

AUTOMATE, a computing package for automata and finite semigroups

J.M. CHAMPARNAUD¹ and G. HANSEL²

May 23, 1991

1. *L.I.T.P., Université Paris 7, Tour 55-56, 2 Place Jussieu, 75221 Paris Cedex 05, France.*
2. *Département de Mathématiques, Faculté des Sciences de Rouen, 76130 Mont-Saint-Aignan.*

Abstract

AUTOMATE is a package for symbolic computation on finite automata, extended rational expressions and finite semigroups. On the one hand, it enables one to compute the deterministic minimal automaton of the language represented by a rational expression or given by its table. On the other hand, given the transition table of a deterministic automaton, AUTOMATE computes the associated transition monoid. The regular \mathcal{D} -classes structure, and many properties of the elements in the monoid are provided. The program AUTOMATE has been written in C and is quite portable. The user interface includes specialized editors for easy displaying of the computed results.

Introduction

AUTOMATE is a package for symbolic computation on finite automata, extended rational expressions and finite semigroups. The original version which is described in this paper has been developed in the “Laboratoire d’Informatique Théorique et Programmation”³ by Jean-Marc Champarnaud and Georges Hansel in 1984-86. Several researchers in theoretical computer science have already used AUTOMATE with success for their own works [3,22,23,25].

Section 1 gathers the prerequisites in formal language and automaton theory useful for a good understanding of the package.

Section 2 presents the main areas of finite automata in theoretical and practical computer science, and also recalls the place of finite semigroups in regular language theory.

Section 3 is dedicated to the external use of AUTOMATE. The main characteristics and the functionalities of the package are described, and the user is guided through a working session of AUTOMATE.

Section 4 deals with the internal aspect of the AUTOMATE program. A description of the main data types is given as well as a justification of the algorithms which have been implemented.

Section 5 discusses the performance of AUTOMATE. Moreover in Sections 4 and 5, we compare AUTOMATE with previous packages designed to perform similar computations.

³Paris VI University, Paris VII University and C.N.R.S.

1 Theoretical prerequisites

In this section, we collect the theoretical notions about automata, rational expressions and monoids which are useful for a good understanding of AUTOMATE. To get more details on automata and rational expressions, good references are the books of J. Berstel [4], S. Eilenberg [9], J. E. Hopcroft and J. D. Ullman [11] and for monoids the books of G. Lallement [16] and J.E. Pin [21].

1.1 Automata

Let A be a finite alphabet and let A^* be the free monoid generated by A , i.e. the set of the “words” over the alphabet A . A *language (over the alphabet A)* is a subset of A^* . An *automaton* (finite or not) over the alphabet A is a 4-uple $\mathcal{A} = (Q, I, F, E)$ where Q is a set of *states*, I is a subset of Q whose elements are the *initial states*, F is a subset of Q whose elements are the *final states*, E is a subset of the cartesian product $Q \times A \times Q$ whose elements are the *edges*.

Let $\mathcal{A} = (Q, I, F, E)$ be an automaton. A *path* of \mathcal{A} is a sequence (q_i, a_i, q_{i+1}) , $i = 1, \dots, n$, of consecutive edges. Its *label* is the word $w = a_1 a_2 \dots a_n$. A word $w = a_1 a_2 \dots a_n$ is *recognized* by the automaton \mathcal{A} if there is a path with the label w such that $q_1 \in I$ and $q_{n+1} \in F$. The language *recognized* by the automaton \mathcal{A} is the set of the words which it recognizes. The automaton \mathcal{A} is *trim* if for all $q \in Q$, there is at least a path through q beginning at an initial state and ending at a final state. The automaton \mathcal{A} is *deterministic* if there is only one initial state and if for all $(q, a) \in Q \times A$ there is at most one state q' such that $(q, a, q') \in E$. Note that for a deterministic automaton, each letter $a \in A$ defines a partial mapping $\bar{a} : Q \rightarrow Q$ defined by $\bar{a}(q) = q'$ if $(q, a, q') \in E$.

1.2 Restricted rational expressions

Let A be an alphabet. An *atomic expression (on A)* is either a letter $a \in A$, or the empty word denoted by 1. A (*restricted*) *rational expression* is obtained by applying recursively the following operators to the atomic expressions: if r and r' are two rational expressions, one defines their *union* $r \cup r'$, their *product* rr' , the *star* of r written r^* ⁴.

To each rational expression r is associated a language $L(r)$, that is to say a subset of A^* recursively defined as follows:

to each letter $a \in A$ is associated the single element subset $L(a) = \{a\}$; if r and r' are two rational expressions, the associated languages of which are already defined, then to the union $r \cup r'$ is associated the language $L(r \cup r') = L(r) \cup L(r')$, union of the languages $L(r)$ and $L(r')$, to the product rr' is associated the product

⁴To be more precise, one should write $(r) \cup (r')$, $(r)(r')$ and $(r)^*$

language of $L(r)$ and $L(r')$, i.e. $L(rr') = \{uv/u \in L(r), v \in L(r')\}$, to the star r^* is associated the language $L(r^*) = \cup_n [L(r)]^n$.

A language $L \subset A^*$ is *rational* if there is a rational expression r such that $L = L(r)$.

