TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES

Francisco Vico

departamento
Lenguajes y
Ciencias de la Computación

área de conocimiento
Ciencias de la Computación e
Inteligencia Artificial

ETSI Informática
Universidad de Málaga

fjvico@uma.es
geb.uma.es/fjv

10 de diciembre de 2015

Foreword

Writing just another textbook in theory of automata, formal languages or computability looks like pointless: there are so many now, that we can hardly do better. However, every lecturer knows what is to be taught, and the lectures benefit from this particular nomenclature in a given subject, or that concrete demonstration. 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 this compilation: to become the textbook 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. In this sense, this manuscript has benefitted from the efforts of many people. Part of its content was extracted from previous documents, kindly offered to the open access community by their authors [Ruohonen, 2009; Jian et al, 2002]. 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* (that is the reason because numbering in some epigraphs is not always correlative). The selection and compilation of the manuscript has been performed by students taking the subject during the course 2015-16: Iustina Andronic (Erasmus student) and Esteban Delgado; and curated by Francisco Vico, professor of Computer Science and Artificial Intelligence at the University of Malaga, responsible for the content of this subject at the university's OpenCourseWare programme.

A package of software that implements the main concepts in this manuscript is also under development for the programming language Octave, and it has been made public under CC0 license at https://bitbucket.org/fjvico/umafol. This document itself, in its latest version can also be accessed at http://j.mp/talf_ocw.

This is just version 1.0, coming years will see it develop, enrich and accommodate new knowledge and experience.

Francisco Vico
Málaga, December 8, 2015

Notation

$\mathbb{N}$	the set of nonnegative integers (or natural numbers), i.e., {0, 1, 2, … } (U+2115)
$\mathbb{P}$	the set of positive integers (U+2119)
$\mathbb{R}$	the set of real numbers (U+211D)
$\mathbb{Z}$	the set of integers (U+2124)
$\varnothing$	the empty set (U+2205)
$\subseteq$	the (infix) subset relation between sets (U+2286)
$\subset$	the (infix) proper subset relation between sets (U+2282)
$\cup$	the infix union operation on sets (U+222A)
$\cap$	the infix intersection operation on sets (U+2229)
$\sim$	the prefix complementation operation on sets (U+007E)
$-$	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× … ×A (n times)
2^A	the powerset of A

1 Regular languages

Languages and grammars

Definition 2.1. An alphabet is a finite nonempty set of symbols. Symbols are assumed to be indivisible.

Definition 2.2. A string over an alphabet Σ is a finite sequence of symbols of Σ.

Definition 2.3. A special string which contains no symbols at all is the empty string, and it is represented by ε (sometimes λ or Λ).

Definition 2.4. The set of all strings over an alphabet Σ is denoted by Σ^* , and the set of all nonempty strings over Σ is denoted by Σ^+ . The empty set of strings is denoted by ∅.

Definition 2.5. The number of symbols in a string x is called its length, denoted by $|x|$. The length of ε is 0.

Definition 2.9. Let $x = a_1\, a_2 \ldots a_n$ and $y = b_1\, b_2 \ldots b_m$ be two strings. The concatenation of x and y, denoted by xy, is the string $a_1\, a_2 \ldots a_n b_1 b_2 \ldots b_m$.

Then for any string x, $\varepsilon x = x\varepsilon = x$. For any string x and integer $n \geq 0$, we use x^n to denote the string formed by sequentially concatenating n copies of x.

These are examples of word concatenation in the alphabet $\{a, b, c\}$:

$x = aacbba$,	$y = caac$,	$xy = aacbbacaac$
$x =$ aacbba,	$y = \varepsilon$,	$xy = x = aacbba$
$x = \varepsilon$,	$y = caac$,	$xy = y = caac$

Concatenation is associative, i.e.,

$$x(yz) = (xy)z$$

As a consequence of this, repeated concatenations can be written without parentheses. On the other hand, concatenation is usually not commutative, As a rule then

$$xy \neq yx,$$

but not always, and in the case of a unary alphabet concatenation is obviously commutative.

