Lectures notes on Theory of automata and Formal languages

Computer Science Department
Malaga University

PGF version: 3.0.1a

October 16, 2017

Contents

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.

> Contributors during the course 2017-18: D. Arroyo Torres, A. Burgueño Romero, R. Ferreira Garcés, P. Gutierrez Ruiz, M. López Reviriego, M. Mejía Jiménez, D. Mérida Granados, J. Morales Joya, J. Pérez Bravo, A. Román Navas and M. Guerrero Ramírez.

Aditionaly, 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. $A \times \overset{n}{\cdots} \times A$
$\mathscr{P}(A)$	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)
$\mathscr{L}$	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

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)*

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

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

Definition 1.1.2. Proper subset *(subconjunto propio)*
B is a proper subset of A iff

$$B \subseteq A \wedge B \neq A \wedge B \neq \emptyset$$

Definition 1.1.3. Partition *(partición)*
$\Pi = \{A_1, A_2, \dots\} \subseteq \mathscr{P}(A)$ 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 $(a, a) \in R \ \ \forall a \in A$
- 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 $R \subseteq A \times A$,

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

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*)

$$f : A \to B \text{ is a function iff } \forall a \in A \; \exists! \; (a, b) \in f.$$

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

Definition 1.1.15. Domain and range of a function (*dominio y rango*)
Given $f : A \to B$,

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

Definition 1.1.16. Injective function (*función inyectiva*)
$f : A \rightarrowtail B$ 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*)
$f : A \twoheadrightarrow B$ iff $Rg(f) = B$

Definition 1.1.18. Bijective function (*función biyectiva*)
$f : A \leftrightarrow B$ iff it is injective and surjetive.

Example 1.1.2.

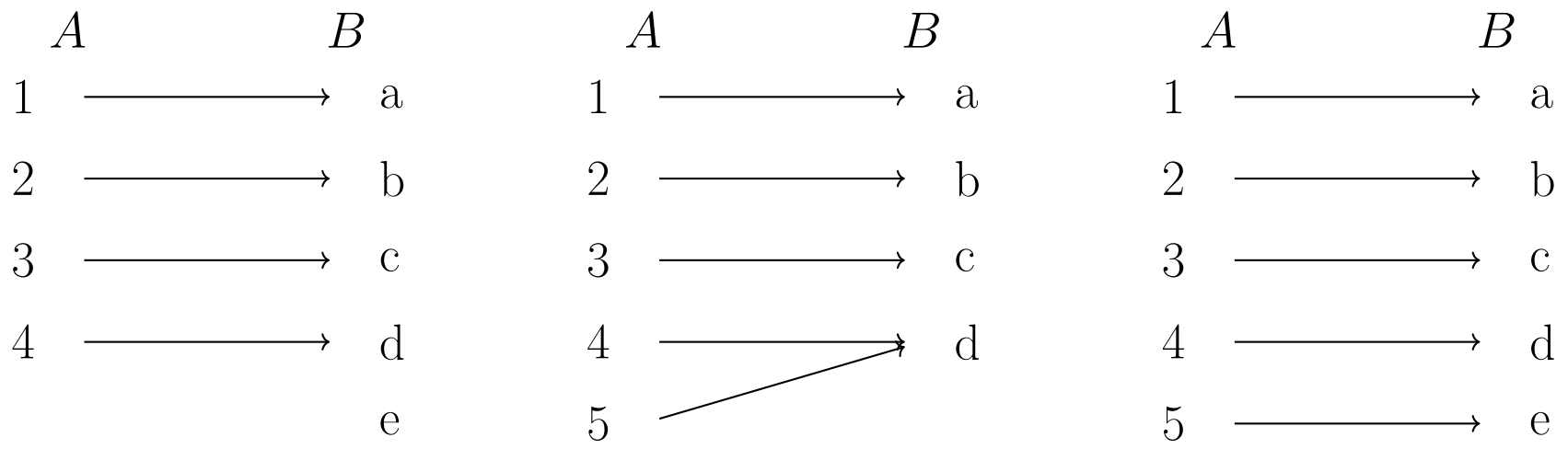


Figure 1.1: Three examples of functions: injective function (left), surjective function(center), and bijective function (right).

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

$$f : A \times A \to A$$

Definition 1.1.20. Associative property (*propiedad asociativa*)
Being A a closed set for the operation $\bullet$, $(\bullet : A \times A \to A)$, $\bullet$ 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*)

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

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 hipothesis.

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*)

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

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

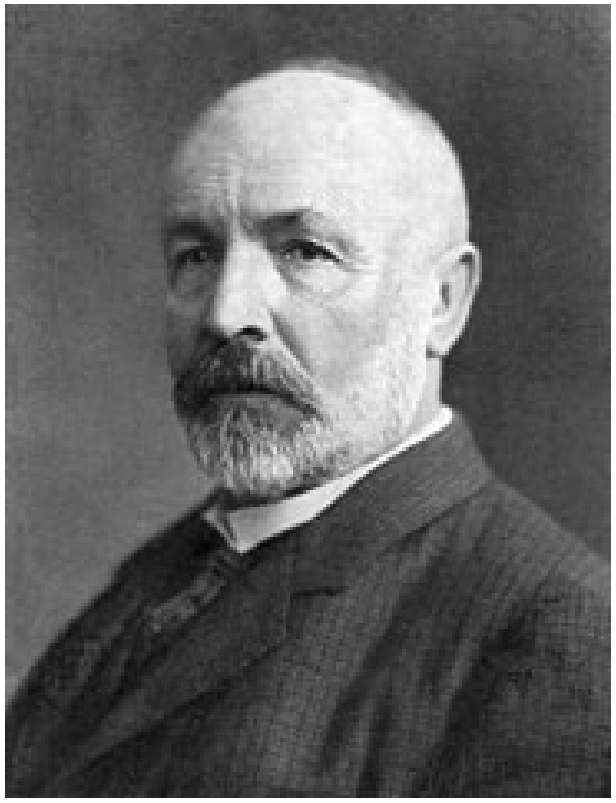


Figure 1.2: Georg Cantor defined the transfinite numbers by the end of the XIX century.

Chapter 2

Languages and grammars

2.1 Languages

2.1.1 Concept of language

Definition 2.1.1. Alphabet (*alfabeto*)
A non empty, finite set of symbols.

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

Definition 2.1.2. String over an alphabet (*cadena sobre un alfabeto*)
A finite sequence of symbols taken from a given alphabet,

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

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

Definition 2.1.3. Empty string (*cadena vacía*)

$$\varepsilon = (\ )$$

Definition 2.1.4. Length of a string

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

Definition 2.1.5. Occurrence (*ocurrencia*)
Given $w = (a_1, \ldots, a_n) \in \Sigma^+$, and $1 \leq j \leq n$,

$$w(j) = a_j$$

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

$$|w|_a = |A| : w(j) = a,\ \forall j \in A \subset \mathbb{N}$$

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 $(\Sigma^*, \cdot\,)$ 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

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

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

$$L \text{ is a language over } \Sigma \text{ iff } L \subseteq \Sigma^*$$

If a language is finite it can be defined by *extension* (e.g. $L = \{ac, bca, caaaad\}$).
If a language is infinite it can only be defined by *compresion* 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, lets assume an order on $\Sigma = \{a_1, \ldots, a_n\}$, then all strings of Σ^* can be enumerated, first by length and secondly by alphabetical order:

length	*strings*		
0	ε		
1	a_1	a_2	$\ldots$
2	a_1a_1	a_1a_2	$\ldots$
3	$\ldots$		

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

$\Sigma = \{a, b, c\}$

$v = bbbcbbba$ $\quad |v| = 8$

$w = cbbca$ $\quad |w| = 5$

$x = vw = bbbcbbbacbbca$

$x(4) = c$ $\quad |x|_b = 8$

$x(9) = c$ $\quad |x|_c = 3$

$bbbc \sqsubset x$ $\quad bacb \prec x$

$acbbca \sqsupset x$ $\quad acbbca \prec x$

$w^3 = cbbcacbbcacbbca$

$w^R = acbbc$

$L = \{\varepsilon, b, acc, ababc\}$ $\quad |L| = 4$

$L = \{w \in \Sigma : abc \sqsubset w\}$ $\quad |L| = \aleph_0$

$L = \{w \in \Sigma : |w|_a = 3n \in \mathbb{N}_0\}$ $\quad |L| = \aleph_0$

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)

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

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

$$\mathscr{R} : \Sigma_M^* \to \mathscr{P}(\Sigma^*)$$

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

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

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

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

Remark.

