-computable function

$\text{T-WHILE} = \{f \in F(\text{WHILE}) : f \text{ is a total function}\}.$

Definition 9.2.3. While-decidable predicates (*predicados While-decidibles*)

$\text{PRED}(\text{T-WHILE}) = \{P_f : f \in \text{T-WHILE}\}.$

Definition 9.2.4. While-enumerable predicates (*predicados While-enumerables*)

$\text{PRED}(\text{WHILE}) = \{P_f : f \in F(\text{WHILE})\}.$

Definition 9.2.5. While-decidable sets (*conjuntos While-decidibles*)

$\text{DEC}(\text{WHILE}) = \{V_P : P \in \text{PRED}(\text{T-WHILE})\}.$

Definition 9.2.6. While-enumerable sets (*conjuntos While-enumerables*)

$\text{ENU}(\text{WHILE}) = \{V_P : P \in \text{PRED}(\text{WHILE})\}.$

Equivalences among functions, predicates and sets:

F(MT)	REC	F(WHILE)
DEC(MT)	DEC	DEC(WHILE)
ENU(MT)	ENU	ENU(WHILE)
T-MT	TREC	T-WHILE
PRED(T-MT)	PRED(TREC)	PRED(T-WHILE)
PRED(MT)	PRED(REC)	PRED(WHILE)

Chapter 10

Turing completeness

10.1 Extended While language

EXWHILE is the set of all programmes in extended While language, $F(\text{EXWHILE})$ is the set of all the functions represented by EXWHILE.

It can be shown that $\text{WHILE} \subseteq \text{EXWHILE}$ and $F(\text{WHILE}) = F(\text{EXWHILE})$.

Example 10.1.1. *factorial* function in Extended While

Factorial = (1, *s*)

s :

```
(* counting starts from the previous value *)
X2 := X1 - 1;
while X2  $\neq$  0 do
    (* macros with prefix or infix notation *)
    X1 := Product(X1, X2);
    X2 := X2 - 1
od
```

Some other control flow structures are allowed:

while C **do**

s
od

do $f(x)$ **times**

s
od

if C **then**

s
fi

if C **then**

s_1
else

s_2
fi

C is defined from a WHILE-enumerable predicate, and $f(x)$ is a WHILE-computable function.

10.2 Recursive \Rightarrow While-computable

Theorem 10.2.1. $\text{REC} = \text{F}(\text{WHILE})$

Proposition 10.2.1. $\text{REC} \subseteq \text{F}(\text{WHILE})$

Proposition 10.2.2. $\text{F}(\text{WHILE}) \subseteq \text{REC}$

Proposition 10.2.3. $INI \subseteq F(\text{WHILE})$

Proof.

$\theta \in F(\text{WHILE})$

Given $Q = (0, x1 := 0)$,

$$F_Q = \theta \Rightarrow \theta \in F(\text{WHILE})$$

$\sigma \in F(\text{WHILE})$

Given $Q = (1, x1 := x1 + 1)$,

$$F_Q = \sigma \Rightarrow \sigma \in F(\text{WHILE})$$

$\pi \subseteq F(\text{WHILE})$

Given $Q = (k, x1 := xi)$, $k \geq 1, 1 \leq i \leq k$,

$$F_Q = \pi_i^k \Rightarrow \pi_i^k \in F(\text{WHILE}) \Rightarrow \pi \subseteq F(\text{WHILE})$$

$$INI = \pi \cup \{\theta, \sigma\} \Rightarrow INI \subseteq F(\text{WHILE})$$

□

Proposition 10.2.4.

$$f = g(h_1, \dots, h_m) \wedge g, h_1, \dots, h_m \in F(\text{WHILE}) \Rightarrow f \in F(\text{WHILE})$$

Proof.

Given $f = g(h_1, \dots, h_m)$,

$$g : \mathbb{N}^m \rightarrow \mathbb{N}, \quad F_G = g,$$

$$h_i : \mathbb{N}^k \rightarrow \mathbb{N}, \quad F_{H_i} = h_i,$$

$$m \geq 1, \quad k \geq 0$$

and

$$C = (k, s)$$

$s :$

$$X_{k+1} := H_1(X_1, \dots, X_k);$$

...

$$X_{k+m} := H_m(X_1, \dots, X_k);$$

$$X_1 := G(X_{k+1}, \dots, X_{k+m})$$

then $F_C = f \Rightarrow f \in F(\text{WHILE})$

□

Proposition 10.2.5.

$$f = \langle g|h \rangle \wedge g, h \in F(\text{WHILE}) \Rightarrow f \in F(\text{WHILE})$$

Proof.