1.3 Extended rational expressions

The *extended rational expressions* are obtained by adding new operators to combine the rational expressions. In the AUTOMATE program the following operators have been implemented:

- The *plus* operator written $+$: to the expression r^+ is associated the language $L(r^+) = L(r^*) \setminus \{1\}$.
- The *intersection* operator written I : to the intersection rIr' of two expressions is associated the language $L(rIr') = L(r) \cap L(r')$, intersection of the languages $L(r)$ and $L(r')$.
- The *difference* operator written \setminus : to the difference $r \setminus r'$ of two expressions is associated the language $L(r \setminus r') = L(r) \setminus L(r')$, the set difference of the languages $L(r)$ and $L(r')$.
- The *left quotient* operator written G : to the left quotient $r'Gr$ of the expression r by the expression r' is associated the language

$$L(r'Gr) = \{v \in A^* \mid \exists u \in L(r') : uv \in L(r)\}.$$

- The *right quotient* operator written D : to the right quotient rDr' of the expression r by the expression r' is associated the language

$$L(rDr') = \{v \in A^* \mid \exists u \in L(r') : vu \in L(r)\}.$$

- The *shuffle product* operator written W : to the shuffle rWr' of two expressions is associated the shuffle of the languages $L(r)$ and $L(r')$ i.e. the set $L(rWr')$ defined by

$$L(rWr') = \{u_1v_1u_2v_2 \dots u_nv_n \mid u_1u_2 \dots u_n \in L(r), v_1v_2 \dots v_n \in L(r')\},$$

the u_i and v_i being arbitrary words.

1.4 Kleene's theorem

Let A be a finite alphabet and $L \subset A^*$ a language. The equivalence relation on A^* denoted by \sim on A^* is defined as follows

$$u \sim v \text{ if and only if for all } w \in A^*, uw \in L \Leftrightarrow vw \in L.$$

For all $u \in A^*$, the class of the word u is denoted by $cl(u)$. One checks that the relation \sim is right regular, i.e. that if $u \sim v$, then for all $w \in A^*$, $uw \sim vw$. As a consequence, one can define *the deterministic minimal automaton \mathcal{A}_L of the language L* as follows: its set Q of states is the set of the classes $cl(u)$, $u \in A^*$, the only initial state is the class $cl(1)$ of the empty word, its set of final states is $F = \{cl(u) \mid u \in L\}$, its set E of edges is the set of the 3-uples (q, a, q') such that if $u \in q$, then $cl(ua) = q'$ (this definition is possible as a consequence of the regularity of \sim). One checks that the automaton \mathcal{A}_L recognizes the language L .

The essential link between automata and rational expressions is then given by the following theorem.

Theorem 1 (Kleene) *A language L is rational if and only if the automaton \mathcal{A}_L has a finite number of states (and moreover \mathcal{A}_L is minimal among the automata recognizing L as far as the number of states is concerned).*

1.5 Monoids

A *monoid* is a set equipped with an associative operation (which we write as a multiplication) having an identity element (written 1)⁵. In this paper we only consider finite monoids.

Let M be a monoid. An element $e \in M$ is *idempotent* if $e^2 = e$. An *ideal* (respectively *right ideal*, *left ideal*) is a subset $I \subset M$ such that $MIM \subset I$ (respectively $IM \subset M$, $MI \subset I$).

⁵Remind that if there is no identity element, this set is called a "semigroup"; since it is always possible to add an auxiliary identity element to a semigroup and thus to get a monoid, we only consider this case.

1.5.1 Green's relations

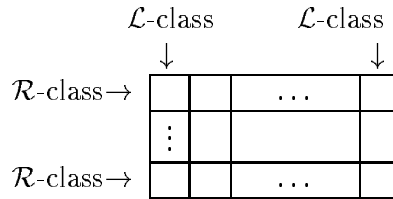
Let M be a monoid. On the monoid M , the four *Green's equivalence relations* written $\mathcal{R}, \mathcal{L}, \mathcal{H}, \mathcal{J}$ are defined, for all pairs (x, y) of elements of M , in the following way: $x\mathcal{R}y$ if x and y generate the same right ideal ($xM = yM$), $x\mathcal{L}y$ if x and y generate the same left ideal ($Mx = My$), \mathcal{H} is the intersection of the relations \mathcal{R} and \mathcal{L} , $x\mathcal{J}y$ if x and y generate the same ideal ($MxM = MyM$).

It turns out that the relations \mathcal{R} and \mathcal{L} commute and that the relation

$$\mathcal{D} = \mathcal{R} \circ \mathcal{L} = \mathcal{L} \circ \mathcal{R}$$

is the smallest equivalence relation containing \mathcal{R} and \mathcal{L} . Then a \mathcal{D} -class is the disjoint union of the \mathcal{R} -classes (respectively of the \mathcal{L} -classes) which it contains. Moreover each \mathcal{H} -class is the intersection of a \mathcal{R} -class and a \mathcal{L} -class. As a consequence of the foregoing, a \mathcal{D} -class can be schematically represented by an "an egg-box" (figure 1). Furthermore, if M is finite, an important result is that the relations \mathcal{D} and \mathcal{J} coincide.

figure 1: The egg-box of a \mathcal{D} -class



We shall use the following results ([21] chapter 3, propositions 1.5 , 1.6 and 1.7).

Proposition 1 *Let M be a monoid, a and b elements of M in the same \mathcal{R} -class. Let u be such that $b = au$. Then the mapping $x \rightarrow xu$ from M to M induces a bijection from the \mathcal{H} -class H_a of a onto the \mathcal{H} -class H_b of b .*

Proposition 2 *Let M be a monoid, x and y elements of M , R_x and R_y the \mathcal{R} -classes of x and y , L_x and L_y their \mathcal{L} -classes. Then $xy \in R_x \cap L_y$ if and only if $R_y \cap L_x$ contains an idempotent.*

Proposition 3 *Let M be a monoid and let H be an \mathcal{H} -class of M . Then H contains an idempotent element if and only if H is a maximal subgroup of M .*

1.6 Regular \mathcal{D} -classes

Let M be a monoid. An element $a \in M$ is *regular* if there exists $m \in M$ such that $ama = a$. A \mathcal{D} -class is *regular* if all its elements are regular. The following proposition gives several characterizations of the regular \mathcal{D} -classes in a monoid.

Proposition 4 *Let M be a monoid and let D a \mathcal{D} -class of M . The following conditions are equivalent:*

- 1) D is regular
- 2) D has a regular element
- 3) D has at least one idempotent
- 4) Each \mathcal{R} -class (respectively each \mathcal{L} -class) contained in D has an idempotent.

