Algorithms for computing finite semigroups

Véronique Froidure and Jean-Éric Pin*

Abstract

The aim of this paper is to present algorithms to compute finite semigroups. The semigroup is given by a set of generators taken in a larger semigroup, called the "universe". This universe can be for instance the semigroup of all functions, partial functions, or relations on the set $\{1, \dots, n\}$, or the semigroup of $n \times n$ matrices with entries in a given finite semiring.

The algorithm produces simultaneously a presentation of the semigroup by generators and relations, a confluent rewriting system for this presentation and the Cayley graph of the semigroup. The elements of the semigroup are identified with the reduced words of the rewriting system.

We also give some efficient algorithms to compute the Green relations, the local subsemigroups and the syntactic quasi-order of a subset of the semigroup.

1 Introduction

There are a number of complete and efficient packages for symbolic computation on groups, such as CAYLEY or GAP. Comparatively, the existing packages for semigroups are much less complete. Computers were used for finding the number of distinct semigroups of small order [34, 35, 36, 6, 21, 23, 13, 12, 30, 9] or to solve specific questions on semigroups [19], but the main efforts were devoted to packages dealing with transformation semigroups. Actually, the primary goal of these packages is to manipulate finite state automata and rational expressions, and thus semigroups were uniquely considered as transition semigroups of finite automata. The first such package [5], written in APL, relied on algorithms developed by Perrot [22, 17]. Given a finite deterministic automaton, it produced the list of elements of its transition semigroup and the structure of the regular $\mathcal{D}$-classes, including the Schützenberger group and the sandwich matrix. A much more efficient version, AUTOMATE, was written in C by Champarnaud and Hansel [3]. This interactive package comprised extended functions to manipulate rational languages, but the semigroup part did not include the computation of the Schützenberger groups. Another package, AMORE, was developed in Germany under the direction of W. Thomas (in particular by A. Potthoff) and can be obtained by anonymous ftp at ftp.informatik.uni-kiel.de:pub/kiel/amore/amore.ps.gz. It is comparable to AUTOMATE, since it is written in C and is also primarily designed to manipulate finite automata. However, it includes the computation of all $\mathcal{D}$-classes (regular or not, but without the Schützenberger groups). One can also test whether the transition semigroup is aperiodic, locally trivial, etc. A much less powerful, but convenient

*LITP, CNRS, Université Paris VII, 2 Place Jussieu, 75251 Paris Cedex 05, FRANCE

package was implemented by Sutner [33] using *Mathematica*. Other algorithms to compute Green's relations in finite transformation semigroups were also proposed in [18].

From the semigroup point of view, the main drawback of these packages lies in their original conception. Semigroups are always considered as transformation semigroups, and the algorithms used in these packages heavily rely on this feature. Although similar algorithms were designed for semigroups of boolean matrices by Konieczny [15, 16], no general purpose algorithm was proposed so far in the literature. However, even in theoretical computer science, semigroups do not always occur as transformation semigroups. For instance, semigroups of boolean matrices [24, 7, 8] and more generally semigroups of matrices over commutative semirings [31, 32] occur quite frequently, and therefore there is a strong need for a semigroup package similar to the existing ones on group theory. As a first step towards this goal, we present in this paper a general purpose algorithm to compute finite semigroups. Only a part of this algorithm has been implemented so far, but the first results are quite promising.

2 Definitions

2.1 Semigroups

A *semigroup* is a set equipped with an internal associative operation which is usually written in a multiplicative form. A *monoid* is a semigroup with an identity element (usually denoted by 1). If S is a semigroup, S^1 denotes the monoid equal to S if S has an identity element and to $S \cup \{1\}$ otherwise. In the latter case, the multiplication on S is extended by setting $s1 = 1s = s$ for every $s \in S^1$. The *dual* of a semigroup S is the semigroup $\tilde{S}$ defined on the set S by the multiplication $x \cdot y = yx$. We refer the interested reader to [11, 17, 25, 10, 1] for more details on semigroup theory.

Example 2.1 Let $\mathcal{T}_n$ be the monoid of all functions from $\{1, \ldots, n\}$ into itself under the multiplication defined by $uv = v \circ u$. This monoid is called the monoid of all transformations on $\{1, \ldots, n\}$. A *transformation semigroup* is simply a subsemigroup of some $\mathcal{T}_n$. It is a well-known fact that every finite semigroup is isomorphic to a transformation semigroup. This is the semigroup counterpart of the group theoretic result that every finite group is isomorphic to a permutation group.

Example 2.2 A *semiring* is a set K equipped with two operations, called respectively addition and multiplication, denoted $(s,t) \to s+t$ and $(s,t) \to st$, and an element, denoted 0, such that:

(1) $(K, +, 0)$ is a commutative monoid,

(2) K is a semigroup for the multiplication,

(3) for all $s, t_1, t_2 \in K$, $s(t_1 + t_2) = st_1 + st_2$ and $(t_1 + t_2)s = t_1 s + t_2 s$,

(4) for all $s \in K$, $0s = s0 = 0$.