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

Proposition 2.1.3. There are uncountable non representable languages

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

$$|\mathscr{L}.REP| \leq |REP|$$

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

$$|REP| \leq \aleph_0$$

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

$$\aleph_0 \leq |\mathscr{L}.REP|$$

and then

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

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

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

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

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

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

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

□

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 (*reconocededor 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 $\alpha \to \beta$

— $S \in N$ 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

$x = uzv$
$y = u\beta v$
$\exists z \to \beta \in P$

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

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

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

$$w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$$

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*)

$$\alpha \to \beta$$

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*)

$$\alpha A \beta \to \alpha \gamma \beta$$

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*)

$$A \to \alpha$$

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

$$A \to aB$$

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

$$A \to Ba$$

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

$$A \to a$$

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

$$A \to \varepsilon$$

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

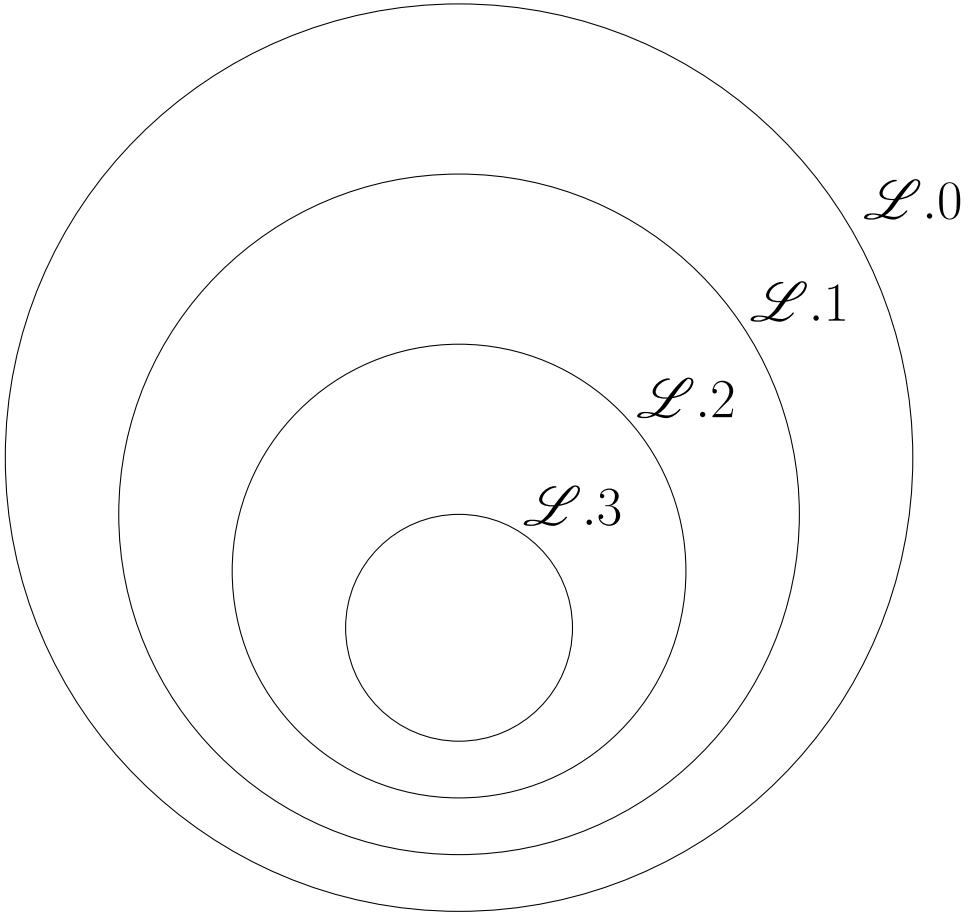
$$\text{Type 3 grammar} \subset \text{Type 2 grammar} \subset \text{Type 1 grammar} \subset \text{Type 0 grammar}$$

2.2.4 Classification of languages

Definition 2.2.24. Type i ($\mathscr{L}.i$) language
A language L is type $i \in \{0 \dots 3\}$ ($L \in \mathscr{L}.i$) if there exists a type i grammar such that $\mathcal{L}(G) = L - \{\varepsilon\}$.

Proposition 2.2.3. Chomsky's formal languages hierarchy

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



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
Finitness	$\|\mathcal{L}(G)\| \in \mathbb{N}$	no	no	yes	yes
Regularity	$\mathcal{L}(G) \in \mathscr{L}.3$	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 '$\cdot$' might be omitted, leaving $L_1 L_2$.

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*)

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

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 languaje 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 $\mathscr{L}.3$	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

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

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

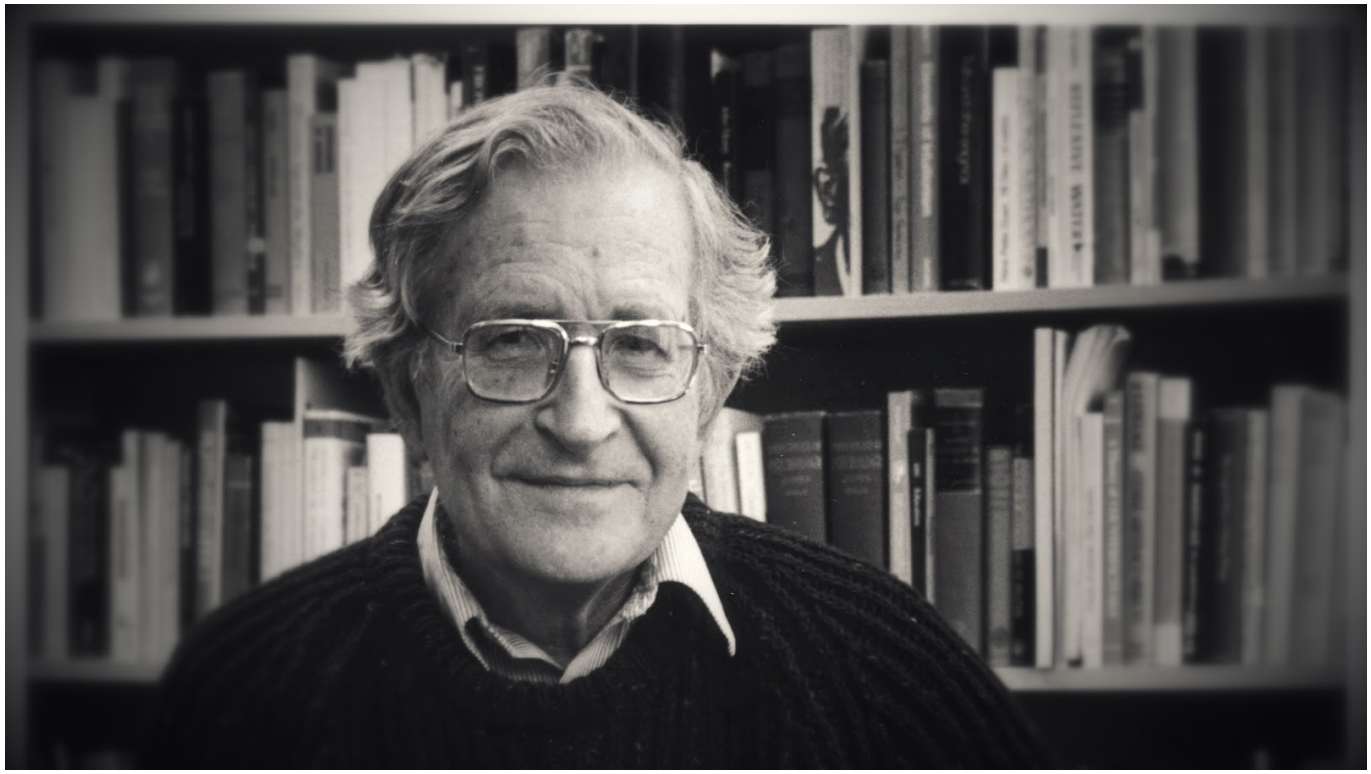


Figure 2.1: Noam Chomsky proposed the generative grammar in 1950.