Given $f = \langle g|h \rangle$,

$$g : \mathbb{N}^k \rightarrow \mathbb{N}, \quad F_G = g,$$

$$h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}, \quad F_H = h,$$

$$k \geq 0$$

and

$$R = (k+1, s)$$

$s :$

$X_{k+2} := G(X_1, \dots, X_k);$

do X_{k+1} **times**

$X_{k+2} := H(X_1, \dots, X_k, X_{k+3}, X_{k+2});$

$X_{k+3} := X_{k+3} + 1$

od

$X_1 := X_{k+2}$

then $F_R = f \Rightarrow f \in F(\text{WHILE})$

□

Proposition 10.2.6.

$$f = \mu[g] \wedge g \in F(\text{WHILE}) \Rightarrow f \in F(\text{WHILE})$$

Proof.

Given $f = \mu[g]$,

$$g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}, \quad F_G = g,$$

$$k \geq 0$$

and

$$M = (k, s)$$

$s :$

while $G(X_1, \dots, X_k, X_{k+1}) \neq 0$ **do**

$$X_{k+1} := X_{k+1} + 1$$

od

$$X_1 := X_{k+1}$$

then $F_M = f \Rightarrow f \in F(\text{WHILE})$

□

Chapter 11

Universality

11.1 Program numbering	138
11.2 Universal function	149
11.3 Universal program	151

11.1 Program numbering

Definition 11.1.1. Index of a set of functions

An index of a countable set of functions \mathbf{F} is any surjective function $h : \mathbb{N} \rightarrow \mathbf{F}$

An index of a function $f \in \mathbf{F}$ under the index h is any $i \in \mathbb{N}$ such that $h(i) = f$.

In order to codify programs, we need to introduce the concepts of Gödel's encoding and decoding. These functions are defined using the set of all vectors of natural numbers.

$$\mathbb{N}^* = \bigcup_{n=0}^{\infty} \mathbb{N}^n$$

Remark. $\mathbb{N}^0 = \{()\}$.

Definition 11.1.2. Cantor pairing function (*cantORIZACIÓN de \mathbb{N}^2*)
Cantor pairing establishes a bijection between \mathbb{N} and \mathbb{N}^2 .

$$\sigma_1^2 : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$\sigma_1^2(x, y) = \frac{(x + y)(x + y + 1)}{2} + y$$

$$\sigma_{2,2}^1 : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$$

$$\sigma_{2,2}^1(z) = z - \sigma_1^2(d(z), 0)$$

$$\sigma_{2,1}^1 : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$$

$$\sigma_{2,1}^1(z) = d(z) - \sigma_{2,2}^1(z)$$

where

$$d(z) = \min\{t \in \mathbb{N} : \sigma_1^2(t, 0) > z\} - 1$$

(See [figure](#) in Cantor pairing function Wikipedia article.)

Definition 11.1.3. Cantor tuple function (*cantorización de \mathbb{N}^{k+1}*)

Each Cantor tuple function establishes a bijective function between \mathbb{N} and \mathbb{N}^{k+1} .

$$\sigma_1^{k+1} : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$$
$$\sigma_1^{k+1}(x_1, \dots, x_{k+1}) = \sigma_1^2(\sigma_1^k(x_1, \dots, x_k), x_{k+1})$$

$$\sigma_{k+1,p}^1 : \mathbb{N} \rightarrow \mathbb{N}$$
$$\sigma_{k+1,p}^1(z) = \begin{cases} \sigma_{k,p}^1(\sigma_{2,1}^1(z)) & \text{if } 1 \leq p \leq k \\ \sigma_{2,2}^1(z) & \text{if } p = k + 1 \end{cases}$$

Remark. Cantor tuple is a family of countable infinite bijective functions.

