Automata and formal languages

Dominique Perrin
Université de Marne-la-Vallée

May 2, 2003

Abstract

This article provides an introduction to the theory of automata and formal languages. The elements are presented in a historical perspective and the links with other areas are underlined. In particular, applications of the field to linguistics, software design, text processing, computational algebra or computational biology are given.

Contents

1 Introduction **2**

2 Finite automata **3**
2.1 The origins . . . 3
2.2 The star-height . . . 4
2.3 Decomposition theorems . . . 5
2.4 Semigroups and varieties . . . 5
2.5 Automata and logic . . . 6
2.6 Infinite words and trees . . . 7
2.7 Symbolic dynamics . . . 7
2.8 Automata and groups . . . 8

3 Context-free grammars **8**
3.1 The origins . . . 8
3.2 Context-free languages . . . 8
3.3 Pushdown automata . . . 9
3.4 Context-free groups . . . 9
3.5 Rewriting systems . . . 10
3.6 An algebraic theory . . . 10

4 Computability **10**
4.1 Turing machines . . . 10
4.2 Recursiveness . . . 11
4.3 Complexity classes . . . 11
4.4 Quantum computing . . . 12

4.5 Circuit complexity 13

5 Formal series **14**
5.1 Rational series 14
5.2 Algebraic series 14

6 Combinatorics on words **15**
6.1 Sturmian words 15
6.2 Equations in words 16

7 Applications **16**
7.1 Compilers 16
7.2 Text processing 17
7.3 Natural languages 17
7.4 Genomes 17

8 Notes on the bibliography **18**

1 Introduction

The theory of automata and formal languages deals with the description of properties of sequences of symbols. Many circumstances may lead to such a sequence, from a discrete time process to a sequence of molecules. This explains the really interdisciplinary interest for this subject as I will briefly show. The mathematical theory grew with the need of formalizing and describing the processes linked with the use of computers and communication devices but its origins are strictly within mathematics.

Indeed, the early roots of formal language and automata theory can be found in the work of logicians at the beginning of the XXth century with outstanding figures such as Emil Post, Alonzo Church or Alan Turing. These developments where motivated by the search for the foundations of the notion of proof in mathematics, as initiated by the work of Hilbert. After the last world war, the development computers and telecommunications, as well as the interest for exploring the functions present in the human brain extended these early approaches with the work of Claude Shannon, Stephen Kleene or John von Neumann. In the sixties, the interest of the linguistic community was stimulated by the work of Noam Chomsky who advocated the use of formal methods to describe natural languages. Later, the development of molecular biology lead to consider the sequences of molecules formed by genomes as sequences of symbols on the alphabet formed by the basic elements, leading to another interest in describing properties like repetitions of occurrences or similarity between sequences.

The classical Chomsky hierarchy introduces four levels of increasing complexity in the description of formal languages: finite-state, context-free context-sensitive and recursively computable. I will try here to give an idea of each of these levels turn by turn, covering the essential definitions in a rather informal way and emphasizing the recent developments.

In the first section, the theory of finite automata is explored. The links of this theory with several other mathematical fields is underlined. The theory of varieties of semigroups and of rational languages is given a particular emphasis. The link with logic, and the extension of languages to infinite objects is also presented. Symbolic dynamics is also introduced with the notion of a symbolic dynamical system.

The second section deals with the second stage Chomsky's hierarchy, namely context-free languages. The link between context-free grammars and pushdown automata is underlined, together with recent results on deterministic pushdown automata. As in the case of finite automata, the link with group theory is mentioned, including an important result on context-free groups.

The last stage of the hierarchy, the class of recursively enumerable languages is introduced through Turing machines. The corresponding class of integer recursive functions is also defined. The complexity classes P, NP and $PSPACE$ are presented with the mention of the famous $P \neq NP$ open problem.

Two additional fields of research close to formal languages are considered. The first one is the field of formal series which has important applications to enumerative combinatorics and to the analysis of algorithms. The second one is combinatorics on words, one of the active branches of combinatorics with links to several fields in algebra and number theory.

The applications of formal languages and automata to a number of fields are finally presented. The areas of applications include software design with the conception of compilers, text processing including text compression, natural language processing and finally genome analysis.

The article ends with a commented bibliography which is intended to help the interested reader to learn more about some specific aspects of this field. If this article has driven the reader to such project, it will have reached its goal.

2 Finite automata

2.1 The origins

The idea of a finite automaton appears explicitly in the work of S. Kleene [24] in the fifties. It is the simplest model of computation and more generally of behavior of a system. It is just a discrete control equipped with a finite memory. The first theorem on this definition is the equivalence between the model of finite automata and the description of sequences of symbols using the three logical primitives of

- set union
- set product
- iteration

The expressions constructed in this way are usually called rational expressions and a language described by a rational expression is also called rational. Kleene's

theorem says that a language is rational if and only if it can be recognized by a finite automaton. This result (known as Kleene's theorem) has a constructive proof leading to algorithms implemented in numerous practical systems (see Section 7). The origins of Kleene's article are actually in a paper written by W. McCulloch and W. Pitts [34] which was an attempt to describe the behavior of the human brain through a model of neurons and synapses (*A logical calculus of ideas immanent in nervous activity*). The early papers on finite automata also have a link with the theory of circuits. A sequential circuit, in which the output depends on an input signal is indeed appropriately modeled by a finite automaton.