Definition 2.14. The n^{th} (concatenation) power of the word w is

$$x^0 = \varepsilon$$
$$x^n = xx^{n-1} = \underbrace{xx\ldots x}_{n \text{ copies}} .$$

Definition 2.9. Nearly every characterization problem is algorithmically decidable for regular languages. The most common ones are the following (where L or L_1 and L_2 are given regular languages):

- *Emptiness Problem*: Is the language L empty (i.e., does it equal $\varnothing$)?
  It is fairly easy to check for a given finite automaton recognizing L, whether or not there is a state transition chain from an initial state to a terminal state.

- *Inclusion Problem*: Is the language L_1 included in the language L_2?
  Clearly $L_1 \subseteq L_2$ if and only if $L_1 - L_2 = \varnothing$.

- *Equivalence Problem*: Is $L_1 = L_2$?
  Clearly $L_1 = L_2$ if and only if $L_1 \subseteq L_2$ and $L_2 \subseteq L_1$.

- *Finiteness Problem*: Is L a finite language?
  It is fairly easy to check for a given finite automaton recognizing L, whether or not it has arbitrarily long state transition chains from an initial state to a terminal state.

- *Membership Problem*: Is the given word w in the language L or not?
  Using a given finite automaton recognizing L it is easy to check whether or not it accepts the given input word w.

Definition 2.20. For any alphabet Σ, a language over Σ is a set of strings over Σ. The members of a language are also called the words of the language.

Definition 2.30. A grammar is a quadruple (Σ, V, S, P), where:

1. Σ is a finite nonempty set called the terminal alphabet. The elements of Σ are called the terminals.
2. V is a finite nonempty set disjoint from Σ. The elements of V are called the nonterminals or variables.
3. $S \in V$ is a distinguished nonterminal called the start symbol.
4. P is a finite set of productions (or rules) of the form

$$\alpha \rightarrow \beta$$

where $\alpha \in (\Sigma \cup V)^* V (\Sigma \cup V)^*$ and $\beta \in (\Sigma \cup V)^*$, i.e. α is a string of terminals and nonterminals containing at least one nonterminal and β is a string of terminals and nonterminals.

Definition 2.31 Let $G = (\Sigma, V, S, P)$ be a grammar, and let γ_1 , γ_2 be two sentential forms of G. We say that γ_1 directly derives γ_2 , written $\gamma_1 \Rightarrow \gamma_2$, if $\gamma_1 = \sigma\alpha\tau$, $\gamma_2 = \sigma\beta\tau$, and $\alpha \rightarrow \beta$ is a production in P .

Definition 2.35. Let γ_1 and γ_2 be two sentential forms of a grammar G. We say that γ_1 derives γ_2, written $\gamma_1 \Rightarrow^* \gamma_2$, if there exists a sequence of (zero or more) sentential forms $\sigma_1, \ldots, \sigma_n$ such that

$$\gamma_1 \Rightarrow \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \gamma_2.$$

The sequence $\gamma_1 \Rightarrow \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \gamma_2$ is called a derivation of γ_2 from γ_1.

Definition 2.37. Let $G = (\Sigma, V, S, P)$ be a grammar. A sentential form of G is any string of terminals and nonterminals, i.e. a string over $\Sigma \cup V$.

Definition 2.39. Let $G = (\Sigma, V, S, P)$ be a grammar. The language generated by G, denoted by $L(G)$, is defined as

$$L(G) = \{x | x \in \Sigma^*, S \Rightarrow^* x\}.$$

The words in $L(G)$ are also called the sentences of $L(G)$.

Let $G = (\Sigma, V, S, P)$ be a grammar.

Chomsky's Hierarchy - In Chomsky's hierarchy grammars are divided into four types:

- Type 0: No restrictions.
- Type 1: CS grammars.
- Type 2: CF grammars.
- Type 3: Linear grammars having productions of the form $X_i \rightarrow wX_j$ or $X_j \rightarrow w$ where X_i and X_j are nonterminals and $w \in \Sigma_T^*$, the so-called right-linear grammars.

Grammars of Types 1 and 2 generate the so-called CS-languages and CF-languages, respectively, the corresponding families of languages are denoted by CS and CF. Languages generated by Type 0 grammars are called computably enumerable languages (CE-languages), the corresponding family is denoted by CE.