Definition 11.1.4. Gödel encoding of \mathbb{N}^* (*godelización de \mathbb{N}^**)

Gödel encoding establishes a bijection between \mathbb{N} and $\mathbb{N}^* = \bigcup_{k \in \mathbb{N}} \mathbb{N}^k$.

$$\Gamma : \mathbb{N}^* \rightarrow \mathbb{N}$$

$$\Gamma(\underline{x}) = \begin{cases} 0 & \text{if } |\underline{x}| = 0 \\ \sigma_1^2(|\underline{x}| - 1, \sigma_1^{|\underline{x}|}(\underline{x})) + 1 & \text{if } |\underline{x}| > 0 \end{cases}$$

$$\gamma : \mathbb{N} \rightarrow \mathbb{N}^*$$

$$\gamma(z, k) = \begin{cases} 0 & \text{if } z = 0 \vee k > \text{length}(z) \\ \sigma_{\text{length}(z), k}^1(\sigma_{2,2}^1(z - 1)) & \text{otherwise} \end{cases}$$

where $\text{length}(z)$ refers to the length of the vector obtained from decoding z , namely

$$\text{length}(z) = m \text{ iff } \Gamma(\underline{x}) = z \wedge \underline{x} \in \mathbb{N}^m$$

or, in terms of the previous encoding: $\text{length}(z) = \sigma_{2,1}^1(z - 1) + 1$ if $z > 0$, and $\text{length}(0) = 0$.

Remark. Gödel encoding enumerates \mathbb{N}^* :

0	1	2	3	4	5	6	7	8	9	10	...
()	(0)	(0, 0)	(1)	(0, 0, 0)	(1, 0)	(2)	(0, 0, 0, 0)	(1, 0, 0)	(0, 1)	(3)	...

11.1.1 Encoding of While programs

Definition 11.1.5. Encoding of While sentences

$sent2N : \text{CODE}^1 \rightarrow \mathbb{N}$ with $\text{CODE}^1 = \mathcal{L}((N, T, P, < \text{sentence} >))$

$$sent2N(c) = \begin{cases} 5(i-1) & \text{if } c = X_i := 0 \\ 5\sigma_1^2(i-1, j-1) + 1 & \text{if } c = X_i := X_j \\ 5\sigma_1^2(i-1, j-1) + 2 & \text{if } c = X_i := X_j + 1 \\ 5\sigma_1^2(i-1, j-1) + 3 & \text{if } c = X_i := X_j - 1 \\ 5\sigma_1^2(i-1, code2N(b)) + 4 & \text{if } c = \text{while } X_i \neq 0 \text{ do } b \text{ od} \end{cases}$$

Definition 11.1.6. Encoding of While codes

Given $c = s_1; \dots; s_m \in \text{CODE}$, where each s_i is a sentence,

$$\text{code}2N(c) = \Gamma(\text{sent}2N(s_1), \dots, \text{sent}2N(s_m)) - 1$$

Definition 11.1.7. Encoding of While programs

Given $Q = (n, c) \in \text{WHILE}$,

$$\text{while}2N : \text{WHILE} \rightarrow \mathbb{N}$$

$$\text{while}2N(Q) = \sigma_1^2(n, \text{code}2N(c))$$

Example 11.1.1. Encoding of the double program

$double = (1, s)$

$s :$

$x2 := x1;$

while $x2 \neq 0$ **do**

$x1 := x1 + 1;$

$x2 := x2 - 1$

od

$code2N(s) = 1e + 39$

$while2N(double) = 5e + 77$

11.1.2 Decoding While programs

Definition 11.1.8. Type of sentence function

$$\textit{senttype} : \mathbb{N} \rightarrow \{0, 1, 2, 3, 4\}$$

$$\textit{senttype}(z) = z \bmod 5$$

Definition 11.1.9. *LHS* and *RHS* functions

$$\textit{LHS}, \textit{RHS} : \mathbb{N} \rightarrow \mathbb{N}$$

$$\textit{LHS}(z) = \begin{cases} 1 + \frac{1}{5}z & \text{if } \textit{senttype}(z) = 0 \\ 1 + \sigma_{2,1}^1 \left(\frac{1}{5}(z - \textit{senttype}(z)) \right) & \text{if } \textit{senttype}(z) \in \{1, 2, 3, 4\} \end{cases}$$

$$\textit{RHS}(z) = 1 + \sigma_{2,2}^1 \left(\frac{1}{5}(z - \textit{senttype}(z)) \right)$$