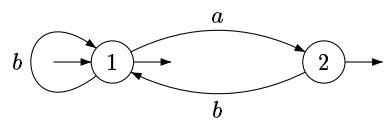

Figure 1: A finite automaton.

As an example, Figure 1 represents a finite automaton. It has two *states* called 1 and 2. State 1 is initial and both states 1 and 2 are final. Its *edges* are labeled by the symbols a and b. The set of words labeling paths from the initial state to a final state is the set of all words on a, b in which an a is always immediately followed by a b. According to Kleene's theorem, this set can also be described by a rational expression, namely,

$$(ab + b)^*(\epsilon + a).$$

where ϵ denotes the empty word, $+$ denotes the union and * the iteration.

2.2 The star-height

The star-height of a rational language X is easy to define. It is the minimal number (over all possible regular expressions describing X) of nested stars in the expression. Thus, the star-height is 0 if and only if the language is finite (i.e. there is no star at all in the expression). The star-height of the expression describing a language is not always the minimal possible. For example, the expressions $(a^*b)^*a^*$ and $(a + b)^*$ both describe the set of all words on a, b. The first one has star-height 2 but the second only 1. The problem of computing the star-height of a given rational language has been raised since the beginnings of automata theory. It was solved in principle by K. Hashiguchi in [22] who showed that the star-height is recursively computable. There remains to study efficient ways to compute or to show that they don't exist. A closely related problem is still unsolved: what is the minimal value of the star-height of extended rational expressions, namely those allowing the additional use of the complementation? No example could yet be proved to have extended star-height more than 1.

2.3 Decomposition theorems

Two finite automata may be composed to form another one. This is easily seen when the automata have an output since then they define functions on words which may be composed in the ordinary way.

A striking result, was obtained by Krohn and Rhodes in the years 1960 [26]. This decomposition theorem states that any finite automaton can be obtained by a composition of automata of two sorts:

1. group automata, in which the actions of the symbols on the states is one-to-one.

2. reset automata, in which the automaton just keeps the memory of the last symbol read.

The result actually applies to finite semigroups and gives an algebraic decomposition theorem for semigroups. Its counterpart on finite groups is the well-known decomposition of a finite group into simple groups corresponding for example to the principal descending series. An unsolved problem is the computation of the complexity of a finite semigroup, in the sense that it is the minimal number of groups appearing in a decomposition.

2.4 Semigroups and varieties

It was soon recognized that finite automata are closely linked with finite semigroups, thus giving an algebraic counterpart of the definition of recognizability by finite automata. Actually, one may characterize the rational languages as those which are recognized by a morphism on a finite semigroup, i.e. of the form $X = \varphi^{-1}(Y)$ for some morphism $\varphi : A^* \to S$ on a finite semigroup S and $Y \subset S$. There is also a minimal possible semigroup S for each X, called the *syntactic semigroup* of X.

The first important result using this connexion is Schützenberger's theorem on star-free languages. Let us give a few definitions. A star-free language is one that can be obtained with a restricted use of the rational operations, namely without the star but allowing all boolean operations including complement. For example, the set of all strings with alternating 0 and 1's is a star-free language since it can be written as the complement of the set of strings with some block 00 or 11. On the other hand, a finite semigroup S is called aperiodic if there is an integer $n \geq 1$ such that for any $x \in S$, one has $x^{n+1} = x^n$.

Schützenberger's theorem says that a rational language is star-free iff it can be recognized by an aperiodic semigroup.

The theory of varieties of rational languages was invented by Samuel Eilenberg in his book [13, 14], which appeared in the years 1970. It is in some sense the culminating point of many earlier efforts since it puts in a unified framework many of the previous known facts. Formally, a variety of semigroups is a family of finite semigroups closed by morphism, direct product of two elements and by taking subsemigroups. Technically, it should be called a pseudo-variety since it

is not closed by arbitrary direct product as in the theory of Birkhoff. The main examples are the variety of aperiodic semigroups (see above) and several of its subvarieties. A general theorem shows that there is a correspondence between varieties of semigroups and families of rational languages also called varieties of languages, in the sense that the semigoups are the syntactic semigroups of the corresponding languages. The variety of aperiodic semigroups thus corresponds to the variety of star-free languages.

An interesting notion is that of the localization of a variety V. A semigroup S is said to belong locally to the variety V if for every idempotent $e \in S$, the semigroup eSe is in V. The family of these semigroups can be shown to form again a variety denoted L_V. For example, the variety of aperiodic semigroups coincides with the variety of semigroups which are locally trivial.

An interesting other example is the variety of locally testable languages. A language X is called locally testable if there is an integer k such that the property $w \in X$ only depends on the set of blocks of length k in w. On the other hand, a semigroup is idempotent and commutative if $x = x^2$ and $xy = yx$ identically. By a deep theorem due McNaughton, Bzrozowski et al., it can be shown that a language is locally testable iff its syntactic monoid is locally idempotent (see [14]).

2.5 Automata and logic

It was Richard Büchi's idea to use finite automata in a context where they do not appear at first sight. The context is formal logic. It was known since the work of Gödel in the years 1930 that the logical theory of integers with the operations $+$ and $\times$ is undecidable. This left open the search for decidable subtheories. A first result was obtained by Pressburger who proved that the theory of integers with $+$ as unique operation is decidable. Büchi obtained in the years 1960 the decidability of another fragment: the monadic second order theory of the integers with the successor. The proof relies on a reduction to finite automata through the idea of considering a set of integers as a binary infinite sequence. For instance, the set of even integers corresponds to the sequence $1010101\cdots$.