1.7 Transformation monoids

Let Q be a finite set. The set T of all the partial mappings from Q to Q , equipped with the usual composition, is a monoid called the *transformation monoid of Q* . In our context, it is convenient to write the mappings on the right of their arguments; if $t : Q \rightarrow Q$ is an element of T and if $q \in Q$, we denote the image of q under t by $q \cdot t$.

Let Q be a set and T the transformation monoid of Q , let $t \in T$; we write $Im(t)$ the image of t , i.e. the set of the $q \cdot t$, $q \in Q$, and $Ker(t)$ the partition of Q induced by the equivalence $q \approx q'$ if $q \cdot t = q' \cdot t$ or if $q \cdot t$ and $q' \cdot t$ are not defined.

In the monoid T , Green's relations have a simple form as a consequence of the following result.

Proposition 5 *Let T be the transformation monoid of the finite set Q , x and y two elements of T . Then:*

- 1) xRy if and only if $Ker(x) = Ker(y)$
- 2) xLy if and only if $Im(x) = Im(y)$.

Let T be the transformation monoid of a finite set Q and let M be a submonoid of T . In this case, we denote the Green's relations in T by \mathcal{R} , \mathcal{L} , \mathcal{H} and the Green's relations in M by \mathcal{R}_M , \mathcal{L}_M , \mathcal{H}_M . Proposition 5 is not generally true in the submonoid M because the restrictions to M of the relations \mathcal{R} and \mathcal{L} can be coarser than the relations \mathcal{R}_M and \mathcal{L}_M . However, the following result holds ([21] chapter 3, problem 1.10).

Proposition 6 *Let T be the transformation monoid of the finite set Q and let M be a submonoid of T . Then on the set of the regular elements of M , the relations \mathcal{R} and \mathcal{L} in T and M coincide.*

The following proposition gives a criterion used by AUTOMATE to characterize the idempotents belonging to a same (necessarily regular) \mathcal{D} -class.

Proposition 7 *Let T be the transformation monoid of the finite set Q and let M be a submonoid of T . Let e and f be two idempotents of M . The following conditions are equivalent:*

- a) e and f are in the same \mathcal{D} -class of M
- b) there are elements x and y in M such that $Ker(x) = Ker(e)$ and $Im(x) = Im(f)$, $Ker(y) = Ker(f)$ and $Im(y) = Im(e)$.

Proof Let us show that $a) \Rightarrow b)$. By hypothesis, there exists $x \in M$ such that $e\mathcal{R}x$ and $x\mathcal{L}f$; hence by proposition 5, we have

$$\text{Ker}(x) = \text{Ker}(e) \text{ and } \text{Im}(x) = \text{Im}(f).$$

The existence of y is proved in the same way.

Let us show that $b) \Rightarrow a)$. By proposition 2, the elements x and y are such that for the relation \mathcal{H} ,

$$xy\mathcal{H}e \quad \text{and} \quad yx\mathcal{H}f.$$

Hence by proposition 3, H_{xy} is a group and therefore, there exists an integer n such that $(xy)^n = e$. Moreover, since e and x are \mathcal{R} -equivalent and since e is idempotent, we have the equality $ex = x$. Since x belongs to M , the equalities $(xy)^n = e$ and $ex = x$ show that e and x are \mathcal{R}_M -equivalent.

A similar argument shows that x and f are \mathcal{L}_M -equivalent and consequently, condition a) is satisfied.

The following proposition is used by AUTOMATE to work out the regular elements of a transformation submonoid when the idempotents have already been sorted according to their \mathcal{D} -classes.

Proposition 8 *Let T be the transformation monoid of the finite set Q , let M be a submonoid of T and let $a \in M$. The following conditions are equivalent:*

- a) *a is regular*
- b) *Some \mathcal{D} -class of M contains idempotents e and f such that*

$$\text{Ker}(a) = \text{Ker}(e), \quad \text{Im}(a) = \text{Im}(f).$$

Proof a) \Rightarrow b) is clear: a being regular, there is $m \in M$ such that $ama = a$. The idempotents $e = am$ and $f = ma$ satisfy the required conditions.

Let us prove that b) \Rightarrow a). Since e and f are in the same \mathcal{D} -class D of M , there exists $b \in D$ such that $b\mathcal{R}e$ and $b\mathcal{L}f$. It follows that

$$\text{Ker}(b) = \text{Ker}(e) \text{ and } \text{Im}(b) = \text{Im}(f)$$

and therefore, according to the hypothesis on a , the elements a and b are in the same \mathcal{H} -class (for T).

Since b and e are in the same \mathcal{R} -class of M , there is $z \in M$ such that $bz = e$. By proposition 1, the mapping $x \rightarrow xz$ is a bijection from H_b onto H_e and then az and e are in the same \mathcal{H} -class. This class is a group with e as its identity element (proposition 3); consequently, there exists an integer n such that $(az)^n = e$ and on the other hand $eaz = az$. As $z \in M$, these equalities show that az and e are \mathcal{H}_M -equivalent. By proposition 1 the pre-images a and b of az and e by the bijection $x \rightarrow xz$ are also \mathcal{H}_M -equivalent and consequently a is regular.

1.8 Transition monoid of a deterministic automaton

Let $\mathcal{A} = (Q, I, F, E)$ be a deterministic automaton. To each letter $a \in A$ is associated the partial mapping $\bar{a} : Q \rightarrow Q$ defined by $q \cdot \bar{a} = q'$ if $(q, a, q') \in E$. Then by induction, to each word $w = ua$, $u \in A^*$, $a \in A$, is associated the mapping $\bar{w} : Q \rightarrow Q$ defined by

$$q \cdot \bar{w} = (q \cdot \bar{u}) \cdot \bar{a}.$$

The mapping $w \rightarrow \bar{w}$ is a morphism from A^* to the monoid T of the partial mappings from Q to Q and its image $\{\bar{w} \mid w \in A^*\}$ is a submonoid M called the *transition monoid of the automaton \mathcal{A}* .