Definition 2.58. G is also called a Type-0 grammar or an unrestricted grammar.

Definition 2.59. G is a Type-1 or context-sensitive grammar if each production $\alpha \rightarrow \beta$ in P satisfies $|\alpha| \leq |\beta|$. By "special dispensation," we also allow a Type-1 grammar to have the production $S \rightarrow \varepsilon$, provided S does not appear on the right-hand side of any production.

Definition 2.60. G is a Type-2 or context-free grammar if each production $\alpha \rightarrow \beta$ in P satisfies $|\alpha| = 1$; i.e., α is a single nonterminal.

Definition 2.61. G is a Type-3 or right-linear or regular grammar if each production has one of the following two forms:

$A \rightarrow cB$

$A \rightarrow c$

where A, B are nonterminals (with $B = A$ allowed) and c is a terminal.

Definition 2.78. Let L_1 and L_2 be two languages over Σ. The concatenation of L_1 and L_2, denoted by L_1L_2 , is the language $\{xy | x \in L_1 , y \in L_2 \}$.

Definition 2.82. Let L be a language over Σ. Define $L_0 = \{\varepsilon\}$ and $L^i = LL^{i-1}$ for $i \geq 1$. The Kleene closure of L, denoted by L^*, is the language

$$L^* = \bigcup_{i \geq 0} L^i .$$

The positive closure of L, denoted by L^+, is the language

$$L^+ = \bigcup_{i \geq 1} L^i .$$

In other words, the Kleene closure of a language L consists of all strings that can be formed by concatenating zero or more words from L. For example, if $L = \{0, 01\}$, then $LL = \{00, 001, 010, 0101\}$, and L^* comprises all binary strings in which every 1 is preceded by a 0. Note that concatenating zero words always gives the empty string, and that a string with no 1s in it still makes the condition on "every 1" true. L + has the meaning "concatenate one or more words from L," and satisfies the properties $L^* = L^+ \cup \{\varepsilon\}$ and $L^+ = LL^*$. Furthermore, for any language L, L^* always contains , and L + contains if and only if L does. Also note that Σ^* is in fact the Kleene closure of the alphabet Σ when Σ is viewed as a language of words of length 1, and Σ^+ is just the positive closure of Σ.

Closure of the types of languages

Closure properties are often useful in constructing new languages from existing languages, and for proving many theoretical properties of languages and grammars. The closure properties of the four types of languages in the Chomsky hierarchy are summarized below. Proofs may be found in [Harrison, 1978], [Hopcroft and Ullman, 1979], or [Gurari, 1989];

the closure of the CSLs under complementation is the famous Immerman-Szelepcsényi theorem.

Theorem.

1. The class of Type-0 languages is closed under union, intersection, concatenation, and Kleene closure, but not under complementation.
2. The class of context-free languages is closed under union, concatenation and Kleene closure, but not under intersection or complementation.
3. The classes of context-sensitive and regular languages are closed under all of the five operations.

For example, let $L_1 = \{0^m1^n2^p | m = n\}$, $L_2 = \{0^m1^n2^p | n = p\}$, and $L_3 = \{0^m1^n2^p | m = n \text{ or } n = p\}$. Now L_1 is the concatenation of the context-free languages $\{0^n1^n | n \geq 0\}$ and 2^* , so L^1 is context-free. Similarly L^2 is context-free. Since $L^3 = L^1 \cup L^2$, L^3 is context-free. However, intersecting L^1 with L^2 gives the language $\{\{0^m1^n2^p | m = n = p\}$, which is not context-free.

Definition 2.11. A class of languages is said to be closed under a particular operation (such as union, intersection, complementation, concatenation, or Kleene closure) if every application of the operation on language(s) of the class yields a language of the class.

Regular expressions

Definition 3.1. The regular expressions over an alphabet Σ and the languages they represent are defined inductively as follows.