Actually, the work of Büchi opened a new field in two directions. One is the extension of the theory of finite automata to infinite words (see below). Another one is the study of the interconnexions between automata theory and formal logic. In this framework, Büchi showed that rational languages are exactly those which can be defined in a logical language allowing to compare positions of letters in words and use quantifiers over sets of positions (monadic second order theory of the integers with $<$). This opened the path to characterizations in terms of automata of other classes of logical theories.

One of the original motivations to study star-free languages was the fact observed by McNaughton that they correspond to the first-order part (i.e. without set variables) of the above theory. More recently, a variant of this logic called temporal logic was introduced aiming at applications in program verification.

2.6 Infinite words and trees

The work of Büchi has produced a theory of languages, sometimes called ω-languages where the elements are infinite words instead of finite ones. All notions known in the ordinary case translate to the case of infinite words, even if the proofs are substantially more difficult. For example, the basic facts concerning rational languages as their closure under all boolean operations become, in the infinite case a delicate result due to R. Büchi, whose proof uses Ramsey theorem. In the same way, the basic determinization algorithm of finite automata becomes in the infinite case a difficult theorem of R. McNaughton. A survey on automata and infinite words can be found in [37].

Beyond words, automata on trees and possibly infinite trees can also be defined. The idea is simply that processing a tree top-down consists in duplicating copies of the automaton at each son of the current vertex. The state reached at the leaves (or the infinitely repeated states if the tree is infinite) determine whether or not the automaton accepts. The most famous result in this field is Rabin's complementation theorem. Its difficult proof has now been better understood thanks to a series of ideas due initially to Gurevitch and Harrington. The key point is the use of winning strategies in combinatorial games. The idea is the following: one can describe a run of an automaton on a tree by a game between two players. One of them (called *Automaton*) chooses a transition of the automaton, the other one (called *Pathfinder*) chooses a direction in the tree. In this way, player I has a winning strategy iff the automaton accepts the tree.

2.7 Symbolic dynamics

The field of finite automata has many connexions with a theory originally developed by Morse and Hedlund under the name of *symbolic dynamics*. This theory studies symbolic dynamical systems which are sets of two-sided infinite words chosen to be invariant under the shift and topologically closed. The simplest example is the set of binary sequences on $\{a, b\}$ without two consecutive b, called the golden mean system. Of particular interest are the *subshifts of finite type* introduced by Williams. They are those which can be defined by the interdiction of a finite number of words. Thus, the golden mean system is a subshift of finite type whereas the set of square-free words is not. The *entropy* $h(S)$ of a subshift S is the limit of $\frac{1}{n}\log u_n$ where u_n is the number of possible blocks of length n in S. For the golden mean system, one has $u_{n+1} = u_n + u_{n-1}$ and thus the entropy is $\log\varphi$ where φ is the golden mean, whence the name of the system. One of the beautiful results of the theory is Krieger's theorem saying that one can isomorphically embed strictly a subshift of finite type S into another one T if and only if

1. $h(S) < h(T)$
2. $p_n(S) \leq p_n(T)$ for all $n \geq 1$

where $p_n(S)$ is the number of points of period n in S. Subshifts of finite type have a close connexion with *circular codes* studied in formal language theory

(see [5]).

2.8 Automata and groups

The theory of groups has a subfield called *computational group theory* dealing with practical computations on groups, especially on groups presented by generators and relations. Many of the classical algorithms on groups, such as the Todd-Coxeter algorithm for coset enumeration can be viewed as algorithms on automata. Actually, a permutation representation of a finite group is a particular case of a finite automaton (see [52] for a presentation of this point of view).

The theory of automatic groups arose from the work of several people including geometers like W. Thurston and computational group theorists like J. Cannon. A group is said to be automatic if can be represented as a set R of words on A in such a way that for each $a \in A$, the relation (r, s) on R defined by $\overline{ra} = s$ is synchronous rational. Here, we denote by $\bar{x}$ the element of R corresponding to x. A relation between words is synchronous rational if it is a rational subset of $(A \times A)^*$, (up to a right padding making the words in a pair of equal length). The basic results of this theory are that automatic groups have nice properties (such as an easily solvable word problem) and that interesting groups, such as the discrete groups of hyperbolic geometry, or hyperbolic groups, are automatic (see [15] for an introduction).

3 Context-free grammars

3.1 The origins

The idea of a context-free grammar is technically simple. It is just a set of rewriting rules used to form set of words on a given alphabet. It appears explicitly in the work of Noam Chomsky in [8] aiming at a model for the syntax of natural languages. The grammars are called *phrase structure* grammars and are based upon ideas in syntax analysis introduced by Z. Harris in the years 1940 [21]. There are also earlier references to similar ideas in the work of logicians. Notably, the work of Turing [56] of A. Thue and Post [38] on formal systems. In parallel and seemingly independently, the same concept was developed by the early inventors of programming languages. In particular, Backus used the form now called BNF (for Backus-Naur Form) to describe the syntax of ALGOL. For a survey on the history of context-free grammars, see [18].

3.2 Context-free languages

A context-free grammar is a set of rules of the form $x \to w$ where x is a non-terminal symbol (or variable) and w is a word on the mixed alphabet of terminal and non-terminal symbols. A derivation consists in substituting a number of times a variable by application of a rule. The *language generated* by

the grammar is the set of words on the terminal alphabet which can be derived from an initial symbol called the axiom.