Chapter 3

Regular expressions

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

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

verifying

1. $\varnothing \in \mathcal{R}$
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

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

verifying

1. $\mathcal{L}(\varnothing) = \varnothing$
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) $\alpha + \varnothing = \alpha$

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

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

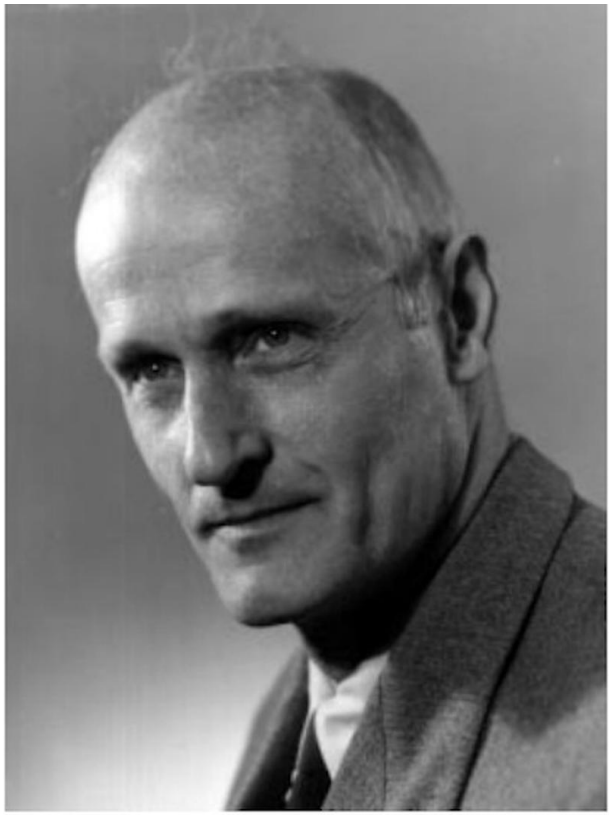


Figure 3.1: Stephen Kleene proposed the regular expressions model in 1951.

Chapter 4

Finite automata

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

$\delta : K \times \Sigma \to K$ 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

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

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 $\exists\, C_1, C_2, \ldots, C_{n-1}$ configurations of that DFA such that

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

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 $C_0, C_1, C_2, ..., C_n$ such that $C_i \vdash C_{i+1}$ $\forall i \in \{0, ..., n-1\}$ with $n \geq 0$.
It will be noted as $C_0 \vdash C_1 \vdash C_2 \vdash ... \vdash C_n$, 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

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

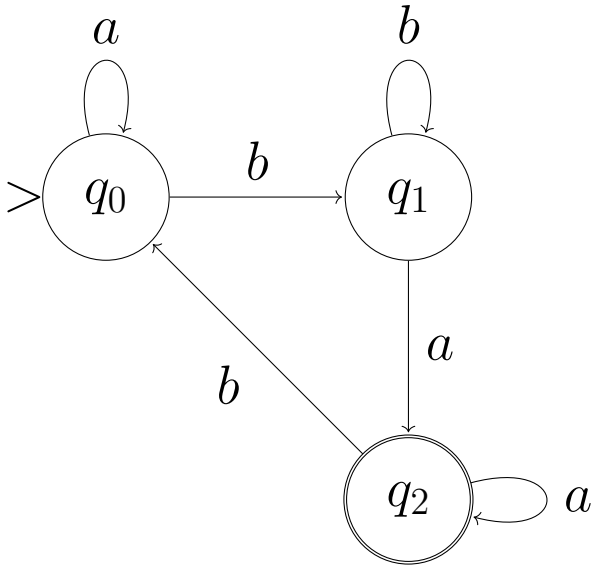
Definition 4.1.14. Set of languages accepted by DFA
The set of the languajes accepted by DFA, denoted as $\mathscr{L}(DFA)$, is defined as:

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

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 conects (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 "$>$".

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 automatons

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 $\exists\, C_1, C_2, \ldots, C_{n-1}$ configurations of that NFA :

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

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 $C_0, C_1, C_2, ..., C_n$ such as $C_i \vdash C_{i+1}$ $\forall i \in \{0, ..., n-1\}$ with $n \geq 0$.
It will be noted as $C_0 \vdash C_1 \vdash C_2 \vdash ... \vdash C_n$, 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 languajes accepted by NFA is defined as:

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

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 conects (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 ">".

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

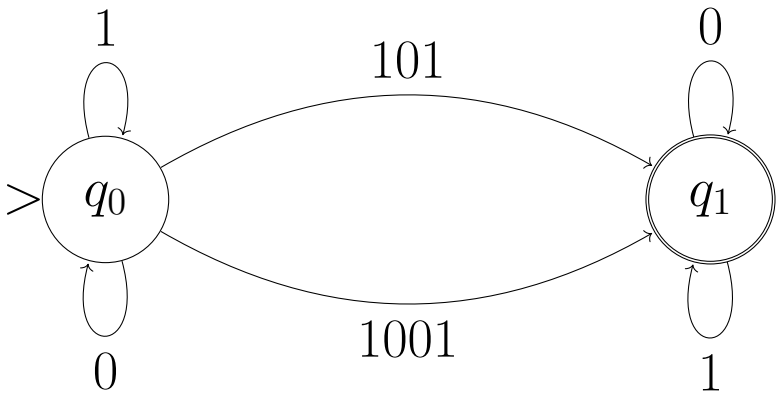
$K = \{q_0, q_1\}$

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

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

$s = q_0$

$F = \{q_1\}$



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

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

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

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

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

Z

Chapter 5

Regularity conditions

5.1 Myhill-Nerode

Definition 5.1.1. Indistinguishable strings (*cadenas 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

$$\exists z \in \Sigma^* \ : \ (xz \in L \wedge yz \notin L) \vee (xz \notin L \wedge yz \in L)$$

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: $L \in \mathscr{L}.3 \Leftrightarrow |\Pi_L| \in \mathbb{N}$

Proof. It has to be proven in both directions.

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

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

Let $M = (K = \{q_0, \ldots, q_n\}, \Sigma, \delta, q_0, F)$ and $\delta^* : K \times \Sigma^* \to K$ 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 $i = 0, \ldots, n$, and $\Pi = \{A_0, \ldots, A_n\}$.
M is a DFA, so Π is a partition of Σ^* and

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

$$\begin{aligned} \text{Let } \Pi' = \{B \subseteq \Sigma^* \,:\, & B = A_{i_1} \cup \ldots \cup A_{i_k}, A_{i_j} \in \Pi \quad \wedge \\ & \forall x, y \in B \; (x, y) \in I_L \quad \wedge \\ & (x \in B \wedge y \notin B) \Rightarrow (x, y) \notin I_L\} \end{aligned}$$

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

$|\Pi_L| \in \mathbb{N} \Rightarrow |\Pi_L| = m \geq 1 \Rightarrow \Pi_L = \{A_0, \ldots, A_{m-1}\}$
Let $\Sigma = \{a_1, \ldots, a_n\}$ the alphabet over which L is defined.
Let $M = (K, \Sigma, \delta, s, F)$ be a DFA with:

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

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

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

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

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

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

Remark.

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

M is a $MDFA$ that represents L.

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

Theorem 5.2.1. Pumping
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 $\mathscr{L}.3$:
$L \in \mathscr{L}.3 \Rightarrow L$ verifies the regular pumping condition.

Example 5.2.1. Determine that $L = \{0^n 1^n : n \in \mathbb{N}\} \notin \mathscr{L}$.3
It must be shown that

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

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

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

But we can se 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

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 $X_1, X_2, ..., X_k$, then:

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

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.

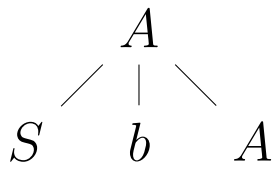
Definition 6.1.3. A-derivation
An A-derivation associated to a derivation A-subtree is the finite sequence of direct productions starting from A used to obtained the A-subtree.

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).

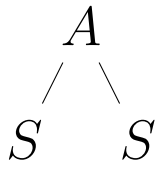
Definition 6.1.6. Left-most derivation. Right-most derivation
If in every step of a derivation it is appliead a rule of production to the non terminal character most to the left/right then the derivation is a left/right-most derivation.

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



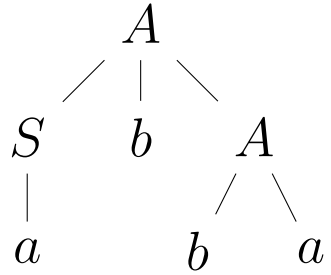
Product: SbA

A-derivation: $A \Rightarrow SbA$



Product: SS

A-derivation: $A \Rightarrow SS$



Product: $abba$

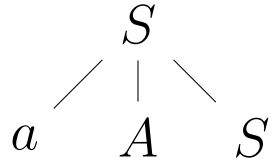
A-derivations:

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

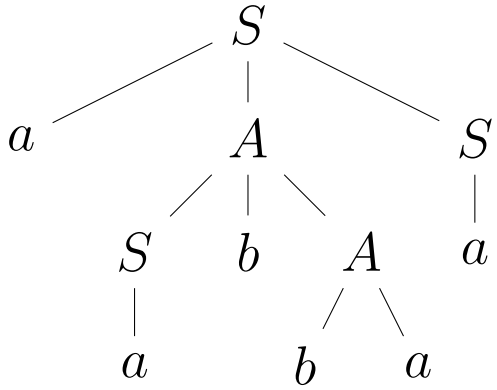
$A \Rightarrow \boldsymbol{S}bA \Rightarrow ab\boldsymbol{A} \Rightarrow abba$

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:

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

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

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

Definition 6.1.7. Ambiguous grammar
Let $G = (N, T, P, S)$ 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 $G = (\{S, A, B\}, \{a\}, \{S \to A : B, A \to a, B \to a\}, S)$.