2 Application fields

Studying finite automata is an old idea. It appeared in the fifties with the works of S. Kleene, in particular in his seminal paper “Representations of events in nerve nets and finite automata” [15]. Afterwards, automaton theory, in its different aspects, has turned out to be fruitful in many fields. We give here only some examples, without any claim to exhaustiveness.

Starting from a description of the lexical rules of a language, it is possible, with a computer, to get an automaton recognizing this language. For instance, the Lex program [18] generates such automata; then, these automata can be used to realize many modifications on an input text.

The Oxford English Dictionary has been computerized in order to make easier its updating and future revisions. Its structure has been described as a regular language [14], thus enabling the use of INR [13], a very efficient program computing finite automata and transducers.

Text editors and word processors currently make use of automata, especially for word searching and substituting in a text.

Lempel and Ziv’s text compression algorithm [26] is nowadays a basic tool; its formalization requires the notion of automaton.

The technology of very large scale integrated circuits (VLSI) makes use of the model of finite automata for some aspects of its development. The behaviour of a circuit is described by a rational expression and the associated automaton allows one to produce the circuit [10].

Monoids has mainly been studied from a theoretical point of view with theoretical goals. Besides the numerous and intrinsic questions to which they give rise [16], they appeared to be fruitful in the language classification problem. For instance, M. P. Schützenberger has proved that a rational language can be defined by a star-free expression if and only if its syntactic monoid has only trivial

groups. I. Simon has deepened this result in several directions. The study of language varieties is presently making great strides. A reference for all these works is the book of J. E. Pin [21].

3 AUTOMATE : external aspects

3.1 Characteristics of the AUTOMATE package

The AUTOMATE program has been written in the C language for portability and efficiency reasons.

From the portability point of view, AUTOMATE which was initially developed on VAX/ 780 with UNIX⁶ BSD 4.2, is now implemented

- under several Berkeley UNIX versions: UNIX BSD 4.3 on VAX/780, UTX⁷/32 2.0 on GOULD 6040 (close to BSD 4.3), ULTRIX⁸-32 Version 1.2 on SUN (close to BSD 4.2)

- under several SYSTEM V UNIX versions: UTX/32 2.0 on GOULD 6040 (with SYSTEM V extensions), SMX⁹ V.2D on SM90, AIX¹⁰ on IBM PC RT

- under MS/PC-DOS¹¹ on IBM PC and compatibles, in a slightly modified version

- on Macintosh Plus and Macintosh SE , in a reduced version.

From the efficiency point of view, pointer arithmetic in the C language is indeed an important element of the execution speed of AUTOMATE program. Section 5 deals more specifically with the performance of the program.

The AUTOMATE package is made up with two external commands: **automate** and **monoide**. Moreover, the **automate** command can start up the execution of the **monoide** command. The size of the source program source is about 6000 instructions for **automate** and 4000 instructions for **monoide**. The associated binary codes are respectively 70 Kbytes and 90 Kbytes long.

AUTOMATE is an interactive package with a quite versatile user interface. In the **automate** command, the requests set is intentionally reduced and the results can be displayed with the standard editors owing to a Shell escape. The **monoide** command offers a full-screen automaton editor as well as a full-screen editor dedicated to the scanning of the results in the case they have a large size. The richness of the user interface (numerous error messages and recalling of the requests list in the **automate** command, full-screen menus and editors in the **monoide** command) facilitates the use of the AUTOMATE package. Moreover,

⁶UNIX is a trademark of Bell/ATT

⁷UTX is a trademark of GOULD

⁸ULTRIX is a trademark of Digital

⁹SMX is a trademark of Telmat

¹⁰AIX is a trademark of IBM

¹¹MSDOS is a trademark of Microsoft Corporation

there is, for each of the **automate** and **monoide** command, a User Manual [5,6] containing commented examples of working sessions.

3.2 Functionalities

3.2.1 Results given by AUTOMATE

The AUTOMATE package mainly enables one to compute

- (i) *the deterministic non-complete minimal automaton of a rational language* entered either as an extended rational expression (e.r.e. in abbreviated form) or as a transition table; the expressions recognized by AUTOMATE include the following operators: finite union, finite concatenation product, Kleene's star, plus operator, words list, finite intersection, set difference, left and right quotients, and shuffle product.
- (ii) *the transition monoid of a deterministic automaton* given by its transition table; the computed results are the following ones:
 1. *a summary* of the transition monoid including:
 - the transition table
 - the number of the elements in the monoid
 - the numbers of the regular and of the idempotent elements
 - the number of the regular \mathcal{D} -classes
 - a table of the regular \mathcal{D} -classes sorted according to their rank and indicating
for each rank, the number of \mathcal{D} -classes
for each \mathcal{D} -class, the number of \mathcal{R} -classes, the number of \mathcal{L} -classes
and the number of elements in each \mathcal{H} -class.
 2. *the list of the elements in the monoid*, with for each element, the three following fields:
 - a word representing this element (the lowest in the "hierarchical" order induced by the right multiplication)
 - the associated transition
 - the type of the element: non-regular, regular, element of a group, idempotent.
 3. *the chart of the regular \mathcal{D} -classes*, sorted by their rank, and displayed under their classical "egg-box" presentation, with the possibility of scanning the set of the words in each \mathcal{H} -class with the help of a special editor.
 4. *a set of relations* defining the monoid.

5. the ability to compute, for an arbitrary word, its *representative* and its *type*.
- (iii) *the syntactic monoid of a rational language*: to obtain it, it is sufficient to compute the transition monoid of the minimal automaton of the language.

3.2.2 Data input and displaying

The data can either be entered from the keyboard, or be read in a file. A full-screen special editor is available to input a deterministic automaton in order to compute its transition monoid.