For example the grammar with one variable σ and the two terminal symbols $+, v$ with the two rules $\sigma \to +vv$ and $\sigma \to v$ generates all the additive expressions in prefix Polish notation with v as an operand symbol. The language generated is often called the Lukasiewicz language.

A language is context-free iff it can be generated by some context-free grammar. It is easy to show that context-free languages are closed by a number of operations (including intersection with a rational language) but that they are not closed under complement.

3.3 Pushdown automata

The idea of a context-free language is closely linked to the notion of a *pushdown automaton*. Such an automaton is a non-deterministic machine with a memory which may be of unbounded size but accessible only through a restricted mode called a stack. It consists in leaving access only to the last element in a first-in/last-out mode. A word is said to be accepted by such an automaton if there is a computation which leads to empty the stack after reading the input. An elementary result states that a language is context-free iff it can be accepted by some pushdown automaton.

The equivalence of general (non deterministic) pushdown automata is an undecidable problem since it is clearly equivalent to the equivalence of context-free grammars. A problem which has been open a long time is the decidability of the equivalence of deterministic pushdown automata. It was solved positively by G. Sénizergues [49] (see also [50] for a simplified proof).

3.4 Context-free groups

A fundamental example of context-free language is the Dyck language, named after a group theorist predating context-free grammars. It is the language of well-formed expressions using n types of parenthesis. It is generated by the grammar with rules $S \to a_n S \bar{a_n}$ for $n = 1, \ldots, n$. A more symmetric version also uses all rules $S \to \bar{a_n} S a_n$. It is actually the set of words on the alphabet $A_n \cup \bar{A}_n$ equivalent to the neutral element in the free group on the set A_n.

More generally, a group is said to be context-free if it admits a presentation as $G =< A|R >$ such that the set $L(G)$ of words on $A \cup \bar{A}$ which cancel modulo the relations from R is a context-free language. A free group is context-free since the Dyck language is context-free. A finite group is also context-free since for any presentation, the language $L(G)$ is rational. A remarkable theorem of Muller and Schupp states that a group is context-free iff it is free by finite, i.e. an extension of a free group by a finite group.

3.5 Rewriting systems

Many computational systems, in particular those dealing with groups, make use of rewriting rules to obtain the reduced form of the elements. For example, the process of reducing a word on $a_i, \bar{a}_i$ to the corresponding reduced form in the free group on the symbols a_i can be done by applying the rewriting rules $a_u\bar{a}_i \to \epsilon$, $\bar{a}_i a_i \to \epsilon$. The uniqueness of the reduced word expresses the property of the rewriting system to be confluent, i.e. giving a result independent of the way the rules have been applied. This would not be the case if the system were made of the rules $aa \to \epsilon$, $bb \to \epsilon$, $ab \to c$. Indeed, from aab, one may derive either b or ac. A pair like (b, ac) formed of two irreducible words derived form the same word is called a *critical pair*.

An algorithm allows one to complete a rewriting system to transform it in a confluent one without altering the underlying equivalence relation. This algorithm, due to Knuth and Bendix looks for critical pairs, and adds them (with a suitable orientation) to the reduction rules (see [7] for example). In the above example, one would add the pairs $ac \to b$ and $cb \to a$, obtaining a confluent system (for a group which is actually context-free).

3.6 An algebraic theory

It is possible to give a completely algebraic definition of context free languages, based upon the the idea that grammars can be seen as system of equations. The characterization uses the left quotient of a language X by a word u defined as $u^{-1}X = \{v \mid uv \in X\}$. Such sets can be used to characterize rational languages. Indeed, a language is rational iff the set of its left quotients is finite. A family $\mathcal{F}$ of subsets of A^* is called *stable* if $u^{-1}X \in \mathcal{F}$ for any $u \in A^*$ and $X \in \mathcal{F}$. One may characterize context-free languages on A as the elements of some finitely generated stable subalgebra of the algebra of subsets of A^* [4]. For example, the Lukasiewicz language L over the alphabet $\{a, b\}$ satisfies the equation $L = b + aLL$. Thus the sets $a^{-1}L = L^2$ and $b^{-1}L = \epsilon$ belong to the algebra generated by L.

4 Computability

Concerning the larger class containing all languages that can be recognized by some machine, there are at least two approaches, one by a machine model (Turing machines), the other one by a class of functions (recursive functions). Actually, this feature was also present with rational and with context-free languages. In the first case, we had finite automata versus rational expressions, in the second one pushdown automata versus grammars.

4.1 Turing machines

A Turing machine works with an infinite memory (actually a word on fixed alphabet, called the tape is enough) in which it can both read *and* write. It has

additionally a finite set of states and it is said to recognize the input word w if after starting with w on its tape, it stops in some final state. It is important to realize that it might never stop (this is familiar to programmers who have all written a program entering an infinite loop).

A language L is said to be recursively enumerable if it can be recognized by some Turing machine M. It is called recursively computable (or simply computable, or decidable) if it recursively enumerable so that its complement. Equivalently, L is computable if it can be recognized by a Turing machine which always halts. A typical undecidable language is the set $(\langle M \rangle, x)$ of pairs of a Turing machine (suitably coded by a word) and a word x such that M halts on input x.

4.2 Recursiveness

Recursive functions can be defined as functions on words but it is simpler to define them on integers as numeric functions $f(x_1, \cdots, x_p)$ with p integer variables and integer value. At this level, it does not matter so much since integers can be encoded by words and conversely. The class of recursive functions is formed by the functions which can be obtained from the initial functions (projection, successor and constant) by the following operations:

1. composition
2. primitive recursion
3. minimization

The primitive recursion allows one to build a function f from functions g, h by $f(0, x) = g(x)$ and $f(k+1, x) = h(f(k, n), k, n)$.