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

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 $A \to B \ : \ A, B \in N$.

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

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

Proposition 6.4.1. $\forall G$ CFG $\exists G'$ 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

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

Proposition 6.4.2. $\forall G$ CFG $\exists G'$ in GNF : $G \equiv G'$

6.5 Closure properties

Theorem 6.5.1. $\mathscr{L}.2$ is closed for the operations of union, concatenation, and Kleene star.

Proposition 6.5.1. $\mathscr{L}.2$ 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

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

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

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

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 $C_0, C_1, \ldots, C_n : C_i \vdash C_{i+1}, i \in \{0 \ldots n-1\}$, noted as

$$C_0 \vdash C_1 \vdash \cdots \vdash C_n$$

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 $\mathscr{L}(NDPA)$, is defined as:

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

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

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
$\exists n \in \mathbf{N} \mid \forall x \in L \; with \; |x| \geq n, \exists u, v, w, y, z \in \Sigma^*$ that:

1) $x = uvwyz$

2) $|uwy| < n$

3) $|vy| > 0$

4) $\forall m \geq 0 \;\; uv^m wy^m z \in L$

Pumping lemma for $\mathscr{L}.2$:
$L \in \mathscr{L}.2 \Rightarrow L$ satisfies CFPC

Chapter 7

The Turing machine

7.1 Formal definition

Definition 7.1.1. Turing Machine (TM)
Being $\Sigma_{\mathrm{R}} = \{a_0, l, r, h\}$ an alphabet resembling reserved words, a TM is a 5-tuple $(K, q_0, \Sigma, \gamma, \delta)$, where

K is a non-empty set of states,

$q_0 \in K$ is the initial state,

Σ is an alphabet, with $\Sigma_{\mathrm{R}} \cap \Sigma = \varnothing$,

$\gamma : K \times \Sigma_{\mathrm{T}} \to \Sigma'$ is the instruction function,

$\delta : K \times \Sigma_{\mathrm{T}} \to K$ is the transition function

where $\Sigma_{\mathrm{T}} = \Sigma \cup \{a_0\}$ and $\Sigma' = \Sigma \cup \Sigma_{\mathrm{R}}$

Remark. a_0 is often referred to as blank symbol, to state that no meaningful symbol is present in the cell where it is stored.

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 one cell left (l) or right (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 go to the state $\delta(q, a) \in K$.

We could represent a TM with a matrix of four columns and $\left\|K\right\| \cdot \left\|\Sigma_{\mathrm{T}}\right\|$ 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 $E : \mathbb{Z} \to \Sigma_{\mathrm{T}}$, satisfying

$$\left\| \{ z \in \mathbb{Z} \mid E(z) \neq a_0 \} \right\| \in \mathbb{N}$$

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 $C = (q, E, z)$ where:
$q \in K$;
E is a tape expression of M;
$z \in \mathbb{Z}$

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

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 $\iff \gamma\big(q, E(z)\big) = h$

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

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

2. $$\big(z' = z + 1 \ \wedge \ \gamma\big(q, E(z)\big) = r\big) \quad \vee$$
$$\big(z' = z \quad \wedge \ \gamma\big(q, E(z)\big) \in \Sigma_{\mathrm{T}}\big)$$

3. $$E'(x) = \begin{cases} E(x) & \text{if} \quad x \neq z \\ E(x) & \text{if} \quad x = z \ \wedge \ \gamma\big(q, E(z)\big) \in \{l, r\} \\ \gamma\big(q, E(x)\big) & \text{if} \quad x = z \ \wedge \ \gamma\big(q, E(z)\big) \in \Sigma_{\mathrm{T}} \end{cases}$$

Definition 7.1.7. To transit in n steps
We say that a TM M transits in n steps from a configuration C to C', denoted as $C \vdash^n C'$, iff:
$\exists C_1, C_2, ..., C_{n-1}$ configurations of M:

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

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 $C_0 \vdash C_1 \vdash ... \vdash C_n$, as a finite sequence of configurations $C_0, C_1, ..., C_n$, 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. No halt after a configuration
Let M be a TM, and C a configuration of M. We say that M does not halt after C iff:

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

Definition 7.1.17. Halt after a configuration
Let M be a TM, and C a configuration of M. We say that M halts after C iff:

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

Definition 7.1.18. 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 $C \vdash^* C'$.

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

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

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

Definition 7.1.19. 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 7.1.20. 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 $w_y = E(x)E(x+1)...E(y)$, and $M = (K, q_0, \Sigma, \gamma, \delta)$.

Definition 7.1.21. Place a TM behind strings
Let $M = (K, q_0, \Sigma, \gamma, \delta)$ be a TM, and $w_1, w_2, ..., w_n$ strings over Σ. We say that we place M behind $w_1, w_2, ..., w_n$, iff we place it behind $w = w_1 a_0 w_2 a_0 ... a_0 w_n$.

7.2 Computability, decidability and enumerability

Definition 7.2.1. Turing computable function
$f : \mathbb{N}^n \to \mathbb{N}$ is Turing computable iff, exists a TM, such that $\forall(x_1, ..., x_n) \in \mathbb{N}^n$:

$$M \text{ halts behind } f(x_1, ...x_n), \text{if we placed it behind } x_1, ..., x_n$$

$F(MT)$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 won't halt.

Definition 7.2.2. *T-MT*

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

Definition 7.2.3. Turing decidable set
$A \subseteq \mathbb{N}^n$ is Turing decidable iff exists a TM M, such that $\forall(x_1, x_2, ..., x_n) \in \mathbb{N}^n$, M placed behind the strings $x_1, x_2, .., x_n$, it halts in a terminal configuration (q, E, z) and:

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

$DEC(MT)$ 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 $\forall(x_1, x_2, ..., x_n) \in \mathbb{N}^n$, M placed behind the strings $x_1, x_2, ..., x_n$:

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

$ENU(MT)$ 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 $P \in PRED(T\text{-}MT)$. We define this set as:

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

Definition 7.2.6. Turing enumerable predicate
A predicate P is Turing decidable, iff $P \in PRED(MT)$. We define this set as:

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

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

Chapter 8

Recursive functions

8.1 Formal definition

Definition 8.1.1. Zero function (θ)
$\theta : N^0 \to N$
$\theta() = 0$

Definition 8.1.2. Successor function
$\sigma : N \to N$
$\sigma(n) = n + 1$

Definition 8.1.3. Projection function π_i^k
For each pair $(k, i) \in N^2$ with $k \geq 1$ and $1 \leq i \leq k$ the $k-$ary projection function that returns the i-th argument is defined as:
$\pi_i^k : N^k \to N$
$\pi_i^k(\underline{n}) = n_i$ where $\underline{n} = (n_1, n_2, ..., n_k)$
The set of of all the projection functions is represented as π, that is:
$\pi = \{\pi_i^k \mid (k, i) \in N^2 \text{ with } k \geq 1 \wedge 1 \leq i \leq k\}$

Definition 8.1.4. Initial functions
The set of the initial functions is defined like:
$INI = \pi \cup \{\theta, \sigma\}$

Definition 8.1.5. Composition
Let $m > 0$, $k \geq 0$ and the functions:
$g : N^m \to N$
$h_1, h_2, ..., h_m : N^k \to N$
If the function $f : N^k \to N$ is:

$$f(\underline{n}) = g(h_1(\underline{n}), h_2(\underline{n}), ..., h_m(\underline{n}))$$

then it said that f is obtained from composition of g and h_1, h_2, ..., h_m
We will express it as $f(\underline{n}) = g(h_1, h_2, ..., h_m)(\underline{n})$, or simply $f = g(h_1, h_2, ..., h_m)$.

Definition 8.1.6. Primitive recursion
Let $k \geq 0$ and the functions
$g : N^k \to N$
$h : N^{k+2} \to N$
If the function $f : N^{k+1} \to N$ is:

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

then it said that f is obtained from g and h by primitive recursion.
We will express it as $f(\underline{n}) = \langle g|h\rangle(\underline{n})$, or simply $f = \langle g|h\rangle$.

Definition 8.1.7. Unbounded minimization
Let $k \geq 0$ and the function:
$g : N^{k+1} \to N$
If the function $f : N^k \to N$ is:

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

$$\text{where } A = \{t \in N \mid g(\underline{n}, t) = 0\} \text{ and } \underline{n} \in N^k$$

then it said that f is obtained from g by bounded 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.