The output is generally “filtered” by a program such as **more** under UNIX BSD or **pg** under UNIX SYSTEM V. A full-screen special editor gives the ability to scan the table of the regular \mathcal{D} -classes in every direction (this table may be several dozens screen deep and several screens wide).

3.3 A session example through AUTOMATE

Here is a commented example of a working session through AUTOMATE. We have adopted the following conventions :

- a) the comments are added in smaller and slanted characters
- b) the characters entered by the user are written in boldface
- c) the answers of AUTOMATE are displayed with typewriter characters

This session has been worked out on a VAX/780 under BSD 4.3 UNIX.

\$ automate

SYSTEME AUTOMATE (LITP)

COMMANDS :

R	:	displaying of the results file
T	:	displaying of the last computation
=letter(s)	:	adding letter(s) to the alphabet
<file(s)	:	redirecting the standard input on the named file(s)
M	:	running the monoide command
!	:	running a UNIX command
Q	:	quit

Any command beginning by a character different from R,T,=,<,M,!, and Q is understood as an extended rational expression

Any expression must end with a semi-column

Any command must be validated by RETURN

R(results) T(trace) =(alphabet) <(file) M(monoide) !(UNIX) Q(quit)

automate>(aaaUbbbUzzz){a,b,z}*;

The alphabet of an e.r.e. is only made of lower case letters. It is the set of the lower case letters which appear in the e.r.e. Any e.r.e. must end with a ';' . The concatenation product is denoted by simple juxtaposition of expressions, U is the union operator, * denotes the star operator. The braces define a list of words : {a,b} is equivalent to aUb. The product has higher precedence than the union but the parenthesis enable one to change the precedence: in our example, the union of the words aaa, bbb and zzz will be first computed and afterwards the product of the expressions aaaUbbbUzzz and {a,b,z}*.

*
(aaaUbbbUzzz){a,b,z}

The minimal automaton has 8 states
on a 3 letters alphabet : {a, b, z}
There is 1 final state : 8

	1	2	3	4	5	6	7	8
a	5	8	0	0	2	0	0	8
b	6	0	8	0	0	3	0	8
z	7	0	0	8	0	0	4	8

For every e.r.e., AUTOMATE computes the deterministic non-complete minimal automaton. The "pit", if there is one, is written 0 and it is not counted as a member of the states. The initial state of the minimal automaton is always the state number 1.

R(results) T(trace) =(alphabet) <(file) M(monoide)!(UNIX) Q(quit)


```
automate> {a,b}*aA9;
```

The minimal automaton has 1024 states
on a 2 letters alphabet: {a, b}
There are 512 final states

Question: Do you wish to see the final states and the transitions ?

Answer (y/n <return>) : n

*If the displaying of the minimal automaton table requires more than one screen, AUTOMATE makes sure that the user really wishes to see this table on the screen. If the user begins the working session with the command **automate -r**, the successive results are kept in the res.automaton file; at any moment in the session, it is then possible to edit this file with the T request.*

```
automate> aba{c,d2}*aUc(b2)*a;
```

```
          2 *      2 *  
aba{c,d } aUc(b ) a
```

The minimal automaton has 8 states
on a 4 letters alphabet: {a, b, c, d}
There is 1 final state : 8

	1	2	3	4	5	6	7	8
a	4	8	0	0	0	7	8	0
b	0	5	0	6	2	0	0	0
c	2	0	0	0	0	0	7	0
d	0	0	7	0	0	0	3	0

```
automate> M
```

*When processing an M request, AUTOMATE creates a temporary file to save the last minimal automaton computed, and then it creates a process which runs the **monoide** command for this file. The user is now in a **monoide** session and he can investigate the results relative to the syntactic monoid of the expression %. The summary of the monoid is displayed.*

number of states : 8
 number of letters: 4

	1	2	3	4	5	6	7	8
a	4	8	-	-	-	7	8	-
b	-	5	-	6	2	-	-	-
c	2	-	-	-	-	-	7	-
d	-	-	7	-	-	-	3	-

Number of elements in the monoid : 31
 Regular elements : 10 Idempotents : 6 Regular D-classes : 5

Rank	8	2		1	0	

Nmbr D-cl.	1	2		1	1	

Nmbr R-cl.	1	1	: 1	2	1	

Nmbr L-cl.	1	1	: 1	2	1	

Nmbr/H-cl.	1	2	: 2	1	1	

The line "Nmbr D-cl." gives the number of regular \mathcal{D} -classes in each rank. The line "Nmbr R-cl." says how many \mathcal{R} -classes there are in each \mathcal{D} -class of a given rank; for instance there are two \mathcal{D} -classes of rank 2, each one having one \mathcal{R} -class as indicated by "1 : 1". The line "Nmbr/H-cl." says how many elements are contained in each \mathcal{H} -class.

The <space> key is pressed. The RESULTS menu is displayed.

RESULTS

- 1 Summary of the monoid
- 2 List of the elements in the monoid
- 3 Chart of the regular D-classes
- 4 Inspection of a particular word
- 5 List of the relations in the monoid
- 6 Results saving
- 7 Return to initial options

Enter the chosen option : 2

Option 2 is selected and the list of the elements in the monoid appears on the screen.