The minimization allows one to build from a function g a function f such that $f(x)$ is the minimal integer m such that $g(x, m) = 0$.

It is a classical result that recursive functions and Turing machines, and several other formalisms, define the same classes of computable objects. It is a philosophical claim (referred to as Church thesis) that they actually correspond to what one means by computable in some sense. Of course, this notion of computability does not take into account the time or space needed by a computation and in this sense it is not realistic. In the next section, we consider the problem of the amount of resources needed.

4.3 Complexity classes

Inside the class of recursive languages, natural subclasses appear which depend on the amount of resources needed for the recognition of a word of size n. The limitation can either bear on the *time* or the *space* used by the Turing machine. An important class is the class P of polynomial languages (or algorithms) which limits the computation time of a deterministic Turing machine by a polynomial function. An other important class is the class NP which is the same as P by

allowing non-deterministic Turing machines. Thus a language is in *NP* if there is a search in a binary tree of polynomial height producing the solution. A typical problem in the class *NP* is the satisfiability of boolean formulas. These formulas, like

$$(x \vee \neg y) \wedge (\neg x \vee z) \wedge \ldots$$

use boolean connectives and negation with variables $x, y, \ldots$. For each choice of values $\{true, false\}$ for the variables, it is easy to check whether the formula is true or false. Thus the problem of finding a set of values for which the formula is valid is in *NP*. It can even be shown to be *NP*-complete, in the sense that any problem in *NP* can be reduced to this one in polynomial time. Obviously $P \subset NP$ and it seems reasonable to guess that $P \neq NP$ but this has not been proved yet, and is considered as one of the central problems in the theory of computing. To mention a class defined by a restriction on the space used, the class *PSPACE* is defined as the class of languages which can be recognized by a Turing machine working in space of size bounded by a polynomial in the length of the input. One has $NP \subset PSPACE$. A typical problem in *PSPACE* is the satisfiability of quantified boolean formulas. Such a formula has the form of a boolean formula using quantifiers like

$$\forall x \forall y (x \wedge \neg y \vee \exists z ((x \wedge z) \vee (y \wedge z)).$$

A recent result gives a characterization of the class *PSPACE* in terms of *interactive proofs*, a modern version of the Socratic dialog. It uses a notion of probabilistic Turing machine in which the transitions are made at random. Thus, instead of accepting an input x, there is a probability that the input is accepted. The computation proceeds in rounds between two Turing machines, the *Prover* and the *Verifier*. The input is considered as an assertion to be proved. The prover and the verifier communicate at each round and the number of rounds is polynomially bounded. Now a language L is in *IP* (for *Interactive Polynomial*) if there exists a probabilistic Turing machine V (the Verifier) working in polynomial time such that $x \in L$ iff there exists a Turing machine P (the Prover) such that x is accepted by the pair (P, V) with probability more than $1 - \epsilon$ for some constant $\epsilon < 1/2$. An example of a problem in *IP* is the non-isomorphism of graphs. The verifier guesses an index $i \in \{1, 2\}$ and a permutation π. It sends to the prover the graph $H = \pi(G_i)$. The verifier founds $j \in \{1, 2\}$ such that G_j and H are isomorphic. The verifier accepts if $i = j$.

It was proved by A. Shamir in 1989 that actually *IP=PSPACE* (see [47] for example).

4.4 Quantum computing

The theory of quantum computing arose from a suggestion by the physicist R. Feynman [16] that the Turing machine might not be the appropriate model to simulate a general quantum physical system.

A theory of quantum computation arose in the years 90, initiated in particular by D. Deutsch [12]. The idea is to use a state space represented by a finite dimensional vector space over the field of complex numbers and to use transitions as unitary transformations of this space. A quantum state is a vector of norm 1. Thus, the norms of the coordinates can be interpreted as the probability of being in a given ordinary state. For example, the transformation with orthogonal matrix

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

transforms the vector $\begin{bmatrix} 1 & 0 \end{bmatrix}$ (interpreted as a bit with value 0 with probability 1) into the vector $\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$ (where the bit has equal probability of value 0 or 1).

The main result was shown by P. Shor [51], who proved that factoring integers is polynomial in the quantum complexity model (while it is thought not to be in P).

4.5 Circuit complexity

Computing the value of a boolean function of n variables gives rise to a circuit, which is just a directed acyclic graph with $2n$ source nodes where each node is either an *OR* or an *AND* function (see Figure 2). Such a circuit recognizes

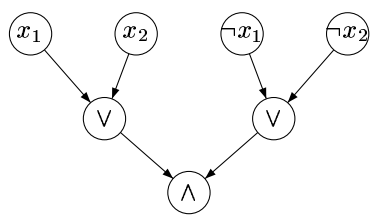

Figure 2: A circuit computing the parity of x_1 and x_2.

a set of words in $\{0,1\}^n$, namely those corresponding to a value one of the function. We denote by AC^0 the class of languages recognized, for each length n by circuits of size bounded by a polynomial in n and depth bounded by of constant. The more restricted class NC^1 is formed by the languages recognized by circuits in which the depth may be logarithmic, instead of constant, but the in-degree of each *AND* or *OR* gate is 2. One has $AC^0 \subset NC^1$. It has been shown [17] that the inclusion is strict since the parity function (which is obviously in NC^1) is not in AC^0. Further work initiated by Barrington and Therien (see [53]) showed that there is a link between circuit complexity and varieties of semigroups. In particular, the fact that the parity function is not in AC^0 is a consequence of the fact that the rational languages in AC^0 are the star-free languages (see Section 2.4).