Definition 8.1.8. Total function, Partial functions
Let the function $f : N^k \to N : k > 0$
then we say that f is a total function if $Dom(f) = N^k$
Let the function $g : N^k \to N$
then we say that g is a total function if $g() \in N$

It said that a function is partial iff it is not total.

Definition 8.1.9. Recursive functions, REC
REC is defined like:

1) $INI \subset REC$

2) $g, h_1, h_2, ..., h_m \in REC \wedge \exists g(h_1, h_2, ..., h_m) \Rightarrow g(h_1, h_2, ..., h_m) \in REC$

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 belongs to REC

It said that f is a recursive function iff $f \in REC$

Definition 8.1.10. $TREC$
$TREC = \{f \in REC \mid f \text{ is total}\}$

8.2 Enumerability, decidability and computability with REC

Definition 8.2.1. Set of truth values of a predicate
Let $k \geq 0$ and P be a predicate of k arguments The set of truth values of a predicate P, noted V_P, is defined like:

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

Definition 8.2.2. Predicate associated to a function
Let $k \geq 0$ and f be a function of k arguments.
The predicate associated with a function f, noted P_f, is defined as:

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

Definition 8.2.3. Characteristic function of a set
Let $k \geq 0$ and $V \subseteq N^k$
The characteristic function of V, noted X_V, is defined like:

$$X_V : N^k \to N$$

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

Definition 8.2.4. Characteristic function of a predicate
Given a predicate P, we call X_V characteristic function of predicate P

Definition 8.2.5. Recursively decidable predicate, $PRED(TREC)$
$PRED(TREC)$ is defined as:

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

It said that P is a recursively decidable predicate iff $P \in PRED(TREC)$

Definition 8.2.6. Predicate recursively enumerable, $PRED(REC)$
$PRED(REC)$ is defined as:

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

It said that P is a recursively enumerable predicate iff $P \in PRED(REC)$

Remark. $PRED(TREC) \subset PRED(REC)$ since $TREC \subset REC$.

Definition 8.2.7. Recursively decidable set, $DECI$
$DECI$ is defined like:

$$DECI = \{V_P \mid P \in PRED(TREC)\}$$

It said that $V \subseteq N^k$ is a set recursively decidable iif $V \in DECI$

Definition 8.2.8. Recursively enumareable set, $ENUM$
$ENUM$ is defined as:

$$ENUM = \{V_P \mid P \in PRED(REC)\}$$

It said that $V \subseteq N^k$ is a recursively enumerable set iff $V \in ENUM$.

Remark. $DECI \subset ENUM$ since $PRED(TREC) \subset PRED(REC)$.

Definition 8.2.9. Recursively generable set
It said that V is a recursively generable set iff V is recursively enumerable

8.3 Examples

Example 8.3.1. Definition of the add function
The add function defined like:
$add : N^2 \to N$
$add(n_1, n_2) = n_1 + n_2$
It is a recursive function, because it can be defined like:
$add = \langle \pi_1^1 | successor_3 \rangle$ where
$successor_3 : N^3 \to N$
$successor_3(n_1, n_2, n_3) = n_3 + 1$
It is also a recursive function, because it can be defined like:
$successor_3 = \sigma(\pi_3^3)$ and the complete expresion is:
$add = \langle \pi_1^1 | \sigma(\pi_3^3) \rangle$

Example 8.3.2. Add two numbers with add function

It is an evaluation step by step of the function add defined in the last example

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

Chapter 9

The WHILE language

9.1 Formal definition

For the formal definition of this model we will use some concepts related to the formal languages that we have seen in previous chapters.
For the definition of some auxiliary grammars that we will use, we need to introduce two alphabets:

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

$$\Sigma_1 = \{X, :=, 1, +, -, while, \neq, 0, do, od, ;\}$$

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

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

$$T = \Sigma_d \cup \Sigma_1$$

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

Definition 9.1.1. *Wile Code*, Set of *While* codes (COD_WHILE)
Let $s \in T^+$, we can say that s is a *While* code iff $s \in L(G)$.
COD_WHILE, is defined as:

$$COD\_WHILE = L(G)$$

Remark. COD_WHILE contains all the syntactically correct codes in the $WHILE$ language.

Definition 9.1.2. $WHILE$ program, Set of $WHILE$ programs ($WHILE$)
$WHILE$, is defined as:
$WHILE = \{(n, p, s) \in \mathbb{N} \times \mathbb{N} \times COD\_WHILE \mid p \geq n \ \wedge \ p > 0 \ \wedge$
$(X_m \prec s \ : \ m \in \Sigma_d^+ \Rightarrow m \leq p)\}$.
We can say that Q is a $WHILE$ program iff $Q \in WHILE$.

Remark. .The expression "$X_m \prec s$" means that the string "X_m" is substring of the string "s".

Definition 9.1.3. Size of a $While$ code
Let $s \in COD\_WHILE$, the size of s, $size(s)$, is defined as:

1. If in G it is fulfilled that $< assignment > \Rightarrow^* s$, then:
   $size(s) = 1$

2. If $s = while\ X_i \neq 0$ do s_1 od, with $i \in \Sigma_d^+$ and $s_1 \in COD\_WHILE$, then:
   $size(s) = size(s_1) + 2$

3. If $s = s_1; s_2$, with $s_1, s_2 \in COD\_WHILE$, then:
   $size(s) = size(s_1) + size(s_2)$

Definition 9.1.4. Size of a $WHILE$ program ($size$)
Let $Q \in WHILE$, with $Q = (n, p, s)$.
It's defined the size of Q, $size(Q)$, as:

$$size(Q) = size(s)$$

Definition 9.1.5. Line function ($line$)
It's defined the line function: $COD\_WHILE \times \mathbb{N} \to T_2^*$ as:

1. If in G it is fulfilled that $< assignment > \Rightarrow^* s$, then:

$$line(s, n) = \begin{cases} s & if \quad n = 1 \\ \\ \varepsilon & otherwise \end{cases}$$

2. If $s = while\ X_i \neq 0$ do s_1 od, with $i \in \Sigma_d^+$ and $s_1 \in COD\_WHILE$, then:

$$line(s,n) = \begin{cases} while\ X_i \neq 0\ do & if & n = 1 \\ od & if & n = size(s) \\ line(s_1, n-1) & if & 1 < n < size(s) \\ \varepsilon & & otherwise \end{cases}$$

3. If $s = s_1; s_2$, with $s_1, s_2 \in COD\_WHILE$, then:

$$line(s,n) = \begin{cases} line(s_1) & if & 0 < n \leq size(s_1) \\ line(s_2, n - size(s_1)) & if & size(s_1) < n \leq size(s) \\ \varepsilon & & otherwise \end{cases}$$

Remark. The line function returns the text from a specific line or an empty string if the line number is invalid.

Definition 9.1.6. Jump function
It's defined the jump function: $COD\_WHILE \times \mathbb{N} \to T_2^*$ as:

1. If in G it is fulfilled that $< assignment > \Rightarrow^* s$, then:

$$jump(s, n) = 0$$

2. If $s = while\ X_i \neq 0$ do s_1 od, with $i \in \Sigma_d^+$ and $s_1 \in COD\_WHILE$, then:

$$jump(s,n) = \begin{cases} size(s)+1 & if \quad n = 1 \\ 1 & if \quad n = size(s) \\ jump(s_1, n-1)+1 & if \quad (1 < n < size(s)) \wedge (jump(s_1, n-1) \neq 0) \\ 0 & otherwise \end{cases}$$

3. If $s = s_1; s_2$, with $s_1, s_2 \in COD\_WHILE$, then:

$$jump(s,n) = \begin{cases} jump(s_1, n) & if \quad 0 < n \leq size(s_1) \\ jump(s_2, n - size(s_1)) + size(s_1) & if \quad (size(s_1) < n \leq size(s)) \\ & \qquad \wedge (jump(s_2, n - size(s_1)) \neq 0) \\ 0 & otherwise \end{cases}$$

Remark. The jump 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 9.1.7. Configuration, Set of configurations
Let $Q \in WHILE$, with $Q = (n, p, s)$.
It's defined the set of Q configurations, C_Q, as:
$C_Q = \{\underline{c} \in \mathbb{N}^{p+1} \mid \underline{c} = (m, \underline{x}) \ \ with \ \ 1 \leq m \leq size(Q) + 1 \ \ and \ \ \underline{x} \in \mathbb{N}^P\}$

We will say that $\underline{c}$ is a configuration of Q iff $c \in C_Q$.