1	1 2 3 4 5 6 7 8 ***	aba	7 0 0 0 0 0 0 0
a	4 8 0 0 0 7 8 0	baa	0 0 0 8 0 0 0 0
b	0 5 0 6 2 0 0 0	bac	0 0 0 7 0 0 0 0
c	2 0 0 0 0 0 7 0	bad	0 0 0 3 0 0 0 0
d	0 0 7 0 0 0 3 0 **	bba	0 8 0 0 0 0 0 0
aa	0 0 0 0 0 8 0 0	bbb	0 5 0 0 2 0 0 0 **
ab	6 0 0 0 0 0 0 0	cbb	2 0 0 0 0 0 0 0
ac	0 0 0 0 0 7 0 0	cca	0 0 0 0 0 0 8 0
ad	0 0 0 0 0 3 0 0	dcd	0 0 3 0 0 0 0 0 ***
ba	0 0 0 7 8 0 0 0	abaa	8 0 0 0 0 0 0 0
bb	0 2 0 0 5 0 0 0 ***	abad	3 0 0 0 0 0 0 0
bc	0 0 0 0 0 0 0 0 ***	bbba	0 0 0 0 8 0 0 0
ca	8 0 0 0 0 0 8 0		
cb	5 0 0 0 0 0 0 0		
cc	0 0 0 0 0 0 7 0 ***		
cd	0 0 0 0 0 0 3 0 *		
da	0 0 8 0 0 0 0 0		
dc	0 0 7 0 0 0 0 0 *		
dd	0 0 3 0 0 0 7 0 ***		

The order of this list is the “hierarchical” one. Each line begins with the word which is the smallest among the words having the same transition function. The “type” of each element is indicated: * for a regular element, ** for an element belonging to a group, *** for an idempotent element.

The <space> key is pressed. The RESULTS menu is displayed. Option 3 is selected in order to call the editor of the regular D-classes; we get the following screen.

\wedge N down, \wedge P up, \wedge F right, \wedge B left, \wedge J next rank, \wedge I previous rank
 \wedge H home, \wedge K end, \wedge L refresh, \wedge A quit

```

1 D-class of rank 8
-----
| D1          |12345678|
-----
|1/2/3/4/5/6/7/8 |1      |
-----

2 D-classes of rank 2
-----
| D2          |25 | | D3          |37|
-----
|134678/2/5 |bb | |124568/3/7 |d |
|          |bbb| |          |dd|
-----

1 D-class of rank 1
-----
| D4          |3 |7 |
-----
|1245678/3 |dcd|dc|
-----
|1234568/7 |cd |cc|
-----

1 D-class of rank 0
-----
| D5          | |
-----
|12345678 |bc|
-----
  
```

The regular \mathcal{D} -classes chart may be very large: several hundreds lines and columns; thus, it is displayed with the help of a special editor which enables one to inspect it in detail.

The $\langle \wedge A \rangle$ key is pressed. The RESULTS menu is displayed. Option 5 is chosen in order to get the list of the monoid relations.

List of relations defining the monoid

```
bd = bc :db = bc :aaa = bc :aab = bc :aac = bc :aad = bc :abb = bc:abc = bc:
aca = aa :acb = bc :acc = ac :acd = ad :ada = bc :adc = bc :add = ac :
bab = bc :bbc = bc :bca = bc :bcb = bc :bcc = bc :bcd = bc :caa = bc :
cab = bc :cac = bc :cad = bc :cba = bc :cbc = bc :ccb = bc :ccc = cc :
ccd = cd :cda = bc :cdc = bc :cdd = cc :daa = bc :dab = bc :dac = bc :
dad = bc :dca = da :dcb = bc :dcc = dc :dda = cca :ddc = cc :ddd = d :
abac = aba :bbaa = bc :bbac = bc :bbad = bc :bbbb = bb :cbba = abaa :
cbbb = cb :
```

...

To continue, press Space

The <space> key is pressed. The RESULTS menu is displayed. The option 4 is selected in order to get the properties of a particular word.

This option computes the transition associated to an arbitrary given word as well as the smallest equivalent word. Moreover, the type (idempotent, element of a group or regular) is indicated.

...

Input the chosen word (then press Return)

The word abaccddda is entered. The following text appears on the screen.

```
abaccddda = abaa element of a group
This transition is defined by
```

```
1 2 3 4 5 6 7 8
8 0 0 0 0 0 0 0
```

...

To continue, press Space

The <space> key is pressed. The RESULTS menu is displayed. Option 6 is selected in order to save the results in a file.

...

Give the name of the save file: **monol.res**

The save file contains the summary, the lists of the elements and of the relations.

The RESULTS menu is displayed. Option 7 is selected and the INITIAL OPTIONS menu is displayed.

INITIAL OPTIONS

1. Input an automaton and compute its transition monoid

- 2. Compute the monoid of an automaton defined in a file
- 3. Help
- 4. Quit
- ...

Select an option: **1**

Option 1 is singled out in order to call the automata editor. The user has to indicate the number of states and their names, the number of letters and their names. The following screen appears, in which the user must enter the transition table of the automaton.

`^B=left, ^F=right, ^N=down, ^P=up, ^H erase, ^L refresh, ^A quit`

	1	2	3	4	5
a					
b					
c					
d					
...					

Enter the transition table

The following transition table is entered, using the editor control characters which are close to Emacs ones.

`^B=left, ^F=right, ^N=down, ^P=up, ^H erase, ^L refresh, ^A quit`

	1	2	3	4	5
a	2	3	-	1	-
b	3	-	5	2	1
c	2	2	3	3	4
d	2	1	3	5	4
...					

Is the automaton well defined ? (y/n) : **y**

Before displaying this last question, the editor makes sure that the states are valid, i.e. are numbers between 1 and 5. The computation is launched and the following screen gives the summary of the monoid.

Number of states : 5 Number of letters : 4

	1	2	3	4	5
a	2	3	-	1	-

```

b 3 - 5 2 1
c 2 2 3 3 4
d 2 1 3 5 4

```

```

Number of elements in the monoid : 1012
regular elements : 970 Idempotents : 182 regular D-classes : 6

```

```

      Rank | 5 | 3      | 2 | 1 | 0 |
-----
Nbr D-cl. | 1 | 2      | 1 | 1 | 1 |
-----
Nbr R-cl. | 1 | 6 : 2 |43 |29 | 1 |
-----
Nbr L-cl. | 1 | 2 : 2 | 9 | 5 | 1 |
-----
Nbr/H-cl. | 2 | 3 : 3 | 2 | 1 | 1 |
-----

```

```

...
To continue, press Space

```

The <space> key is pressed to return to the RESULTS menu...