Definition 11.1.10. Decoding a While sentence

$N2sent : \mathbb{N} \rightarrow \text{CODE}$

$$N2sent(z) = \begin{cases} X_i := 0 & \text{if } senttype(z) = 0 \\ X_i := X_j & \text{if } senttype(z) = 1 \\ X_i := X_j + 1 & \text{if } senttype(z) = 2 \\ X_i := X_j - 1 & \text{if } senttype(z) = 3 \\ \text{while } X_i \neq 0 \text{ do } N2code(j-1) \text{ od} & \text{if } senttype(z) = 4 \end{cases}$$

where $i = LHS(z)$, $j = RHS(z)$

Definition 11.1.11. Decoding While codes

$N2code : \mathbb{N} \rightarrow \text{CODE}$

$$N2code(z) = \begin{cases} N2sent(\gamma(z+1, 1)) & \text{if } z = 0 \\ N2sent(\gamma(z+1, 1)); \dots; N2sent(\gamma(z+1, length(z))) & \text{if } z > 0 \end{cases}$$

Definition 11.1.12. Decoding WHILE programs

$N2while : \mathbb{N} \rightarrow \text{WHILE}$

$$N2while(z) = (\sigma_{2,1}^1(z), N2code(\sigma_{2,2}^1(z)))$$

Corollary 11.1.1. Indexing of REC

$$h : \mathbb{N} \rightarrow F(\text{WHILE})$$

$$h(z) = F_{N2while(z)}$$

Given the equivalence theorem, the function is also an indexing of REC.

Corollary 11.1.2. Indexing of REC^n

$$h : \mathbb{N} \rightarrow F(\text{WHILE})^n$$

$$h(z) = F_{(n, N2code(z))}$$

Given the equivalence theorem, the function is also an indexing of REC^n .

11.2 Universal function

Definition 11.2.1. Universal function

Given a countable set of functions \mathbf{F} verifying $f \in \mathbf{F} \Rightarrow f : \mathbb{N}^n \rightarrow \mathbb{N}, n \geq 0$,

$U[\mathbf{F}] : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ is universal for \mathbf{F} iff

$\exists h : \mathbb{N} \rightarrow \mathbf{F}$ that is an indexing of \mathbf{F} : $U[\mathbf{F}](i, \underline{x}) = h(i)(\underline{x}) \ \forall i \in \mathbb{N} \ \wedge \ \forall \underline{x} \in \mathbb{N}^n$.

Remark. $U[\mathbf{F}]$ is the universal function for \mathbf{F} under the indexing h .

Theorem 11.2.1. $U[\text{REC}^n] \in \text{REC}^{n+1}$

Assuming

$$\begin{aligned} \exists U \in \text{WHILE} \quad &: F_U = U[\text{REC}^n] \Rightarrow \\ U[\text{REC}^n] \in \text{F(WHILE)}^n + 1 &\Rightarrow \\ U[\text{REC}^n] \in \text{REC}^{n+1} & \end{aligned}$$

it has to be demonstrated that $\exists U \in \text{WHILE} \quad : F_U = U[\text{REC}^n]$

Remark. U will be denoted as the universal program.

11.3 Universal program

Definition 11.3.1. Universal program

$U = (k + 1, s)$

$s :$

```
(* X1 stores the code of the program *)
(* X2, ..., Xk+1 store the arguments of the program *)
(* arguments are encoded in a single Gödel number *)
Xk+2 := godelk(X2, ..., Xk+1);
(* variables after simulating the program *)
Xk+2 := simulate(X1, Xk+2);
(* first variable is the output *)
X1 := degodel(Xk+2, 1)
```

simulate = (2, *s*)

s :

(* X1 = sentence, X2 = variables *)

X1 := X1 + 1;

while X1 ≠ 0 **do**

(* extract first sentence and check type *)

X3 := degodel(X1, 1);

(* check if assignment or loop *)

if senttype(X3) ≤ 3 **then**

(* change variable and remove *)

X2 := execute(X3, X2);

X1 := pop(X1)

else

(* value of control variable *)

X4 := lhs(X3);

if degodel(X2, X4) ≠ 0 **then**

(* push body of the loop to top of the stack *)

X4 := rhs(X3);

X1 := merge(X4)

else

(* remove sentence from the stack *)

X1 := pop(X1)

fi

fi

od

```

execute = (2, s)
s :
    (* X1 = sentence, X2 = variables *)
    X3 := senttype(X1);
    (* index of target variable *)
    X4 := lhs(X1);
    if X3 = 0 then
        X1 := replace(X2, X4, 0)
    else
        (* value of source variable *)
        X5 := rhs(X1);
        X5 := degodel(X2, X5);
        if X3 = 1 then
            X1 := replace(X2, X4, X5)
        fi
        if X3 = 2 then
            X5 := X5 + 1;
            X1 := replace(X2, X4, X5)
        fi
        if X3 = 3 then
            X5 := X5 - 1;
            X1 := replace(X2, X4, X5)
        fi
    fi