Thus the only difference with a ring is that inverses with respect to addition may not exist. Given a semiring K, the set $K^{n \times n}$ of $n \times n$ matrices over K is naturally equipped with a structure of semiring. In particular, $K^{n \times n}$ is a monoid under multiplication defined by

$$(rs)_{i,j} = \sum_{1 \leq k \leq n} r_{i,k} s_{k,j}$$

Besides the finite rings, like $\mathbb{Z}/n\mathbb{Z}$, several other finite semirings are commonly used in the literature. We first mention the *boolean semiring* $\mathbb{B} = \{0, 1\}$, defined by the operations $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1 + 1 = 1$ and $1 \cdot 1 = 1$, $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$. Let us also mention the semiring $\mathbb{Z}_n = \{0, 1, \ldots, n\}$, with the operations $\oplus$ and $\otimes$ defined by $s \oplus t = \min\{s + t, n\}$ and $s \otimes t = \min\{st, n\}$. Other examples include the *tropical semiring* $(\mathbb{N} \cup \{\infty\}, \min, +)$ [31].

A relation $\mathcal{R}$ on a semigroup S is *stable on the right* (resp. *left*) if, for every $x, y, z \in S$, $x \mathrel{\mathcal{R}} y$ implies $xz \mathrel{\mathcal{R}} yz$ (resp. $zx \mathrel{\mathcal{R}} zy$). A relation is *stable* if it is stable on the right and on the left. A *congruence* is a stable equivalence relation. Thus, an equivalence relation $\sim$ on S is a congruence if and only if, for every $s, t \in S$ and $x, y \in S^1$, $s \sim t$ implies $xsy \sim xty$. If $\sim$ is a congruence on S, then there is a well-defined multiplication on the quotient set $S/{\sim}$, given by

$$[s][t] = [st]$$

where $[s]$ denotes the $\sim$-class of $s \in S$.

Given two semigroups S and T, a *semigroup morphism* $\varphi : S \to T$ is a map from S into T such that for all $x, y \in S$, $\varphi(xy) = \varphi(x)\varphi(y)$. *Monoid morphisms* are defined analogously, but of course, the condition $\varphi(1) = 1$ is also required.

A semigroup (resp. monoid) S is a *quotient* of a semigroup (resp. monoid) T if there exists a surjective morphism from T onto S. In particular, if $\sim$ is a congruence on a semigroup S, then $S/\sim$ is a quotient of S and the map $\pi : S \to S/{\sim}$ defined by $\pi(s) = [s]$ is a surjective morphism, called the *quotient morphism* associated with $\sim$.

Let S be a semigroup. A *subsemigroup* of S is a subset T of S such that $t, t' \in T$ implies $tt' \in T$. Subsemigroups are closed under intersection. In particular, given a subset A of S, the smallest subsemigroup of S containing A is called the subsemigroup of S *generated* by A. The *Cayley graph* of S (relative to A) is the graph $\Gamma(S, A)$ having S^1 as set of vertices, and for each vertex s and each generator a, an edge labeled by a from s to sa.

An element e of a semigroup S is *idempotent* if $e^2 = e$. If s is an element of a finite semigroup, the subsemigroup generated by s contains a unique idempotent and a unique maximal subgroup, whose identity is the unique idempotent. Thus s has a unique idempotent power, denoted s^ω.

A *zero* is an element 0 such that, for every $s \in S$, $s0 = 0s = 0$. It is a routine exercise to see that there is at most one zero in a semigroup.

2.2 Free semigroups

An *alphabet* is a finite set whose elements are *letters*. A *word* (over the alphabet A) is a finite sequence $u = (a_1, a_2, \ldots, a_n)$ of letters of A. The integer n is the *length* of the word and is denoted $|u|$. In practice, the notation $(a_1, a_2, \ldots, a_n)$

is shortened to $a_1a_2\cdots a_n$. The empty word, which is the unique word of length 0, is denoted by 1. The (concatenation) *product* of two words $u = a_1a_2\cdots a_p$ and $v = b_1b_2\cdots b_q$ is the word $uv = a_1a_2\cdots a_pb_1b_2\cdots b_q$. The product is an associative operation on words. The set of all words on the alphabet A is denoted by A^*. Equipped with the product of words, it is a monoid, with the empty word as an identity. It is in fact the free monoid on the set A. This means that A^* satisfies the following universal property: if $\varphi : A \to M$ is a map from A into a monoid M, there exists a unique monoid morphism from A^* into M that extends φ. This morphism, also denoted φ, is simply defined by $\varphi(a_1\cdots a_n) = \varphi(a_1)\cdots\varphi(a_n)$.

Let Σ be a subset of $A^* \times A^*$, and let $\sim_\Sigma$ be the least congruence $\sim$ on A^* such that $u \sim v$ for every pair $(u, v) \in \Sigma$. The quotient monoid $A^*/{\sim_\Sigma}$ is called the *monoid presented* by (A, Σ). A pair $(u, v) \in \Sigma$ is often denoted $u = v$. For instance, the monoid presented by $(\{a, b\}, ab = ba)$ is isomorphic to $\mathbb{N}^2$, the free commutative monoid on two generators. Presentations of semigroups are defined in the same way. Given a presentation Σ, the *word problem* is to know whether two given words are equivalent modulo $\sim_\Sigma$.

From the algorithmic point of view, presentations are in general intractable. See [14] for a survey. For instance, it is an undecidable problem to know whether a finitely presented semigroup is finite or not. There also exist finite presented semigroups with an undecidable word problem. To avoid these difficulties, we will follow a different approach, that goes back to Sakarovitch [28, 29, 20]. Some definitions are in order to describe it in a precise way.