1. The symbol ∅ is a regular expression, and represents the empty language.
2. The symbol ε is a regular expression, and represents the language whose only member is the empty string, namely {ε}.
3. For each $c \in \Sigma$, c is a regular expression, and represents the language $\{c\}$, whose only member is the string consisting of the single character c.
4. If r and s are regular expressions representing the languages R and S, then $(r + s)$, (rs) and (r^*) are regular expressions that represent the languages $R \cup S$, RS, and R^*, respectively.

Finite-state automata

Automata are used to recognize words of a language. An automaton then ”processes” a word and, after finishing the processing, ”decides” whether or not the word is the language.

An automaton is finite if it has a finite memory, i.e., the automaton may be thought to be in one of its (finitely many) (memory)states. A finite deterministic automaton is defined formally by giving its states, input symbols (the alphabet), the initial state, rules for the state transition, and the criteria for accepting the input word.

Definition 4.1. A finite (deterministic) automaton (DFA) is a quintuple M = (Q, Σ, q_0 , δ, A) where

- $Q = \{q_0, q_1, \ldots, q_m\}$ is a finite set of states, the elements of which are called states;
- Σ is the set input symbols (the alphabet of the language);
- q_0 is the initial state ($q_0 \in Q$);
- δ is the (state) transition function which maps each pair (q_i, a), where q_i is a state, and a is an input symbol, to exactly one next state q_j : $\delta(q_i, a) = q_j$;
- A is the so-called set of terminal states ($A \subseteq Q$).

As its input the automaton M receives a word

$$w = a_1 \ldots a_n$$

which it starts to read from the left. In the beginning M is in its initial state q_0 reading the first symbol a 1 of w. The next state q_j is then determined by the transition function:

$$q_j = \delta(q_0 , a_1).$$

In general, if M is in state q_j reading the symbol a_i, its next state is $\delta(q_j, a_i)$ and it moves on to read the next input symbol a_{i+1}, if any. If the final state of M after the last input symbol a n is read is one of the terminal states (a state in A), then M accepts w, otherwise it rejects w. In particular, M accepts the empty input ε if the initial state q_0 is also a terminal state.

The language recognized by an automaton M is the set of the words accepted by the automaton, denoted by $L(M)$.

Any word $w = a_1 \ldots a_n$, be it an input or not, determines a so-called state transition chain of the automaton M from a state q_{j0} to a state q_{jn} :

$$q_{j0} , q_{j1} , \ldots , q_{jn} ,$$

where always $q_{ji+1} = \delta(q_{ji}, a_{i+1})$.

Any finite automaton can be represented graphically as a so-called state diagram. A state is then represented by a circle enclosing the symbol of the state, and in particular a terminal state is represented by a double circle:

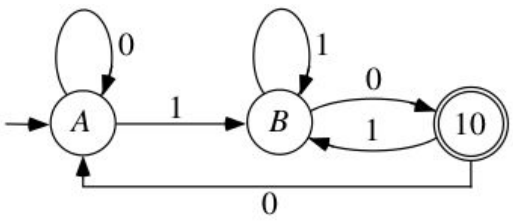
A state transition $\delta(q_i, a) = q_j$ is represented by an arrow labelled by a, and in particular the initial state is indicated by an incoming arrow:

Such a representation is in fact an edge-labelled directed graph.

Example. The automaton $\{A, B, 10\}$, $\{0, 1\}$, A, δ, $\{10\}$ where δ is given by the state transition table

δ	0	1
A	A	B
B	10	B
10	A	B

is represented by the state transition diagram



The language recognized by the automaton is the regular language $(0+1)^*10$.

Definition 4.21. Defined formally a nondeterministic finite automaton (NFA) is a quintuple $M = (Q, \Sigma, S, \delta, A)$ where:

- Q, Σ and A are as for the deterministic finite automaton;
- S is the set of initial states;
- δ is the (state) transition function which maps each pair (q_i, a), where qi is a state and a is an input symbol, to exactly one subset T of the state set Q: $\delta(q_i , a) = T$.

Definition 4.34. *M* accepts a word w if there is at least one terminal state in the set of states $\delta^*(S, w)$. Λ is accepted if there is at least one terminal state in *S*. The set of exactly all words accepted by *M* is the language *L(M)* recognized by *M*.