Definition 9.1.8. Initial configuration
Let $Q \in WHILE$, with $Q = (n, p, s)$, and be $\underline{c} \in C_Q$, with $\underline{c} = (m, \underline{x})$.
We will say that $\underline{c}$ is a initial configuration of Q iff:

$$m = 1 \quad \wedge \quad x_{n+1} = \ldots = x_p = 0$$

Definition 9.1.9. Terminal configuration, non-terminal configuration
Let $Q \in WHILE$, and be $\underline{c} \in C_Q$, with $\underline{c} = (m, \underline{x})$.
We will say that $\underline{c}$ is a terminal configuration of Q iff $m = size(Q) + 1$.
Otherwise we say that it is a non-terminal configuration.

Remark. A configuration can not be initial and terminal at the same time.

Definition 9.1.10. Direct transition
Let $Q \in WHILE$, with $Q = (n, p, s)$.
Let $\underline{c_1}, \underline{c_2} \in C_Q$, with $\underline{c_1} = (m, \underline{x})$ and $\underline{c_2} = (t, \underline{z})$.
We will say that Q transits directly from c_1 to c_2, $c_1 \vdash c_2$, iff

$line(s, m) = X_i := 0$ with $i \in \Sigma_d^+$
$\Rightarrow\ t = m + 1\ \wedge\ z_i = 0\ \wedge\ (z_r = x_r\ \forall r \neq i)$

$line(s, m) = X_i := X_j$ with $i, j \in \Sigma_d^+$
$\Rightarrow\ t = m + 1\ \wedge\ z_i = x_j\ \wedge\ (z_r = x_r\ \forall r \neq i)$

$line(s, m) = X_i := X_j + 1$ with $i, j \in \Sigma_d^+$
$\Rightarrow\ t = m + 1\ \wedge\ z_i = x_j + 1\ \wedge\ (z_r = x_r\ \forall r \neq i)$

$line(s, m) = X_i := X_j - 1$ with $i, j \in \Sigma_d^+$
$\Rightarrow\ t = m + 1\ \wedge\ z_i = x_j - 1\ \wedge\ (z_r = x_r\ \forall r \neq i)$

$line(s, m) = while\ X_i \neq 0\ do\ \ with\ \ i \in \Sigma_d^+$
$\Rightarrow\ (x_i \neq 0\ \Rightarrow\ t = m + 1)\ \wedge\ (x_i = 0\ \Rightarrow\ t = jump(s, m))\ \wedge\ \underline{z} = \underline{x}$

$line(s, m) = od$
$\Rightarrow t = jump(s, m)\ \wedge\ \underline{z} = \underline{x}$

Remark. Assignments make the requested assignment on the appropriate variable and go to the next line. The loop header does not touch the contents of the variables and passes to the next line if the control variable of the loop is nonzero, and to the next line to its queue if the control variable of the loop is zero. The loop tail does not touch the content of the variables and goes to the line of its header.

Definition 9.1.11. Next function ($NEXT$)
Let $Q \in WHILE$, with $Q = (n, p, s)$.
Let C_Q the set of all Q configurations.
It's defined the next function of Q, $NEXT_Q$, as:

$$NEXT_Q : \mathbb{N}^{p+1} \to \mathbb{N}^{p+1}$$

$$NEXT_Q(\underline{c}) = \begin{cases} \underline{c'} & if \quad \underline{c} \vdash \underline{c'} \; with \; \underline{c}, \underline{c'} \in C_Q \\ \underline{c} & if \quad \nexists \underline{c'} \in C_Q \mid \underline{c} \vdash \underline{c'} \; with \; \underline{c} \in C_Q \\ \underline{c} & if \quad \underline{c} \notin C_Q \end{cases}$$

Definition 9.1.12. Calculation function (CAL)
Let $Q \in WHILE$, with $Q = (n, p, s)$.
It's defined the calculation function of Q, CAL_Q, as:
$CAL_Q : \mathbb{N}^{n+1} \to \mathbb{N}^{p+1}$

$$CAL_Q(\underline{x}, i) = \begin{cases} (1, \underline{x}, \underline{0}) & if \; i = 0 \\ NEXT_Q(CAL_Q(\underline{x}, i-1)) & if \; i > 0 \end{cases}$$

Remark. $i \in \mathbb{N}$, $\underline{x} \in \mathbb{N}^n$ *and* $\underline{0} \in \mathbb{N}^{p-n}$.

Definition 9.1.13. Temporal complexity function (T)
Let $Q \in WHILE$, with $Q = (n, p, s)$.
It's defined the temporal complexity function of Q, T_Q, as:
$T_Q : \mathbb{N}^n \to \mathbb{N}$

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

where $A = \{j \in \mathbb{N} \mid \pi_1^{p+1}(CAL_Q(\underline{x}, j)) = size(Q) + 1\}$

Remark. $\underline{x} \in \mathbb{N}^n$.

Definition 9.1.14. Calculated function (F)
Let $Q \in WHILE$, with $Q = (n, p, s)$.
It's defined the calculated function of Q, F_Q, as:
$F_Q : \mathbb{N}^n \to \mathbb{N}$
$F_Q(\underline{x}) = \pi_2^{p+1}(CAL_Q(\underline{x}, T_Q(\underline{x})))$

Remark. $\underline{x} \in \mathbb{N}^n$ and $j \in \mathbb{N}$.

9.2 Example of **WHILE** program

9.3 Computability, decidability and enumerability with **WHILE**

F(MT)	REC	F(WHILE)
DEC(MT)	DECI	DEC(WHILE)
ENUM(MT)	ENUM	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 Expanded WHILE language

The first extension is the use of macrosenteces which are asignation sentences in which a variable is asigned the value obtained from a call to a $WHILE$-computable function.

Q = (1, 3, s')

s':

$X_3 := X_1$;
$X_1 := 0$;
while $X_2 \neq 0$ **do**
$X_1 := suma(\ X_1, X_3\ )$;
$X_2 := X_2 +(\ X_3 \times X_1\ )$;
od

The second extension is the free name of variables, to avoid confussion it is necessary to specify at the begging of the program the input variables and the output variable, the variables not specified as input or output are the auxiliar variables.

Input: data
Output: double

Code: ...

The third extension is include remarks between the symbols (*$remark$*):
$X_1 := X_2 \times X_1$; (*This is the remark*)

The fourth and last extension is allowing the use of four more control sentences:

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 a boolean condition as long as it correspond with a WHILE-enumerable predicate, and $f(x)$ is a WHILE-computability function.

Noted by $WHILE\_A$ the set of all the programmes in expand WHILE language. $F(WHILE\_A)$ the set of all the functions represented by $WHILE\_A$

Although $WHILE \subset WHILE\_A$ is true that $F(WHILE\_A) = F(WHILE)$

10.2 Recursive $\Rightarrow$ WHILE-calculable

Theorem 10.2.1.
$REC = F(WHILE)$

Proposition 10.2.1.
$REC \subseteq F(WHILE)$

Proposition 10.2.2.
$F(WHILE) \subseteq REC$

Proposition 10.2.3.
$INI \subseteq F(WHILE)$

Proof.
$a)\ \theta \in F(WHILE)$
Let the programme $Q = (0,\ 1,\ X_1 := 0)$.
$F_Q = \theta \Rightarrow \theta \in F(WHILE)$

$b)\sigma \in F(WHILE)$
Let the programme $Q = (1,\ 1,\ X_1 := X_1 + 1)$.
$F_Q = \sigma \Rightarrow \sigma \in F(WHILE)$

$c)\pi \subseteq F(WHILE)$
Let the programme $Q = (k,\ k,\ X_1 := X_i)$, whith $k \geq 1$ y $1 \leq i \leq k$.
$F_Q = \pi_i^k \Rightarrow \pi_i^k \in F(WHILE) \Rightarrow \pi \subseteq F(WHILE)$

$$INI = \pi \cup \{\theta,\ \sigma\} \underset{\text{for } a), b) \, y \, c)}{\Rightarrow} INI \subseteq F(WHILE)$$

□

Proposition 10.2.4.
$f = g(h_1,\ h_2,\ ...,\ h_m)\ \ \wedge\ \ g,\ h_1,\ h_2,\ ...,\ h_m \in F(WHILE) \Rightarrow f \in F(WHILE)$

Proof.
Let $m > 0,\ \ k \geq 0$ and the functions:
$g : N^m \to N$
$h_1,\ h_2,\ ...,\ h_m : N^k \to N$
and let $f = g(h_1,\ h_2,\ ...,\ h_m)$.
Let the macroprogramme $Q = (k,\ k+m,\ s)$ with s:

$X_{k+1} := h_1(X_1,\ X_2,\ ..., X_k);$
$X_{k+2} := h_2(X_1,\ X_2,\ ..., X_k);$
...
$X_{k+m} := h_m(X_1,\ X_2,\ ..., X_k);$
$X_1 := g(X_{k+1},\ X_{k+2},\ ..., X_{k+m})$