Let A be a totally ordered alphabet. The *lexicographic order* is the total order used in a dictionary. Formally, it is the order $\leq_{lex}$ on A^* defined by $u \leq_{lex} v$ if and only if u is a prefix of v or $u = pau'$ and $v = pbv'$ for some $p \in A^*$, $a, b \in A$ with $a < b$. In the *military order*, words are ordered by length and words of equal length are ordered according to the lexicographic order. Formally, it is the order $\leq$ on A^* defined by $u \leq v$ if and only if $|u| < |v|$ or $|u| = |v|$ and $u \leq_{lex} v$.

For instance, if $A = \{a, b\}$ with $a < b$, then $ababb \leq_{lex} abba$ but $abba < ababb$. The next proposition summarizes elementary properties of the order $\leq$. The proof is straightforward and omitted.

Proposition 2.1 *Let $u, v \in A^*$ and let $a, b \in A$.*

(1) If $u < v$, then $au < av$ and $ua < va$.

(2) If $ua \leq vb$, then $u \leq v$.

An important consequence of Proposition 2.1 is that $\leq$ is a *stable order* on A^*: if $u \leq v$, then $xuy \leq xvy$ for all $x, y \in A^*$.
A *reduction* is a mapping $\rho : A^* \to A^*$ satisfying the following conditions:

(1) $\rho \circ \rho = \rho$

(2) For all $u \in A^*$ and $a \in A$, $\rho(ua) = \rho(\rho(u)a)$ and $\rho(au) = \rho(a\rho(u))$.

Condition (2) can be extended as follows.

Lemma 2.2 *Let ρ be a reduction on A^*. Then for all $u, v \in A^*$, $\rho(uv) = \rho(\rho(u)v) = \rho(u\rho(v))$.*

Proof. We prove the equality $\rho(uv) = \rho(\rho(u)v)$ by induction on $|v|$. If $v = 1$, the result follows from condition (1). If the result holds for v, then for every letter $a \in A$, the following equalities hold by (2)

$$\rho(uva) = \rho((uv)a) = \rho(\rho(uv)a) = \rho(\rho(\rho(u)v)a) = \rho(\rho(u)va).$$

Similarly the equality $\rho(uv) = \rho(u\rho(v))$ is proved by induction on $|u|$. $\square$

The set $R = \rho(A^*)$ is called the set of *reduced words* for ρ. The next proposition shows how reductions can be used to define monoids.

Proposition 2.3 *Let ρ be a reduction on A^* and let R be the set of its reduced words. Then R, equipped with the multiplication defined by $u\cdot v = \rho(uv)$, is a monoid.*

Proof. If $u, v, w \in A^*$,

$$(u\cdot v)\cdot w = \rho(\rho(uv)w) = \rho(uvw) = \rho(u\rho(vw)) = u\cdot (v\cdot w)$$

and thus the multiplication is associative. We claim that $\rho(1)$ is the identity of the multiplication. If $r \in R$, then $r = \rho(u)$ for some $u \in A^*$. Therefore $r\cdot \rho(1) = \rho(\rho(u)\rho(1)) = \rho(\rho(u)1) = \rho(\rho(u)) = \rho(u) = r$ and similarly, $\rho(1)\cdot r = r$. $\square$

Conversely, given a monoid M generated by a set A, there is a natural morphism $\pi : A^* \to M$ defined by $\pi(a) = a$. Define a mapping $\rho : A^* \to A^*$ by setting, for each $u \in A^*$,

$$\rho(u) = \min\{v \in A^* \mid \pi(v) = \pi(u)\}$$

where the minimum is taken with respect to the military order. Then ρ is a reduction, called the *military reduction*, and the elements of M can be identified with the reduced words for ρ. This reduction also gives a presentation for M.

Theorem 2.4 *The monoid M is presented on A by the set of relations $\{(u = \rho(u)) \mid u \in MA \setminus M\}$.*

Proof. Let $\Sigma = \{(u = \rho(u)) \mid u \in MA \setminus M\}$. First, since $\pi(u) = \pi(\rho(u))$ by definition, M satisfies all relations of Σ and thus $u \sim_\Sigma v$ implies $\pi(u) = \pi(v)$. We claim that $u \sim_\Sigma \rho(u)$ for every word $u \in A^*$. Since $1 \in M$, every word $u \in A^*$ admits a unique factorization of the form $u = p(u)s(u)$ where $p(u)$ is the maximal prefix of u belonging to M. We prove the claim by induction on the length n of $s(u)$. If $n = 0$, then $p(u) = u$, $u = \rho(u)$ and the claim is trivial. Assume the claim holds for n and let u be a word such that $s(u) = a_1 \cdots a_{n+1}$. Then $p(u)a_1 \cdots a_n \sim_\Sigma \rho(p(u)a_1 \cdots a_n)$ by induction, and thus $u \sim_\Sigma \rho(p(u)a_1 \cdots a_n)a_{n+1}$. Now, by definition of Σ, $\rho(p(u)a_1 \cdots a_n)a_{n+1} \sim_\Sigma \rho(\rho(p(u)a_1 \cdots a_n)a_{n+1}) = \rho(u)$. Therefore, $u \sim_\Sigma \rho(u)$, proving the claim. Now $\pi(u) = \pi(v)$ implies $\rho(u) = \rho(v)$ and thus $u \sim_\Sigma v$. Thus Σ is a presentation of M. $\square$

3 The main algorithm

A semigroup S will be given as a subsemigroup of a given semigroup U, called the *universe*, generated by a subset A. This universe can be for instance the semigroup $\mathcal{T}_n$ or the semigroup of n by n matrices over a given semiring. We require the following information on the universe:

- the type of the elements (arrays, matrices over a semiring, etc.),
- an algorithm to compute the product of two elements of U,
- an algorithm to test equality of two elements of U,
- the set of generators A.