The monoid computed in the last example has 1012 elements. The computation of the elements and of the regular \mathcal{D} -classes is 13 seconds long on VAX/780. The save file (summary + list of words + relations) has about 1100 lines. The regular D-classes chart has a size of 29437 characters in a file of 237 lines, which is 10 screens deep and 2 screens wide.

4 Internal aspect

4.1 Programming

An important characteristic of the AUTOMATE program is its modularity. To each class of objects (automata, transition monoids, ...) we associate a data type and several libraries of functions which enable one to deal with these types.

The C language is especially adapted to the programming of AUTOMATE. Indeed, it gives the possibility to dynamically allocate memory blocks the size of which is only known at run time (this is the usual situation for the computation of automata and monoids); moreover, pointer arithmetic in C increases the program speed.

4.2 Implemented algorithms

4.2.1 The “**automate**” command

The construction of the non-deterministic automaton is realized according to a modified version of Thomson’s method[24]. Minimization is done according to the method described in the book of Aho, Hopcroft and Ullman [1]. It is an $O(n \log n)$ algorithm, n being the number of states of the deterministic automaton.

The following features distinguish the **automate** command from the previous programs [2,17]:

- The automata which successively appear in our construction have a unique initial state and are trim.
- The program makes use of intermediate automata which are, whenever it is possible, directly deterministic (i.e. without an explicit determinization). The functions which operate on the automata take into account the possible deterministic character of their arguments. These features noticeably increase the computing speed.
- Determinization is the part of the program which usually requires most of the CPU time. Systematic use of binary search trees for the comparison between subsets of states appreciably alleviates this step of the computation.

4.2.2 The “**monoide**” command

The following features distinguish the **monoide** command from the previous programs [8,17,19,20]:

- All the elements of the transition monoid of a deterministic automaton are produced; only regular ones were computed in [8,20].
- For each element, the smallest representative according to the hierarchical order is given.
- A minimal set of relations between words which defines the monoid is exhibited.
- The regular \mathcal{D} -classes with their structure can be displayed and the content of each \mathcal{H} -class is given by the list of its minimal words.

The whole computation is made possible owing to the systematic use of binary search trees and owing to the taking into account of the relations which are progressively obtained as the work proceeds. More precisely the algorithm can be

split into two successive phases: first constructing the monoid and then computing the regular \mathcal{D} -classes.

1) Constructing the monoid

As far as the program is concerned, a monoid element is not exclusively a partial mapping but is a data structure collecting the following informations: the partial mapping from Q to Q associated to the element, the smallest word (in the hierarchical order) representative of the mapping, the kernel and the image of the mapping, the type of the element (idempotent, element of a group, regular, non-regular).

If v denotes the word representing the mapping, then the mapping itself is written \bar{v} , its kernel is written $k = k(v)$ and its image is written $i = i(v)$. In the sequel, we may identify the monoid element, its representative word, and the associated mapping.

In this first phase we simultaneously construct:

- the linked list of the elements in the monoid
 - the tree of the relations which define the monoid
- and for each rank three search binary trees
- a *kernel-tree*, the vertices of which are idempotent elements and such that, for each idempotent in the monoid, its kernel appears only once
 - an *image-tree*, the vertices of which are idempotent elements and such that, for each idempotent in the monoid, its image appears only once
 - the tree of the (kernel, image) pairs of the monoid elements; each vertex of this tree is labelled with a pair (k, i) and points to the tree of all the monoid elements the (kernel, image) pair of which is (k, i) . This last tree is written $A(k, i)$.

At each step of the construction, the tree of the relations and the trees of (kernel, image) pairs already produced are used to speed up the computation. Note that during this first phase, only the “idempotent” and “element of a group” characteristics are worked out.

Let us now describe the algorithm in more detail.

INITIALIZATIONS The list of the elements is initialized with the empty word 1 (the mapping $\bar{1}$ being the identity on Q). The tree of the (kernel, image) pairs of rank $\text{card}(Q)$ has as its root the pair $(k(1), i(1))$ and the tree $A(k(1), i(1))$ has as its root the mapping $\bar{1}$. This element is idempotent; hence, its kernel is the root of the kernel-tree and its image is the root of the image-tree. Moreover, this element is initially the current element in the list of the elements.

CURRENT STEP The current word u is successively concatenated with each letter a of the alphabet and for each word we get, we inspect if it represents a new element of the monoid. Let $v = ua$ be the resulting word to be inspected. We proceed as follows.

- If there is already a relation of the form $s = w$, s being a suffix of v , then the word v represents a monoid element which has already appeared.
- Else, let (k, i) be the (kernel, image) pair of the mapping \bar{v} ; if the vertex labelled (k, i) exists in the tree of the (kernel, image) pairs, and if the vertex labelled \bar{v} exists in the tree $A(k, i)$, again the word v represents a monoid element which has already appeared. Let w be the representative word of this element; we store the new relation $v = w$ in the tree of the relations.
- If the vertex labelled \bar{v} does not exist, the word v represents a new element of the monoid and we add a vertex labelled \bar{v} in the tree $A(k, i)$.
- If the vertex labelled (k, i) does not exist, the word v represents a new element of the monoid; we add a vertex labelled (k, i) in the tree of the (kernel, image) pairs and we create the tree $A(k, i)$ with \bar{v} as its root.

In case v is the representative of a new monoid element, we work out whether \bar{v} is idempotent or element of a group. If need be, the kernel-trees and the image-trees are completed.

2) Computing the regular \mathcal{D} -classes.

The regular \mathcal{D} -classes computation is realized with the help of the kernel-trees and of the image-trees. For each rank, we group together the idempotents of the kernel-tree into distinct lists, each list collecting the ones which belong to the same \mathcal{D} -class (which is necessarily regular). For this purpose, we make use of proposition 7: if e and f are two idempotents, they are in the same \mathcal{D} -class if and only if the pairs $(k(e), i(f))$ and $(k(f), i(e))$ are in the tree of the (kernel, image) pairs. The same processing is done on the image-tree.