$F_Q = f \Rightarrow f \in F(WHILE)$ □

Proposition 10.2.5.
$f = \langle g|h\rangle \wedge g,\ h \in F(WHILE) \Rightarrow f \in F(WHILE)$

Proof.
Let $k \geq 0$ and the functions:

$g : N^k \to N$
$h : N^{k+2} \to N$
and let $f = \langle g|h \rangle$
Let the macroprgramme $Q = (k+1,\ k+3,\ s)$ con s:

$X_{k+2} = g(X_1,\ X_2,\ ..., X_k);$
do X_{k+1} *times*
$X_{k+2} := h(X_1,\ X_2,\ ...,\ X_k, X_{k+3},\ X_{k+2});$
$X_{k+3} := X_{k+3} + 1$
od;
$X_1 := X_{k+2}$

$F_Q = f \Rightarrow f \in F(WHILE)$

□

Proposition 10.2.6.
$f = \mu[g] \wedge g \in F(WHILE) \Rightarrow f \in F(WHILE)$

Proof.
Let $k \geq 0$ and the function
$g : N^{k+1} \to N$
and let $f = \mu[g]$.
And let the macropgramme $Q = (k,\ k+1,\ s)$ with s:

while $g(X_1,\ X_2,\ ...,\ X_k,\ X_{k+1}) \neq 0$ *do*

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

od;

$X_1 := X_{k+1}$

$F_Q = f \Rightarrow f \in F(WHILE)$

□

Chapter 11

Universality

Definition 11.0.1. Indexing functions
We denote an indexing of $\mathbf{F}$ to any surjective function $h : \mathbb{N} \to \mathbf{F}$
An index of a function $f \in \mathbf{F}$ under h is any natural number i such that $h(i) = f$

Remark. $\mathbf{F}$ is a countable set of functions.

11.1 Program codification

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.

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

Definition 11.1.1. Gödel numbering
Let $god : \mathbb{N}^* \to \mathbb{N}$ be a recursive and bijective function also known as gödelization.

We just need to know that this function exists and let us encode a vector of naturals $v \in \mathbb{N}^*$ into a natural number $n \in \mathbb{N}$.

Definition 11.1.2. Gödel's decoding
Let $degod : \mathbb{N}^2 \to \mathbb{N}$ be a recursive and total function also known as degödelization:

$$degod(z,i) = \begin{cases} 0 & if \quad z = 0 \ \vee \ k > l(z) \\ \\ x_i & if \quad god(x_1, x_2, \ldots, x_i, \ldots, x_{l(z)}) = z \end{cases}$$

where $l : \mathbb{N} \to \mathbb{N}$ is a recursive and total function defined as:

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

$l(z)$ refers to the length of the vector obtained from decoding z

11.1.1 Encoding of WHILE programs

Definition 11.1.3. Function for encoding $WHILE$ statement

$codi: \ COD\_WHILE \to \mathbb{N}$

$$codi(s) = \begin{cases} 5(i-1) & \text{if } s = X_i := 0 \\ 5\sigma_1^2(i-1, j-1) + 1 & \text{if } s = X_i := X_j \\ 5\sigma_1^2(i-1, j-1) + 2 & \text{if } s = X_i := X_j + 1 \\ 5\sigma_1^2(i-1, j-1) + 3 & \text{if } s = X_i := X_j - 1 \\ 5\sigma_1^2\big(i-1, Codi(s_1)\big) + 4 & \text{if } s = while\ X_i \neq 0\ do\ s_1\ od \end{cases}$$

Definition 11.1.4. Function for encoding $WHILE$ codes
Let $s \in COD\_WHILE$ with $s = s_1; s_2; \ldots; s_m$ where each s_i is a statement, then:

$$Codi(s) = god\big(codi(s_1), codi(s_2), \ldots, codi(s_m)\big) - 1$$

Definition 11.1.5. Function for encoding $WHILE$ programs
Let $Q \in WHILE$, with $Q = (n, p, s)$

CODI: $WHILE \to \mathbb{N}$
$\text{CODI}(Q) = \sigma_1^3\big(n, p - max\{n, k\}, Codi(s)\big)$

where $k = max\{m \in \Sigma_d^+ \mid X_m \prec s\}$

11.1.2 Decoding WHILE programs

Definition 11.1.6. Type of statement function
$type : \mathbb{N} \to \{0, 1, 2, 3, 4\}$
$type(z) = z \ mod \ 5$

Definition 11.1.7. Extract function
$extr : \mathbb{N} \times \{1, 2\} \to \mathbb{N}$

$$extr(z, i) = \begin{cases} 1 + z/5 & if \quad type(z) = 0 \\ \\ 1 + \sigma^1_{2,i}\big((z - type(z))/5\big) & if \quad type(z) \neq 0 \end{cases}$$

Definition 11.1.8. Function for decoding $WHILE$ statement

$decodi : \mathbb{N} \to COD\_WHILE$

$$decodi(z) = \begin{cases} X_i := 0 & if \quad type(z) = 0 \\ X_i := X_j & if \quad type(z) = 1 \\ X_i := X_j + 1 & if \quad type(z) = 2 \\ X_i := X_j - 1 & if \quad type(z) = 3 \\ while\ X_i \neq 0\ do\ DeCodi(j-1)\ od & if \quad type(z) = 4 \end{cases}$$

where $i = extr(z, 1),\ j = extr(z, 2)$

Definition 11.1.9. Function for decoding $WHILE$ codes
$DeCodi : \mathbb{N} \to COD\_WHILE$
$DeCodi(z) =$

$$decodi\big(degod(z+1,1)\big); decodi\big(degod(z+1,2)\big); \ldots; decodi\big(degod(z+1,l(z))\big)$$

Definition 11.1.10. Function for decoding $WHILE$ programs
$\mathrm{DECODI} : \mathbb{N} \to WHILE$
$\mathrm{DECODI}(z) = \big(\sigma^1_{3,1}(z), \sigma^1_{3,2}(z) + max\{\sigma^1_{3,1}(z), k\}, DeCodi(\sigma^1_{3,3}(z))\big)$

where $k = max\{m \in \Sigma^+_d \mid X_m \prec DeCodi(\sigma^1_{3,3}(z))\}$

Corollary 11.1.1. Indexing of REC
$h : \mathbb{N} \to F(WHILE)$
$h(z) = F_{DECODI(z)}$

Given the equivalence theorem, the function is also an indexation of REC.

Corollary 11.1.2. Indexing of REC^n
$h : \mathbb{N} \to F^n(WHILE)$
$h(z) = F_{(n,max\{n,k\},DeCodi(z))}$

where $k = max\{m \in \Sigma_d^+ \mid X_m \prec DeCodi(z)\}$

Given the equivalence theorem, the function is also an indexing of REC^n.

11.2 Universal function

Definition 11.2.1. Universal function
We say that the function $U[\mathbf{F}] : \mathbb{N}^{n+1} \to \mathbb{N}$ is universal for $\mathbf{F}$ iff:

$$\exists\, h : \mathbb{N} \to \mathbf{F} \text{ that is an indexing of } \mathbf{F} \mid\ U[\mathbf{F}](i, \underline{x}) = h(i)(\underline{x})\ \forall i \in \mathbb{N}\ \wedge\ \forall \underline{x} \in \mathbb{N}^n$$

Remark. $\mathbf{F}$ is a countable set of functions.
When we choose an indexing h of $U[\mathbf{F}]$, we say that $U[\mathbf{F}]$ is the universal function for $\mathbf{F}$ under indexing h .

Theorem 11.2.1. $U[REC^n] \in \ REC^{n+1}$

Let's assume that:

$$\exists\, U \in WHILE \mid F_U = U[REC^n] \Rightarrow$$

$$\vdots$$

$$U[REC^n] \in F^{n+1}(WHILE) \Rightarrow$$

$$\vdots$$

$$U[REC^n] \in \ REC^{n+1}$$

So we only need to prove that $\exists\, U \in WHILE \mid F_U = U[REC^n]$, that will be made in the next topic.

Remark. We will refer to U as the universal program.