Given a subset A of a universe U, our main algorithm computes the submonoid of U generated by A. It is a little simpler to deal with monoids, so this point of view will be adopted in this presentation, but it is fairly easy to modify our algorithm to obtain the semigroup generated by A. The result of our computation can be formalized as follows:

Input : A universe U, a subset A of U and a total order on A.

Output : The military reduction $\rho : A^* \to M$ defining the submonoid M of U generated by A, the list of elements of M (sorted in military order) and the Cayley graphs $\Gamma(M, A)$ and $\Gamma(\tilde{M}, A)$.

3.1 A simplified version

We first present a simplified version of our algorithm, which just produces the sorted list of elements of M and the rewriting system. As was explained above, the elements of M are identified with reduced words of A^*. To each element $u \in M$ is associated its value $\nu(u)$ in the universe U.

The following pseudocode is our basic algorithm. The set of generators is given as a totally ordered set $A = \{a_1 < \ldots < a_k\}$. The first element of the list of elements is the empty word 1. The successor of the word u in the list is denoted $Next(u)$. The variable $Last$ denotes the last element of the current list.

Let $u := 1$ and $Last := 1$.
while true
    for $i := 1$ to k,
        compute $\nu(ua_i)$;
        if $\nu(ua_i)$ is new
            $Next(Last) := ua_i$;
            $Last := ua_i$;
        else if $\nu(ua_i) = \nu(u')$ for some $u' < ua_i$, produce the rule $ua_i \to u'$;
    if u has a successor, $u := Next(u)$
    else break;

The algorithm works as follows. For each element u of the list being completed and for each generator $a \in A$, the value ua is computed. If this value is the value of some element u' already in the list, a rule $ua \to u'$ is produced. Otherwise, a new element ua is created. The main properties of our algorithm are given in the following proposition.

Proposition 3.1 *The list of elements of M produced by the algorithm is sorted for the military order and the rules are all of the form $u \to v$ with $v < u$.*

Proof. A state of the program is determined by the values of the triple $(u, Last, i)$ at the beginning of the **for** loop. In a given state $(u, Last, i)$, the output is a list of the form $(1, \ldots, u, \ldots, Last)$. We claim that the interval $(u, \ldots, Last)$ is sorted for the military order and that $Last < ua_i$. This property is trivially satisfied at the initial state $(1, 1, 1)$. Passing from state $(u, Last, i)$ to state $(u, Last, i+1)$ (resp. $(u, ua_i, i+1)$) leaves the property invariant, since $ua_i < ua_{i+1}$. Passing from state $(u, Last, k)$ to state $(Next(u), Last, 1)$ (resp. $(Next(u), ua_k, 1)$) also leaves the property invariant. Indeed the interval $(Next(u), \ldots, Last)$ is a subinterval of $(u, \ldots, Last)$ and $Last < ua_k$. Thus $(u, \ldots, Last)$ (resp. $(u, \ldots, Last, ua_k)$) is sorted. Furthermore, $u < Next(u) \leq Last < ua_k$ by assumption. Therefore, either $|Next(u)| = |Last| = |u| + 1$ and then $Last < ua_k < Next(u)a_1$, or $|Next(u)| = |u|$ and $Last < ua_k < Next(u)a_1$, since $u < Next(u)$. This proves the claim and shows that the output is sorted. The second part of the proposition is clear. □

We illustrate our algorithm on an example.

Example 3.1 Let $U = T_6$ and let $A = \{a, b\}$ be the set of generators given in the following table

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

We first calculate the value of the empty word 1 and of the words $1a = a$ and $1b = b$. Next the value of aa is equal to the value of a. This produces the rule $aa \to a$. The values of ab, ba and bb are new. Next, we calculate in this order the values of aba, abb, baa, bab, bba and bbb. The first value is new, but the other ones are not and produce the following rules: $abb \to aba$, $baa \to ba$, $bab \to bb$, $bba \to bb$ and $bbb \to bb$. Therefore, aba is the unique element of length 3 created at this step. It remains to calculate the values of $abaa$ and $abab$, which give the rules $abaa \to aba$ and $abab \to aba$. Finally, the elements of the monoid are represented in the following table

	1	2	3	4	5	6
1	1	2	3	4	5	6
a	2	2	4	4	5	6
b	5	3	4	4	6	6
ab	3	3	4	4	6	6
ba	5	4	4	4	6	6
bb	6	4	4	4	6	6
aba	4	4	4	4	6	6

and the rewriting rules are

$$\begin{array}{llllllll} aa \to a & abb \to aba & baa \to ba & bab \to bb \\ bba \to bb & bbb \to bb & abaa \to aba & abab \to aba \end{array}$$

3.2 The extended version

Some computations are redundant in this algorithm. For instance, in the previous example, the computation of baa could have been avoided, since the rule $aa \to a$ infers $baa \to ba$. This example is generic. Let u be an element of M, and let $u = bs$, where b is the first letter of u. If, for some generator $a \in A$, the word sa is not reduced, the word ua will not be reduced. Furthermore, if $sa \to r$, then $\rho(ua) = \rho(bsa) = \rho(br)$.