For each \mathcal{D} -class, we get two lists (e_1, e_2, \dots, e_p) and (f_1, f_2, \dots, f_q) corresponding to the different kernels and images of the \mathcal{D} -class. According to proposition 8, the \mathcal{D} -class elements are then all the elements of the trees $A(k(e_j), i(f_l)), j = 1, \dots, p, l = 1, \dots, q$.

5 Performance

Let us consider the expression

$$E = c(a \cup b)^* b c^* ((a \cup b) c^*)^n,$$

with n from 6 to 10. The following table gives an example of the **automate** command performance.

n	n_1	n_2	n_3	t
5	21	191	128	1.5s
6	24	383	256	3.7s
7	27	767	512	8.3s
8	30	1535	1024	22.1s
9	33	3071	2048	64.2s
10	36	6143	4096	211s

In this table, the following indications are given: n is the exponent which appears in the expression, n_1 is the number of states in the non-deterministic automaton recognizing the expression, n_2 is the number of states in the associated deterministic automaton, n_3 is the number of states in the minimal automaton and t is the CPU time on VAX/780.

Among previous similar packages, REGPACK [17] seems to be the most efficient.¹² The computation of the expression E with $n = 5$ by REGPACK on IBM/370 is reported to be done in 35 seconds (versus 1.5 seconds by AUTOMATE).¹³

The next example gives an idea of the performance of the **monoid** command. Let us consider the deterministic automaton defined by the following table:

	1	2	3	4	5	6	7	8
a	3	1	2	5	4	3	8	7
b	4	1	2	3	4	5	2	8

The transition monoid has 5778 elements. Its computation is achieved in 54 seconds on VAX/780.

¹²The referee has pointed out to us that INR also is highly successful in computing the minimal automaton of such regular expressions.

¹³A new efficient implementation of automata algorithms called AMORE is being developed at the Aachen University; according to [12], its status is still preliminary and we have no elements of comparison with respect to the computation speed.

References

- [1] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, The design and analysis of computer algorithms, *Addison-Wesley*, 1976.
- [2] Alaiwan, H., Algorithmes d'analyse des automates finis et applications aux problèmes de synchronisation, *Thèse de 3ème cycle à l'Université Paris VII*, 1983.
- [3] Beal, M.P., Codage, automates locaux et entropie, *Thèse à l'Université Paris VII, Rapport L.I.T.P. 88-35*, 1988.
- [4] Berstel, J., Transductions and Context-Free Languages, *Stuttgart Teubner*, 1979.
- [5] Champarnaud, J.M., La commande automate, Manuel de l'utilisateur, *Rapport L.I.T.P. 88-44*, 1988.
- [6] Champarnaud, J.M. et G. Hansel, La commande monoïde, Manuel de l'utilisateur, *Rapport L.I.T.P. 88-45bis*, 1988.
- [7] Champarnaud, J.M., AUTOMATE: Un système de manipulation des automates finis, *Rapport L.I.T.P. 85-48*, 1985.
- [8] Cousineau, G., J.F. Perrot and J.M. Rifflet, APL for direct computation of finite semigroups, *Actes du congrès APL 73, North Holland Publ. Comp.*
- [9] Eilenberg, S., Automata, Languages, and Machines, *Academic Press*, 1974.
- [10] Floyd, R. and D. Ullman, The compilation of regular expressions into integrated circuits, *JACM*, **29**, 603–622, 1982.
- [11] Hopcroft, J. E. and J. D. Ullman, Introduction to automata theory, languages and computation, *Addison-Wesley*, 1979.
- [12] Jansen, V., A. Potthoff, W. Thomas, U. Wermuth, A Short Guide to the AMORE System, *Aachener Informatik-Berichte Nr. 90-2*, 1990.
- [13] Johnson, J. H., INR: A Program for Computing Finite Automata, *Unpublished report, University of Waterloo*, 1984.
- [14] Kazman, R., Structuring the Text of the Oxford English Dictionary through Finite State Transduction, *Thesis at the University of Waterloo*, 1986.
- [15] Kleene, S., Representation of events in nerve nets and finite automata, *Automata Studies, C. E. Shannon and J. McCarthy eds., Princeton*, 1956.

- [16] Lallement, G., Semigroups and Combinatorial Applications, *Wiley-Interscience*, 1979.
- [17] Leiss, E., Regpack, an interactive package for regular languages and finite automata, *Research report CS-77-32*, *University of Waterloo*, 1977.
- [18] Lesk, M. E., Lex, A Lexical Analyzer Generator, *Comp. Sci. Tech. Rep. 39*, *Bell Laboratories: Murray Hill, New-Jersey*, 1975.
- [19] McNaughton and Papert, Counter-Free Automata, *MIT Press*, 1972.
- [20] Perrot, J.F., Utilisation d'APL pour calculer des monoïdes finis, *Institut de Programmation de Paris VI*, 1972.
- [21] Pin, J.E., Variétés de langages formels, *Masson*, 1984.
- [22] Pin, J.E., H. Straubing and D. Thérien, New results on the generalized star-height problem, *STACS 89*, **LNCS 349**, 458–467, 1989.
- [23] Simon, I., The Non-deterministic Complexity of a Finite Automaton, *Instituto de Matemática e estatística, Universidade de São Paulo*, *RT-MAP-8703*, 1987.
- [24] Thomson, K., Regular expression search algorithm, *Comm. Assoc. Comput. Mach.*, **11**, 419–422, 1968.
- [25] Weil, P. Inverse monoids of dot-depth two, à paraître in *Theoretical Computer Science*.
- [26] Ziv, J. and A. Lempel, Compression of individual sequences via variable-rate coding, *IEEE Transactions on Information Theory*, volume IT-24, **5**, 530–536, 1978.