```

Chapter 12

Theoretical limits of computing

12.1 Halting problem	156
12.2 The busy beaver function	159

12.1 Halting problem

Definition 12.1.1. Problem associated to a predicate

Problem	Predicate	Function
solvable (<i>resoluble</i>)	decidable	$\in TREC$
partially solvable (<i>parcialmente resoluble</i>)	enumerable	$\in REC$
unsolvable (<i>irresoluble</i>)	undecidable	$\notin TREC$
totally unsolvable (<i>totalmente irresoluble</i>)	not enumerable	$\notin REC$

Definition 12.1.2. H predicate

Given $Q \in \text{WHILE}$, $g = \text{while2N}(Q)$, and $\underline{x} \in \mathbb{N}^*$,

$$H(g, \underline{x}) \Leftrightarrow F_Q(\underline{x}) \in \mathbb{N}$$

Remark. $H(g, \underline{x})$ is true iff Q halts with input \underline{x} .

Definition 12.1.3. H^n predicate

Given $Q \in \text{WHILE} : F_Q \in \text{F(WHILE)}^n$, $g = \text{while2N}(Q)$, and $\underline{x} \in \mathbb{N}^n$.

$$H^n(g, \underline{x}) \Leftrightarrow F_Q(\underline{x}) \in \mathbb{N}.$$

Remark. From now on, g will be referred as 'a program' (i.e. the program it encodes), or 'the number' of a program.

Theorem 12.1.1. Problem H^1 is not solvable

Proof. By contradiction:

$$H^1 \text{ solvable} \Rightarrow H^1 \text{ decidable} \Rightarrow \exists h^1 \in \text{TREC} : h^1 = X_{V_{H^1}}$$

$$h^1(g, x) = \begin{cases} 0 & \text{if } \overline{H^1(g, x)} \\ 1 & \text{if } H^1(g, x) \end{cases}$$

Let's consider the function

$$\text{bomb}(x) = \begin{cases} 1 & \text{if } h^1(x, x) = 0 \\ \uparrow & \text{if } h^1(x, x) = 1 \end{cases}$$

$Bomb = (1, s)$

$s :$

```

while  $H^1(X1, X1) \neq 0$  do
   $X1 := X1$ 
od
 $X1 := X2 + 1;$ 

```

If $b = \text{while}2N(Bomb)$ then

$$h^1(b, b) = 1 \Rightarrow \text{bomb}(b) = \uparrow \Rightarrow h^1(b, b) = 0 \quad \Rightarrow \times$$

$$h^1(b, b) = 0 \Rightarrow \text{bomb}(b) = 1 \Rightarrow h^1(b, b) = 1 \quad \Rightarrow \times$$

□

Corollary 12.1.1. H is not solvable

Remark. H^1 not solvable $\Rightarrow H^n$ not solvable $\Rightarrow H$ not solvable.

Proposition 12.1.1. H^1 is partially solvable

Proof. Let's consider

$$f : \mathbb{N}^2 \rightarrow \mathbb{N}$$

$$f = \sigma(U[\text{REC}^1])$$

$$P_f = H^1 \xRightarrow[\substack{\uparrow \\ f \in \text{REC}}]{} H^1 \in \text{PRED}(\text{REC}) \Rightarrow H^1 \text{ is partially solvable.}$$

□

Remark. Being H^1 enumerable means that all the pairs $(g, x) \in \mathbb{N} : F_{N2while(g)}(x) \in \mathbb{N}$ can be generated. This can be done by simulating each program with a given input for a number of steps, and checking if it has halted.

Corollary 12.1.2. H is partially solvable

Remark. H^1 partially solvable $\Rightarrow H^n$ partially solvable $\Rightarrow H$ partially solvable.