Example 3.2 Applying this improvement on the previous example would reduce the set of rules to the following set

$$aa \to a \qquad abb \to aba \qquad bab \to bb \qquad bba \to bb \qquad bbb \to bb$$

Thus $M = (\{a, b\} \mid aa = a, abb = aba, bab = bb, bba = bb, bbb = bb)$

However, the computation of $\rho(ua)$ requires the knowledge of $\rho(br)$. Therefore, in order to use the suggested improvement, it is necessary to compute $\rho(au)$ for every word u and every generator a. In other words, a simultaneous computation of the Cayley graphs $\Gamma(M, A)$ and $\Gamma(\tilde{M}, A)$ is required and it is not clear anymore whether the improvement is not compensated by the extra amount of computation needed for $\Gamma(\tilde{M}, A)$.

By modifying sligthly our algorithm, we can get around this difficulty. The main trick is to organize the computation by length. For each value of n, we compute the products $\rho(ua)$ for $|u| = n$ and $a \in A$. At this stage, the improvement can be applied : if $u = bs$ and $sa \to r$ for some generator $a \in A$, then $\rho(ua) = \rho(br)$. If $r = 1$, then $\rho(ua) = b$. Otherwise, let $r = tc$, with $c \in A$. Then $t \leq s$ by Proposition 2.1 and if $t = s$, then $c < a$. Thus if $t = s$, $\rho(ua) = \rho(uc)$ and the computation of $\rho(uc)$ has been done, since $c < a$. Now if $t < s$, then $|t| \leq |s| < |u|$ and thus the computation of $\rho(bt)$ has been done and furthermore $\rho(bt) \leq bt < bs = u$. Therefore, the computation of $\rho(\rho(bt)c)$ has been done and $\rho(ua) = \rho(\rho(bt)c)$.

Once all the products $\rho(ua)$ are known for all words u of length n, a simple observation leads to the computation of all $\rho(au)$ for $|u| = n$. Indeed, if $u = pb$, and $a \in A$, $\rho(au) = \rho(tb)$, where $t = \rho(ap)$. Since $|t| \leq |u|$, the computation of $\rho(tb)$ has been done at this stage.

It is interesting to compute the precise number of calls to the procedure procedure *Product* that computes the product of two elements in the universe U.. Let R be the set of relations generated by the program.

Theorem 3.2 *The number of calls to the procedure Product is equal to* $|M| + |R| - |A| - 1$.

Proof. First note that *Product* is only called during the computation of $\Gamma(M, A)$. Let $u = bs$. The calls to *Product* during the computation of $\rho(ua)$ occur when sa is reduced. Then, either ua is a new element, or a new rule $ua \to \rho(ua)$ is produced. Now all rules and all elements of M (except for the identity and the generators, which are given) are produced in this way. This gives the required formula. $\square$

Another advantage of this algorithm will become apparent in the next sections. Indeed, the computation of $\Gamma(\tilde{M}, A)$ is actually needed to calculate the

Green relations, the local subsemigroups and the syntactic quasi orders (see the next sections).

A description of the data structure used to represent elements of M is in order. For a non-empty word u, denote by $f(u)$ (resp. $\ell(u)$) its first (resp. last) letters and by $p(u)$ (resp. $s(u)$) its prefix (resp. suffix) of length $|u|-1$. Each element u of M (recall that u is a reduced word of A^*) is represented by a pointer on the following data:

(1) The value $\nu(u)$ (an element of U).

(2) The letters $f(u)$ and $\ell(u)$.

(3) The addresses of $p(u)$ and $s(u)$.

(4) The address `Next` of the successor of u, the minimal word of the set

$$\{v \in A^* \mid v \text{ is reduced and } u < v\}$$

(5) For each generator $a \in A$, the address of $\rho(ua)$ and a flag to indicate whether ua is reduced.

(6) For each generator $a \in A$, the address of $\rho(au)$.

(7) The length of u.

Note that the word u itself is not stored in this data structure, but can be easily retrieved, since we know its first letter and the address of its prefix $p(u)$. The $|M|$ addresses are stored in a sufficiently large table T, of size at least $\frac{10}{9}|M|s$, where s is the size of an address (in practice a size of $5|M|s$ was used). Fast access is ensured through open addressing using a standard double hashing technique [4, pp. 235–237]. Two hash functions $h, h' : U \to \mathbb{N}$ are given. Let $v \in U$ be an element to be searched in the table T. The slots $T[h(v)]$, $T[h(v)+h'(v)]$, $T[h(v)+2h'(v)]$, $T[h(v)+3h'(v)]$, $\ldots$ are probed in this order and their values are compared to v. The search terminates successfully if the value v is found and terminates unsuccessfully if an empty slot is found. In the latter case, v is stored in the empty slot.