Definition 4.36. The nondeterministic finite automaton may be thought of as a generalization of the deterministic finite automaton, obtained by identifying in the latter each state q_i by the corresponding singleton set $\{q_i\}$. It is however no more powerful in recognition ability.

Proof. Consider a language L recognized by the nondeterministic finite automaton $M = (Q, \Sigma, S, \delta, A)$. The equivalent deterministic finite automaton is then $M_1 = (Q_1, \Sigma, q_0, \delta_1, A_1)$ where

$$Q_1 = 2^Q \, , \; q_0 = S \, , \; \delta_1 = \hat{\delta} \, ,$$

and A_1 consists of exactly all sets of states having a nonempty intersection with *A*. The states of M_1 are thus all sets of states of *M*. We clearly have $\delta_1^*(q_0, w) = \hat{\delta}^*(S, w)$, so *M* and M_1 accept exactly the same words, and M_1 recognizes the language *L*.

Regularity conditions

Definition 5.1. The language L separates the words w and v if there exists a word u such that one of the words wu and vu is in L and the other one is not. If L does not separate the words w and v, then the words wu and vu are always either both in L or both in L, depending on u. There is a connection between the separation power of a language recognized by a finite automaton and the structure of the automaton:

Theorem. If the finite automaton $M = (Q, \Sigma, q_0, \delta, A)$ recognizes the language L, and for the words w and v it verifies $\delta^*(q_0, w) = \delta^*(q_0, v)$, then L does not separate w and v.

Proof. As is easily seen, in general

$$\delta^*(q_i, xy) = \delta^*(\delta^*(q_i, x), y).$$

So

$$\delta^*(q_0, wu) = \delta^*(\delta^*(q_0, w), u) = \delta^*(\delta^*(q_0, v), u) = \delta^*(q_0, vu)$$

Definition 5.6. If the language L can be recognized by a finite automaton with n states, $x \in L$ *and* $|x| \geq n$, then x can be written in the form $x = uvw$ where $|uv| \leq n$, $v \neq \Lambda$ and the "pumped" words $uv^m w$ are all in L.

Myhill–Nerode Theorem - A language is regular if and only if it has a finite index.

If a regular language L is defined by a deterministic finite automaton $M = (Q, \Sigma, q_0, \delta, A)$ recognizing it, then the minimization naturally starts from M. The first step is to remove all idle states of M, i.e., states that cannot be reached from the initial state. After this we may assume that all states of M can expressed as $\delta^*(q_0, w)$ for some word w. For the minimization the states of M are partitioned into M-equivalence classes as follows. The states q_i and q_j are not M-equivalent if there is a word u such that one of the states $\delta^*(q_i, u)$ and $\delta^*(q_j, u)$ is terminal and the other one is not, denoted by $q_i \equiv M\, q_j$. If there is no such word u, then q_i and q_j are M-equivalent, denoted by $q_i \equiv_M q_j$. We may obviously assume $q_i \equiv_M q_i$. Furthermore, if $q_i \equiv_M q_j$, then also $q_j \equiv_M q_i$, and if $q_i \equiv_M q_j$ and $q_j \equiv_M q_k$ it follows that $q_i \equiv_M q_k$. Each equivalence class consists of mutually M-equivalent states, and the classes are disjoint. (Cf. the L-equivalence classes and the equivalence relation $\equiv L$.) Let us denote the M-equivalence class represented by the state q_i by $\{q_i\}$. Note that it does not matter which of the M-equivalent states is chosen as the representative of the class. Let us then denote the set of all M-equivalence classes by Q.

M-equivalence and L-equivalence are related since $\langle \delta^*(q_0, w) \rangle = \langle \delta^*(q_0, v) \rangle$ if and only if

$[w] = [v]$. Because now all states can be reached from the initial state, there are as many M-equivalence classes as there are L-equivalence classes, i.e., the number given by the index of L. Moreover, M-equivalence classes and L-equivalence classes are in a one-to-one correspondence:

$$\langle \delta^*(q_0, w) \rangle \fallingdotseq [w],$$

in particular $\langle q_0 \rangle \fallingdotseq [\Lambda]$.