5 Formal series

Instead of considering just sets of words, it is mathematically natural to consider functions from the set of words into a set of numerical values. For example, the interpretation of a binary word as the expansion of an integer at base 2 is such a function. It can be computed as the upper-right element of the matrix obtained via the substitution

$$0 \to \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \quad 1 \to \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

This approach can capture several important notions such as multiplicities (when the values are integers) or probabilities (when the values are real numbers).

5.1 Rational series

Formally, these functions are often called formal series and denoted as a sum

$$\sum (S, w) w$$

of words w with a coefficient w. A series on the alphabet A is said to be rational if there is a morphism μ from A^* in the monoid of $n \times n$ matrices such that $(S, w) = \lambda\mu(w)\gamma$ for some vectors λ, γ. This algebraic set-up has the advantage of simplifying many concepts. Thus, rational languages correspond to solutions of systems of linear equations. As an example, the rational language $X = (ab+b)^*$ is the solution of the equation $X = (ab+b)X + 1$. The generating series $f(z) = \sum f_n z^n$ of the number of words of length n in X is thus the rational fraction

$$f(z) = \frac{1}{1 - z - z^2}.$$

5.2 Algebraic series

In the same way, context-free languages correspond to solutions of algebraic equations. For example, the Dyck language generated by the grammar $D \to aDbD|\epsilon$ is just the solution of the equation $D = aDbD + 1$. Accordingly, the generating series $d(z) = \sum d_n z^n$ of the number d_n of words of length n in D is the series

$$d(z) = \frac{1 - \sqrt{1 - 4z^2}}{2}$$

solution of the equation $2z^2 d(z) - d(z) + 1 = 0$.

The use of formal series has found important applications to the analysis of algorithms (see [48] for example) so that to combinatorial enumeration.

6 Combinatorics on words

The combinatorial problems using words where studied very early and the papers of Axel Thue [54, 55] represent a historical mark. The most classical result is the existence of infinite square-free words, a result originally due to Thue and reproved several times later. A square in a word is a factor of the form ww, i.e. repeated twice immediately. The simplest way to obtain a square-free word is the following. Start with the Thue-Morse word

$$t = abbabaab\cdots$$

defined as follows. Let $\beta(n)$ denote the number of 1 in the binary expansion of n. Then $t_n = a$ if $\beta(n)$ is even and $t_n = b$ if it is odd. Then form the word

$$m = abcacbabcbac\cdots$$

which is the inverse image of t under the substitution

$$a \to abb, \quad b \to ab, \quad c \to a.$$

Then it can be shown that m is square-free.

The Thue-Morse word is obviously not square-free since it is on a binary alphabet and every long enough binary word has a square. However, it it is cube-free and even more: it does not contain an overlap, i.e. a factor of the form $uvuvu$ with u non-empty. The study of repetitions in words has done a lot of progress since Thue's initial work (see [32] for a survey).

6.1 Sturmian words

Sturmian words have a nice history going back to the astronomer Jean Bernoulli (see the chapter by Jean Berstel and Patrice Séébold in [32]). One of the equivalent definitions of a Sturmian word is that of an infinite word x such that for each n, the number $p(n)$ of distinct blocks of length n appearing in x is $n+1$ (it can be shown that if $p(n) \leq n$, then it is actually constant and the word x is ultimately periodic). The simplest example of a Sturmian word is the Fibonacci word

$$f = 01001010\cdots$$

which is the fix-point of the substitution $a \to ab, b \to a$. There are 3 factors of length 2, namely $00, 01$ and 10 and 4 factors of length 3, namely $001, 010, 100$ and 101. Actually, one has also an alternative definition of the Fibonacci word using approximations of irrationals by rationals. Let indeed s be the sequence

$$s_n = \lfloor (n+1)\alpha \rfloor - \lfloor n\alpha \rfloor$$

where $\alpha = 1/\varphi^2$ with $\varphi = (1+\sqrt{5})/2$ and where $\lfloor x \rfloor$ denotes the lower integral part of x. Then one has $s = 0f$. This formula show that the symbols f_n can be interpreted as the approximation of a line of slope α by points with integer coordinates (see Figure 6.1). It is a theorem due to Morse and Hedlund that Sturmian words can be defined equivalently by the property of having $n+1$ factors of length n or by a formula as above with α an irrational number.

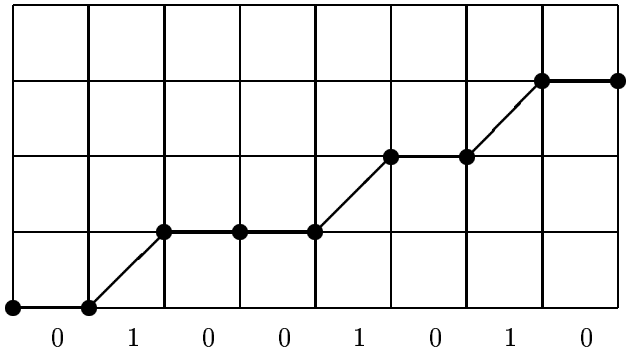

Figure 3: The graphical representation of the Fibonacci word

6.2 Equations in words

Equations in words have been the subject of active research. Such an equation is just a pair $x = y$ of two words on an alphabet $A \cup X$ of constants from A and unknowns from X. A solution is a substitution of a word on A for each unknown $x \in X$. For example, the equation $ax = yb$ has the solution $x = b, y = a$. It was shown first by Makanin that the existence of a solution is a decidable problem. See [32] for a survey on this subject.