11.3 Universal program

In order to increase the readability of the extended $WHILE$ program, instead of:

Name
Input: $\underline{x}$
Output: X_1
Code: s

We will write

Name($\underline{x}$)
    s

Remark. All programs written in this way has as output the variable X_1

Definition 11.3.1. Universal program (U)

$U(z, \underline{x})$

```
1 m := god(x);
2 m := Simul(z, m) ;
3 X_1 := degod(m, 1) ;
```

Definition 11.3.2. Simul program $Simul(z, m)$

```
1  z := z + 1;
2  while l(z) ≠ 0 do
3      s := degod(z, 1);
4      if type(s) ≤ 3 then
5          m := Execute(s, m);
6          z := Reduce(z);
7      else
8          if degod(m, extr(s, 1)) ≠ 0 then
9              z := Add(z, extr(s, 2))
10         else
11             z := Reduce(z)
12         fi
13     fi
14 od
15 X_1 := m
```

Definition 11.3.3. Replace function
$replace : \mathbb{N}^3 \rightarrow : \mathbb{N}$

$replace(z, k, x) =$

$$\begin{cases} god(degod(z,1), \ldots, degod(z, l(z)), 0, \ldots, 0, x) & if \quad k > l(z) \wedge l(z) > 0 \\ god(0, \ldots, 0, x) & if \quad k > l(z) \wedge l(z) = 0 \\ z & if \quad k = 0 \\ god(degod(z,1), \ldots, degod(z, k-1), \\ \quad x, degod(z, k+1), \ldots, degod(z, l(z))) & if \quad k \leq l(z) \wedge k \neq 0 \end{cases}$$

Remark. $replace(z, k, x)$ is the code of the vector obtained after replacing in the vector codified by z, the component k for the value of x .

Definition 11.3.4. Execute program

$Execute(s, m)$

---

1 **if** $type(s) = 0$ **then**
2 $\quad m := replace(m, extr(s, 1), 0);$
3 **fi**
4 **if** $type(s) = 1$ **then**
5 $\quad m := replace(m, extr(s, 1), degod(m, extr(s, 2)));$
6 **fi**
7 **if** $type(s) = 2$ **then**
8 $\quad m := replace(m, extr(s, 1), degod(m, extr(s, 2)) + 1);$
9 **fi**
10 **if** $type(s) = 3$ **then**
11 $\quad m := replace(m, extr(s, 1), degod(m, extr(s, 2)) - 1);$
12 **fi**
13 $X_1 := m$

Definition 11.3.5. Reduce program

Reduce(z)

```
1 if l(z) < 2 then
2     z := 0;
3 else
4     z := god(degod(z, 2), degod(z, 3), ..., degod(z, l(z)));
5 fi
6 X_1 := z
```

Definition 11.3.6. Add program

$Add(z, s)$

---

1 $X_1 := god(degod(s, 1), \ldots, degod(s, l(s)), degod(z, 1), \ldots, degod(z, l(z)));$

Remark. What is added to the top of z is not the code of the body of the loop but the sequence of instructions codified by s. This is why the degod function is applied to s before applying the function god to z.

Chapter 12

Theoretical limits of computing

12.1 Halting problem

Problem	**Predicate**	**Function**
solvable	decidable	$\in TREC$
partially solvable	enumerable	$\in REC$
unsolvable	undecidable	$\notin TREC$
totally unsolvable	not enumerable	$\notin REC$

Definition 12.1.1. H predicate
g is the number of a $WHILE$ program.
$\underline{x}$ is an input vector to g.
$H(g, \underline{x})$ is true iff the program g halts with the input $\underline{x}$.

Definition 12.1.2. H^n predicate
g is the number of a $WHILE$ program with n inputs.
$\underline{x}$ is an input vector to g ($\underline{x} \in N^n$).
$H^n(g, \underline{x})$ is true iff the program g halts with the input $\underline{x}$.

Theorem 12.1.1. H^1 is not solvable

Proof. H^1 is solvable $\Rightarrow$ H^1 is decidable $\Rightarrow$ $\exists H^1 \in TREC | H^1$ is the characteristic function of the H^1 predicate.

$$H^1(g,x) = \begin{cases} 1 & \text{if the program } g \text{ halts with the input } x \\ 0 & \text{if the program } g \text{ does not halt with the input } x \end{cases}$$

Consider the function:

$$bad(x) = \begin{cases} 1 & \text{if } H^1(x,x) = 0 \\ \uparrow & \text{if } H^1(x,x) = 1 \end{cases}$$

The function *bad* is calculated by:

Bad = (1, 2, c)

c:

```
1 X2 := X2 + 1;
2 while H^1(X1, X1) ≠ 0 do
3       X2 := 0;
4 od
5 X1:= X2
```

Consider b the Bad number:

• if $H^1(b,b) = 1 \underset{\substack{\uparrow \\ bad \text{ definition}}}{\Rightarrow} bad(b) = \uparrow \underset{\substack{\uparrow \\ H^1 \text{definition}}}{\Rightarrow} H^1(b,b) = 0 \Rightarrow\Leftarrow$

• if $H^1(b,b) = 0 \underset{\substack{\uparrow \\ bad \text{ definition}}}{\Rightarrow} bad(b) = 1 \underset{\substack{\uparrow \\ H^1 \text{ definition}}}{\Rightarrow} H^1(b,b) = 1 \Rightarrow\Leftarrow$

Proved by contradiction

Remark. This result is extensible to H^n and H. □

Proposition 12.1.1. H^1 is partially solvable

Proof. Consider:

$$f : N^2 \to N$$

$$f = \sigma\left(U[REC^1]\right)$$

$P_f = H^1 \underset{\substack{\uparrow \\ f \in REC}}{\Rightarrow} H^1 \in PRED(REC) \Rightarrow H^1$ is partially solvable.

Remark. This result is extensible to H^n and H. □

12.2 The busy beaver function

Definition 12.2.1. Length of a $WHILE$ program (long), P^n
$Q \in WHILE, Q = (n, p, s)$.

$$long : WHILE \to N$$

$$long(Q) = |s|_{do} + |s|_{:=}$$

P^n is the set of all $WHILE$ programs with length n:

$$P^n = \{Q \in WHILE \mid long(Q) = n\}$$

Remark. The length of a program is the number of sentences its code has, while the size is the number of lines of that code.

Definition 12.2.2. Busy beaver function(Σ)
$\Sigma : N \to N$
$\Sigma(n) = max\{Q(0) \mid Q \in WHILE^1 \wedge Q \in P^n \wedge Q(0) \in N\}$

Proposition 12.2.1. $\Sigma(n+1) > \Sigma(n) \qquad \forall n \in N$

Proof. Consider:
$Q = (1, p, s) \;\mid\; long(Q) = n \quad \wedge \quad Q(0) = \Sigma(n)$

Q' = (1, p, s')

s':

1 s;
2 $X_1 := X_1 + 1$;

$long(Q') = n + 1 \quad \wedge \quad Q'(0) = \Sigma(n) + 1 \;\Rightarrow\; \Sigma(n+1) \geq \Sigma(n) + 1 > \Sigma(n)$
$\Rightarrow \quad \Sigma(n+1) > \Sigma(n)$ □

Proposition 12.2.2. $\exists Q \in P^k \mid f = F_Q \;\Rightarrow\; \Sigma(n+k) \geq f(n) \quad \forall n \in N$

Proof. Consider:
$Q = (1, p, s) \;\mid\; long(Q) = k \;\wedge\; F_Q = f$

Q' = (1, p, s')

s':

$$\left.\begin{array}{l} X_1 := X_1 + 1; \\ \quad \cdots \\ X_1 := X_1 + 1; \end{array}\right\} n \text{ times}$$

s;

$long(Q') = n + k \;\wedge\; Q'(0) = f(n) \;\Rightarrow\; \Sigma(n+k) \geq f(n)$ □

Theorem 12.2.1. $\Sigma \notin F(WHILE)$

Proof. Consider:

$$\left.\begin{array}{r} \Sigma \in F(WHILE) \\ g(n) = 2n; \quad g \in F(WHILE) \\ f(n) = \Sigma(2n) \end{array}\right\} \quad \Rightarrow \; f \in F(WHILE) \; \Rightarrow$$

$$\exists Q \in P^k \;\; | F_Q = f \underset{\uparrow \text{prop. } 12.2.2}{\Rightarrow} \; \Sigma(n+k) \geq f(n) \;\; \forall n \in N \underset{\uparrow f(n)=\Sigma(2n)}{\Rightarrow}$$

$$\Sigma(n+k) \geq \Sigma(2n) \;\; \forall n \in N \underset{\uparrow \text{for } n=k+1}{\Rightarrow} \; \Sigma(2k+1) \geq \Sigma(2k+2) \Rightarrow\Leftarrow \underset{\uparrow \text{prop. } 12.2.1}{}$$

Proved by contradiction □

Proposition 12.2.3. $\Sigma \notin F(WHILE) \Rightarrow H^1$ is not solvable

Proof. Suppose $\Sigma \notin F(WHILE) \wedge H^1$ is solvable.
In this case we can choose only the programs that halt with the input equals zero for each length and choose the biggest output.
It implies: $\Sigma \in F(WHILE) \Rightarrow\Leftarrow$

Proved by contradiction □

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.