12.2 The busy beaver function

Definition 12.2.1. Length of a WHILE program (*longitud de un programa WHILE*)

$length : \text{WHILE} \rightarrow \mathbb{N}$

$length((n, s)) = |s|_{do} + |s|_{:=}$

Definition 12.2.2. Set of the While programs of length n (WHILE^n)

$\text{WHILE}^n = \{Q \in \text{WHILE} : length(Q) = n\}$

Definition 12.2.3. Busy beaver function (Σ)

$\Sigma : \mathbb{N} \rightarrow \mathbb{N}$

$\Sigma(n) = \max\{F_Q(0) : F_Q \in F(\text{WHILE})^1 \wedge Q \in \text{WHILE}^n \wedge Q(0) \in \mathbb{N}\}$

Proposition 12.2.1. $\Sigma(n + 1) > \Sigma(n)$

Proof. Let's consider $Q = (1, s) \in \text{WHILE}^n \wedge F_Q(0) = \Sigma(n)$, then

$Q' = (1, s')$

$s' :$

$s ;$

$\text{X1} := \text{X1} + 1$

$\text{length}(Q') = n + 1 \wedge F_{Q'}(0) = \Sigma(n) + 1 \Rightarrow$

$\Sigma(n + 1) \geq \Sigma(n) + 1 > \Sigma(n) \Rightarrow \Sigma(n + 1) > \Sigma(n)$

□

Proposition 12.2.2. $\exists Q \in \text{WHILE}^k : f = F_Q \Rightarrow \Sigma(n + k) \geq f(n)$

Proof. Let's consider $Q = (1, s) \in \text{WHILE}^k \wedge F_Q = f$, then

$Q' = (1, s')$

$s' :$

$\left. \begin{array}{l} \text{X1} := \text{X1} + 1; \\ \dots \\ \text{X1} := \text{X1} + 1; \end{array} \right\} n \text{ times}$

s

$Q' \in \text{WHILE}^n + k \wedge F_{Q'}(0) = f(n) \Rightarrow \Sigma(n + k) \geq f(n)$

□

Theorem 12.2.1. $\Sigma \notin F(\text{WHILE})$

Proof. By contradiction: let's consider

$$\left. \begin{array}{l} \Sigma \in F(\text{WHILE}) \\ \text{let } g(n) = 2n \Rightarrow g \in F(\text{WHILE}) \\ \text{let } f(n) = \Sigma(2n) \end{array} \right\} \Rightarrow f \in F(\text{WHILE}) \Rightarrow$$

$$\begin{array}{c} \exists Q \in \text{WHILE}^k : F_Q = f \Rightarrow \Sigma(n+k) \geq f(n), \forall n \in \mathbb{N} \Rightarrow \\ \quad \uparrow \text{prop. 12.2.2} \quad \quad \quad \uparrow f(n)=\Sigma(2n) \\ \Sigma(n+k) \geq \Sigma(2n), \forall n \in \mathbb{N} \Rightarrow \Sigma(2k+1) \geq \Sigma(2k+2) \Rightarrow \not\Leftarrow \\ \quad \uparrow n=k+1 \quad \quad \quad \uparrow \text{prop. 12.2.1} \end{array}$$

□

Proposition 12.2.3. $\Sigma \notin F(\text{WHILE}) \Rightarrow H^1$ is not solvable

Proof. By contradiction: let's consider $\Sigma \notin F(\text{WHILE}) \wedge H^1$ is solvable.

Since there are a finite number of (equivalent) programs of a given length, then the program that halts (with zero input) for each length can be found, and so the biggest output.

This implies $\Sigma \in F(\text{WHILE}) \Rightarrow \not\Leftarrow$

□

References

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(2), 113–124.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, 2(2), 137–167. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0019995859903626>

Chomsky, N., & Miller, G. (1958). Finite-state languages. *Information and Control*, 1, 91–112.

Drobot, V. (1989). *Formal languages and automata theory*. Rockville (MD): Computer Science Press.

Floyd, R. W., & Beigel, R. (1994). *The language of machines: an introduction to computability and formal languages*. New York: Computer Science Press.

Gurari, E. (1989). *An introduction to the theory of computation*. Rockville, MD: Computer Science Press.

Harel, D. (1992). *Algorithmics: the spirit of computing*. Reading, MA: Addison-Wesley.

Harrison, M. (1978). *Introduction to formal language theory*. Reading, MA: Addison-Wesley.

Hopcroft, J., & Ullman, J. (1979). *Introduction to automata theory, languages and computation*. Reading, MA: Addison-Wesley.

Kleene, S. (1952). *Introduction to metamathematics*. Walters-Noordhoff & North-Holland.

Ramos, G., & Morales, R. (2011). *Teoría de autómatas y lenguajes formales* (G. Ramos, Ed.).

Turing, A. M. (1936). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1), 230–265.