7 Applications

We briefly mention below some of the practical applications of the notions and methods described above.

7.1 Compilers

The possibility of compiling a high-level programming language in an efficient way can be considered as one of an achievement of the history of computing. The first Fortran compiler was written by J. Backus in the years 1950. Since then, hundreds of programming languages have been designed and writing a compiler is part of the process in the language. The link with formal languages is roughly as follows. The lexical part of a compiler dealing with low-level notions such as format of the input is described by finite automata. Several software tools exist to easy the implementation of this part, known as *lexical analysis*. Second, the syntax of a programming language is usually described using a context-free grammar (or an equivalent formalism). The process of checking the syntactic correctness (and computing the syntactic structure) is known as *syntax analysis*. It is performed by methods which implement a form a pushdown automaton (see Section 3.3). The translation from the source language to the

object language (some kind of low-level machine language) is a third part of the process implementing a tree traversal which can be described by *attribute grammars*.

The possibility of using the methods of formal language theory for the construction of compilers is recognized as a striking fact (see [1] for example).

7.2 Text processing

The problems involved with text-processing are relatively low-level but of everyday importance. They include the use of word processors for the public and also the more complicated techniques used by professionals to produce high quality printed texts.

A domain of active research has been the study of pattern-matching algorithms. The most famous of these algorithms is the Knuth-Morris-Pratt algorithm [25] which allows to locate a word w in a text t in time proportional to the sum of the lengths of w and t (and not their product as in the naive algorithm looking for all possible positions of w in t at every index). This algorithm is actually closely linked with the computation of the minimal automaton recognizing the set of words ending with w.

A great number of algorithms have been devised to perform the compression of texts. This is important to speed-up the transmission as well as reduce the size for archives. Most of them are based on ideas relevant to finite automata and formal languages. One the most famous is the Ziv-Lempel method which builds a factorization of the input in blocks $x_1 x_2 \cdots x_n \cdots$ where x_n is the shortest word which is not in the list $(x_1, x_2, \ldots, x_{n-1})$.

7.3 Natural languages

The use of automata and grammars in natural language processing is present since the origins of these concepts as we have seen. The use of finite automata (often under the name of transition networks in this context) is wide, with applications to the description of phenomena in phonetics as well as lexicography or description of elementary sentences (see [41] for an overview). The use of grammars allows to handle nested structures such as those generated by subordinate clauses.

7.4 Genomes

The progress of molecular biology, and in particular the discovery of the genetic code, has opened a field now called *computational biology* dealing with biological sequences as computational objects (see [57] for an introduction).

Many algorithms have been applied to the analysis of biological sequences and some of them have been specifically design for this purpose. One of the most famous of these algorithms is the sequence comparison based on the search of a longest common subsequence. This algorithm, originally due to Hirschberg has been rediscovered by many authors. It allows to find by a technique often

called *dynamic programming* the longest common subsequence of two sequences in time proportional to the product of the lengths of the sequences. This can be viewed as approximate string-matching and is the object of active research (see [20] or [33]).

8 Notes on the bibliography

Although there exist many books treating the subject of automata and formal languages, few of them present a complete overview. This is probably due in part to the variation of point of views, either from the mathematical side or the computer science and engineering one. The classical textbook for computer scientists is the book of Hopcroft and Ullman [23], which covers finite automata, context-free grammars and also computability and Turing machines. The two volumes of Eilenberg [13, 14] cover automata with multiplicities, semigroups and varieties. Its publication is a landmark in the history of the subject because of the high quality of presentation and mathematical sophistication. The little book of Conway [10] is both elegant and challenging by its original viewpoint. The classical books on formal series are those of Salomaa and Soittola [45] and of Berstel and Reutenauer [6]. The series of Lothaire books [31, 32, 33] has been written to serve as a handbook in the field of combinatorics on words. The Handbook of Theoretical Computer Science [28, 29] covers all fields of Theoretical Computer Science. The volume B contains survey chapters on finite automata, infinite words, context-free grammars, computability,... More specifically, the Handbook of Formal Languages [42, 43, 44] is a source of information on all aspects of formal languages, including most of those mentioned here.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, Principles, Techniques and Tools.* Addison Wesley, 1986.

[2] John W. Backus. The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM Conference. In *Proc. Intl. Conf. on Information Processing*, pages 125–132, Paris, 1959. UNESCO.

[3] Thimothy Bell, John Cleary, and Ian Witten. *Text Compression.* Prentice Hall, 1990.

[4] Jean Berstel and Luc Boasson. Towards an algebraic theory of context-free languages. *Fund. Inform.*, 25(3-4):217–239, 1996.

[5] Jean Berstel and Dominique Perrin. *Theory of Codes.* Academic Press, 1985.

[6] Jean Berstel and Christophe Reutenauer. *Rational Series and their Languages.* Springer, 1988.

[7] Ronald Book and Friedrich Otto. *String-Rewriting Systems.* Springer, 1993.

[8] Noam Chomsky. Three models for the description of language. *IRE Trans. Inf. Th.*, IT-2, 1956. (Proc. of the Symposium on Information Theory, sept. 1956).