This data structure also gives a representation of the Cayley graphs $\Gamma(M, A)$ and $\Gamma(\tilde{M}, A)$. Indeed, the addresses of $\rho(ua)$ and $\rho(au)$ are stored for each element $u \in M$ and each generator $a \in A$.

The number of relations, the number of elements and the maximal length of the reduced words are stored in global variables. Initially, all these variables are set to 0. We now give the details of the algorithm.

Initialization. The data corresponding to the empty word 1 are filled. The value $\nu(1)$ is the identity of U. The fields corresponding to $f(u)$, $\ell(u)$, $p(u)$ and $s(u)$ are irrelevant. The successor of 1 is the letter a_1. For each generator a_i, the value $\nu(a_i)$ is computed. If $\nu(a_i) = \nu(a_j)$ for some $j < i$, the generator a_i is eliminated, and the rule $a_i \to a_j$ is created. Similarly, if $\nu(a_i) = 1$, the identity of U, the generator a_i is eliminated, and the rule $a_i \to 1$ is created. Otherwise, the value $\nu(a_i)$ is stored, its address is given to the field $\rho(a_i)$ of 1 and the flag is set to "reduced". The fields $f(a_i)$, $\ell(a_i)$, $p(a_i)$, $s(a_i)$ and $Next$ are also filled. The variables giving the number of relations and the number of elements are updated.

Without lost of generality, we may suppose that no generator has been eliminated during this initialization. Thus the list of elements is $(1, a_1, \ldots, a_k)$. The pseudocode of the main loop is listed below. The variable u represents the current word and v represents the minimal word of the current length.

```
let u := a_1, v := u and Last := a_k;
let CurrentLength := 1;
repeat
  while Length(u) = CurrentLength    (Computation of ua_i)
  {
     let b := f(u), s := s(u);
     for i := 1 to k
     {
        if sa_i is not reduced
        {
           let r := ρ(sa_i);        (note that r < sa_i)
           if r = 1        (empty word)
              ρ(ua_i) := b
           else
              ρ(ua_i) := ρ(ρ(bt)c) where c := ℓ(r), t := p(r);
        }
        else
        {
           compute ν(ua_i) and search this value in the table;
           if ν(ua_i) = ν(u') for some u' < ua_i, produce the rule ua_i → u';
           else
              T[Last].Next := ua_i;
              f(ua_i) := b; ℓ(ua_i) := a_i; p(ua_i) := u; s(ua_i) := s(u)a_i;
              Length(ua_i) = Length(u) + 1;
              Last := ua_i;
        }
     }
     u := T[u].Next;
  }
  u := v;
  while Length(u) = CurrentLength    (Computation of a_iu)
  {
     let p := p(u);
     for i := 1 to k
        ρ(a_iu) := ρ(ρ(a_ip)ℓ(u));
     u := T[u].Next;
  }
  v := u;
  CurrentLength := CurrentLength + 1;
until u = Last;
```

Example 3.3 Let U the semigroup of 2×2 matrices with entries in $\mathbb{Z}_3$ and let

$A = \{a, b\}$, where

$$a = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \qquad b = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$$

The semigroup S generated by A contains 11 elements.

$$\begin{array}{rclrclrclrcl}
a &=& \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} & b &=& \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix} & aa &=& \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix} & ab &=& \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix} \\
ba &=& \begin{pmatrix} 3 & 1 \\ 3 & 2 \end{pmatrix} & bb &=& \begin{pmatrix} 1 & 3 \\ 0 & 3 \end{pmatrix} & aab &=& \begin{pmatrix} 1 & 1 \\ 3 & 3 \end{pmatrix} & aba &=& \begin{pmatrix} 3 & 1 \\ 3 & 3 \end{pmatrix} \\
abb &=& \begin{pmatrix} 1 & 3 \\ 2 & 3 \end{pmatrix} & bab &=& \begin{pmatrix} 3 & 3 \\ 3 & 3 \end{pmatrix} & aabb &=& \begin{pmatrix} 1 & 3 \\ 3 & 3 \end{pmatrix} & & &
\end{array}$$

Our algorithm produces the following rewriting rules

$$\begin{array}{rclrclrclrcl}
aaa &\to& aa & baa &\to& ba & bba &\to& bab & bbb &\to& bb \\
aaba &\to& aba & abab &\to& bab & baba &\to& bab & babb &\to& bab
\end{array}$$

Note that bab is a zero of S (it is easy to modify our algorithm to search for a zero). Thus $S = \langle \{a, b\} \mid aaa = aa, baa = ba, bba = 0, bbb = bb, aaba = aba, abab = 0 \rangle$.

Compared to the first version, the advantage of this algorithm is to avoid a number of computations inside the semigroup U.

4 Green relations

Green's relations on a semigroup S are defined as follows [17, 25]. If s and t are elements of S, we set

$s \,\mathcal{L}\, t$	if there exist $x, y \in S^1$ such that $s = xt$ and $t = ys$,
$s \,\mathcal{R}\, t$	if there exist $x, y \in S^1$ such that $s = tx$ and $t = sy$,
$s \,\mathcal{J}\, t$	if there exist $x, y, u, v \in S^1$ such that $s = xty$ and $t = usv$.
$s \,\mathcal{H}\, t$	if $s \,\mathcal{R}\, t$ and $s \,\mathcal{L}\, t$.