[9] Noam Chomsky. *Syntactic Structures.* Mouton, 1957.

[10] John H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.

[11] Maxime Crochemore and Wojciech Rytter. *Text Algorithms.* The Clarendon Press Oxford University Press, New York, 1994.

[12] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. London A*, 400:97–117, 1985.

[13] Samuel Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.

[14] Samuel Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

[15] David B. A. Epstein, James W. Cannon, Derek F. Holt, Silvio V. F. Levy, Michael S. Paterson, and William P. Thurston. *Word Processing in Groups.* Jones and Bartlett Publishers, Boston, MA, 1992.

[16] Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, 1982.

[17] Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits and the polynomial time hierarchy. *Math. Syst. Theory*, 17:13–27, 1984.

[18] Sheila Greibach. Formal languages: origins and directions. *Ann. Hist. Comput.*, 3, 1981.

[19] Maurice Gross and André Lentin. *Notions sur les Grammaires Formelles.* Gauthier-Villars, 1967.

[20] Dan Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge University Press, Cambridge, 1997.

[21] Zelig Harris. From morpheme to utterance. *Language*, 22:161–183, 1946.

[22] Kosaburo Hashiguchi. Algorithms for determining relative star heigth and star height. *Information and Computation*, 78:124–169, 1987.

[23] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[24] Stephen C. Kleene. Representation of events in nerve nets and finite automata. In C.E.Shannon and J.McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, 1956.

[25] Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[26] Keneth Krohn and John L. Rhodes. Algebraic theory of machines, I prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965.

[27] Gérard Lallement. *Semigroups and Combinatorial Applications.* Wiley, 1979.

[28] J. Van Leeuwen. *Handbook of Theoretical Computer Science, vol. A: Algorithms and Complexity.* North Holland, 1990.

[29] J. Van Leeuwen. *Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics.* North Holland, 1990.

[30] Douglas Lind and Brian Marcus. *Symbolic Dynamics and Coding.* Cambridge University Press, 1996.

[31] M. Lothaire. *Combinatorics on Words.* Cambridge University Press, 1982.

[32] M. Lothaire. *Algebraic Combinatorics on Words.* Cambridge University Press, 2002.

[33] M. Lothaire. *Applied Combinatorics on Words.* Cambridge University Press, 2003. (to appear).

[34] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[35] Robert McNaughton and Seymour Papert. *Counter-Free Automata.* MIT Press, 1971.

[36] David E. Muller and Paul E. Schupp. Groups, the theory of ends, and context-free languages. *J. Comput. System Sci.*, 26(3):295–310, 1983.

[37] Dominique Perrin and Jean-Eric Pin. *Infinite Words, automata, semigroups, logic and games.* Elsevier, 2003. (to appear).

[38] Emil L. Post. Finite combinatory processes-Formulation 1. *J. Symbolic Logic*, 1:103–105, 1936.

[39] Emil L. Post. Formal reductions of the general combinatorial decision problem. *Amer. J. Math.*, 65:197–215, 1943.

[40] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of Research and developpement*, 3:114–125, 1959. repris dans Sequential Machines, E.F.Moore ed.,Addison Wesley,1964, pp.63-91.

[41] Emmanuel Roche and Yves Schabes. *Finite-state Language Processing.* MIT Press, 1997.

[42] Grzegorz Rozenberg and Arto Salomaa. *Handbook of Formal Languages*, volume 1, Word, Language, Grammar. Springer, 1997.

[43] Grzegorz Rozenberg and Arto Salomaa. *Handbook of Formal Languages*, volume 2, Linear Modeling, Background and Application. Springer, 1997.

[44] Grzegorz Rozenberg and Arto Salomaa. *Handbook of Formal Languages*, volume 3, Beyond Words. Springer, 1997.

[45] A. Salomaa and Matti Soittola. *Automatic-Theoretic Aspects of Formal Power Series.* New York, Springer-Verlag, 1978.

[46] David Sankoff and Joseph B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison.* Addison-Wesley Publishing Company Advanced Book Program, Reading, MA, 1983.

[47] Uwe Schöning and Randall Pruim. *Gems of Theoretical Computer Science.* Springer, 1998.

[48] Robert Sedgewick and Philippe Flajolet. *Analysis of algorithms.* Reading, Mass., Addison Wesley, 1996.

[49] Géraud Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Automata, languages and programming (Bologna, 1997)*, volume 1256 of *Lecture Notes in Comput. Sci.*, pages 671–681. Springer, Berlin, 1997.

[50] Géraud Sénizergues. $L(A) = L(B)$? A simplified decidability proof. *Theoret. Comput. Sci.*, 281(1-2):555–608, 2002. Selected papers in honour of Maurice Nivat.

[51] Peter Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26:1484–1509, 1997.

[52] Charles Sims. *Computation with finitely presented groups*, volume 48 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 1994.

[53] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, 1994.

[54] Axel Thue. Über unendliche Zeichenreihen. *Norske Vid. Selsk. Skr. I Math-Nat. Kl.*, 7:1–22, 1906.

[55] Axel Thue. Über die gegenseitige Loge gleicher Teile gewisser Zeichenreihen. *Norske Vid. Selsk. Skr. I Math-Nat. Kl. Chris.*, 1:1–67, 1912.

[56] Alan Turing. On computable numbers with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936-37.

[57] Michael S. Waterman. *Introduction to Computational Biology.* Chapman and Hall, 1995.