For finite semigroups, there is a convenient representation of the corresponding equivalence classes. The elements of a given $\mathcal{R}$-class (resp. $\mathcal{L}$-class) are represented in a row (resp. column). The intersection of an $\mathcal{R}$-class and an $\mathcal{L}$-class is an $\mathcal{H}$-class. Each $\mathcal{J}$-class is a union of $\mathcal{R}$-classes (and also of $\mathcal{L}$-classes). It is not obvious to see that this representation is consistent: it relies in particular on the fact that, in finite semigroups, the relations $\mathcal{R}$ and $\mathcal{L}$ commute. The presence of an idempotent in an $\mathcal{H}$-class is indicated by a star. One can show that each $\mathcal{H}$-class containing an idempotent e is a subsemigroup of S, which is in fact a group with identity e. Furthermore, all $\mathcal{R}$-classes (resp. $\mathcal{L}$-classes) of a given $\mathcal{J}$-class have the same number of elements.

$^*a_1, a_2$	$^*a_3, a_4$	a_5, a_6
b_1, b_2	$^*b_3, b_4$	$^*b_5, b_6$

A $\mathcal{J}$-class.

In this figure, each row is an $\mathcal{R}$-class and each column is an $\mathcal{L}$-class. There are 6 $\mathcal{H}$-classes and 4 idempotents. Each idempotent is the identity of a group of order 2.
A $\mathcal{J}$-class containing an idempotent is called *regular*. One can show that in a regular $\mathcal{J}$-class, every $\mathcal{R}$-class and every $\mathcal{L}$-class contains an idempotent.

The computation of the $\mathcal{R}$-classes is fairly easy. It follows from the observation that the $\mathcal{R}$-classes of a semigroup S generated by A are the strongly connected components of the Cayley graph $\Gamma(S, A)$. The $\mathcal{L}$-classes can be computed in a similar way from the graph $\Gamma(\tilde{S}, A)$ corresponding to the left action of A on S. This fact is actually used in AMORE to compute the non regular $\mathcal{R}$-classes. Since our algorithm computes the graphs $\Gamma(S, A)$ and $\Gamma(\tilde{S}, A)$, Tarjan's algorithm [37] can now be used to compute its strongly connected components.

5 Local subsemigroups

If e is an idempotent of a finite semigroup S, the set

$$eSe = \{ese \mid s \in s\}$$

is a subsemigroup of S, called the *local subsemigroup* associated with e. This semigroup is in fact a monoid, since e is an identity in eSe. Local semigroups play an important role in the classification of rational languages [26]. Their computation is based on the following elementary lemma:

Lemma 5.1 *For every idempotent e, $eSe = eS \cap Se$.*

Proof. The inclusion $eSe \subset eS \cap Se$ is clear. For the opposite inclusion, let $s \in eS \cap Se$. Then $s = er = te$ for some $r, t \in S$. Therefore $ese = e(er)e = ere = tee = te = s$ and thus $s \in eSe$. $\square$

This gives a simple way to compute the local semigroup associated with e. Indeed, eS (resp. Se) is simply the set of vertices reachable from e in the graph $\Gamma(S, A)$ (resp. $\Gamma(\tilde{S}, A)$). Again this computation can be achieved by standard algorithms [4].

6 Syntactic quasi orders

Let P be a subset of a monoid M. The *syntactic quasi-order* of P is the quasi-order $\leq_P$ on M defined by $u \leq_P v$ if and only if, for every $x, y \in M$,

$$xvy \in P \Longrightarrow xuy \in P$$

The associated congruence $\sim_P$, defined by $u \sim_P v$ if and only if $u \leq_P v$ and $v \leq_P u$, is called the *syntactic congruence* of P. The quotient semigroup $S(P) = S/{\sim_P}$ is called the *syntactic semigroup* of P. See [27] for more details.

The computation of $\leq_P$ can be achieved as follows. Consider the graph G whose vertices are the pairs $(u, v) \in M \times M$ and the edges are of the form $(ua, va) \to (u, v)$ or $(au, av) \to (u, v)$, for some $a \in A$. This graph has $|M|^2$ vertices and at most $2|A||M|^2$ edges. Observe that for every $u, v \in M$, $u \not\leq_P v$ if and only if there exist $x, y \in M$ such that $xuy \notin P$ and $xvy \in P$. In other words, $u \not\leq_P v$ if and only if the vertex (u, v) is reachable in G from a vertex in $\bar{P} \times P$ (where $\bar{P}$ denotes the complement of P). Therefore, the computation of $\leq_P$ can be reduced to standard graph algorithms as follows:

(1) First compute the graph G. This is easy from the knowledge of $\Gamma(S, A)$ and $\Gamma(\tilde{S}, A)$.

(2) Label each vertex (u, v) as follows:

$$\begin{cases} (0,1) & \text{if } u \notin P \text{ and } v \in P \\ (1,0) & \text{if } u \in P \text{ and } v \notin P \\ (1,1) & \text{otherwise} \end{cases}$$

(3) Do a depth first search in G (starting from each vertex labeled by $(0, 1)$) and set to 0 the first component of the label of all visited vertices.

(4) Do a depth first search in G (starting from each vertex labeled by $(0, 0)$ or $(1, 0)$) and set to 0 the second component of the label of all visited vertices.

(5) The label of each vertex now encodes the syntactic quasi-order of P in the following way:

$$\begin{cases} (1,1) & \text{if } u \sim_P v \\ (1,0) & \text{if } u \leq_P v \\ (0,1) & \text{if } v \leq_P u \\ (0,0) & \text{if } u \text{ and } v \text{ are incomparable} \end{cases}$$

7 Experimental results

It is well-known that the monoid $\mathcal{T}_n$ is generated, for $n \geq 3$, by the set $A = \{a, b, c\}$, where $a = \begin{pmatrix} 1 & 2 & 3 & \dots & n-1 & n \\ 2 & 3 & 4 & \dots & n & 1 \end{pmatrix}$, $b = \begin{pmatrix} 1 & 2 & 3 & \dots & n-1 & n \\ 2 & 1 & 3 & \dots & n-1 & n \end{pmatrix}$ and $c = \begin{pmatrix} 1 & 2 & 3 & \dots & n-1 & n \\ 1 & 2 & 3 & \dots & n-1 & 1 \end{pmatrix}$ For $3 \leq n \leq 7$, the following table gives the number of elements and the number of relations defining $\mathcal{T}_n$ over A^*. The last line gives the number of calls to the function Product

n	3	4	5	6	7
Number of elements	27	256	3125	46656	823543
Number of relations	13	83	751	7935	102592
Calls to Product	36	335	3872	54587	926131

In particular, although the multiplication table of $\mathcal{T}_7$ has 678223072849 entries, less than one million calls to the procedure Product were actually used to compute this monoid. Some performances on a PowerMac 7500/100 are given in the next table:

Type	Generators	Elements	Time
Upper-triangular boolean matrices of size 5	16	32768	$3.73s$
Unitriangular boolean matrices of size 6	16	32768	$5.51s$
$\mathcal{T}_6$	3	46656	$1.21s$
Symmetric group on 8 elements	2	40320	$0.95s$
Order preserving maps on $\{1..9\}$	10	48620	$2.40s$

The semigroup generated by the matrices $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ over $Z/59Z$ (205320 elements) was computed in 8.95s. The computation required 29 megabytes of memory.
The computation of $\mathcal{T}_7$ (823543 elements) required 110 megabytes of memory on a Sun Sparc 10000.

8 Conclusion

We have given several algorithms to compute finite semigroups. Contrary to most of the algorithms used in existing packages, our algorithms do not assume that semigroups are given as transformation semigroups. Furthermore, the number of calls to the procedure Product has been minimized.

9 Acknowledgements

We would like to thank warmly our colleagues of the university of São Paulo Arnaldo Mandel, Alair Pereira Do Lago and Imre Simon, whose numerous suggestions greatly improved our algorithms and programs. In particular, Arnaldo Mandel gave the key suggestion that lead to Theorem 3.2.

References

[1] J. Almeida, *Finite semigroups and universal algebra*, Series in Algebra Vol 3, Word Scientific, Singapore, (1994).

[2] J. J. Cannon, Computing the ideal structure of finite semigroups, *Numer. Math.* **18**, (1971), 254–266.

[3] J.M. Champarnaud and G. Hansel, AUTOMATE, a computing package for automata and finite semigroups, *J. Symbolic Computation* **12**, (1991), 197–220.

[4] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, MIT Press and McGraw-Hill, (1990).

[5] G. Cousineau, J. F. Perrot and J. M. Rifflet, APL programs for direct computation of a finite semigroup, *APL Congres 73, Amsterdam, North Holand Publishing Co.*, (1973), 67–74.

[6] G.E. Forsythe, SWAC computes 126 distinct semigroups of order 4, *Proc. Amer. Math. Soc.* **6** (1955), 443–447.

[7] V. Froidure, *Rangs des relations binaires et Semigroupes de relations non ambigus*, Thèse, Univ. Paris 6, France, (1995).

[8] V. Froidure, Ranks of binary relations, *Semigroup Forum*, to appear.

[9] P.A. Grillet, Computing finite commutative semigroups, *Semigroup Forum*, **53**, (1996), 140–154.

[10] P.M. Higgins, *Techniques of Semigroup Theory*, Oxford Univ. Press, (1992).

[11] J.M. Howie, *An Introduction to Semigroup Theory*, Academic Press, London, (1976).

[12] H. Jürgensen, Computers in semigroups, *Semigroup Forum*, **15**, (1977), 1–20.

[13] H. Jürgensen and P. Wick, Die Halbgruppen der Ordnungen ≤ 7, *Semigroup Forum*, **14**, (1977), 69–79.

[14] O.G. Kharlampovich and M.V. Sapir, Algorithmic problems on varieties, *International Journal of Algebra and Computation* **5**, (1995) 379–602.

[15] J. Konieczny, *Semigroups of Binary Relations*, Ph.D. Thesis, State Univ. of Pennsylvania (1992).

[16] J. Konieczny, Green's equivalences in finite semigroups of binary relations, *Semigroup Forum* **48**, (1994), 235–252.

[17] G. Lallement, *Semigroups and Combinatorial Applications*, John Wiley & Sons, New York, (1979).

[18] G. Lallement and R. McFadden, On the determination of Green's relations in finite transformation semigroups, *J. Symbolic Comput.* **10**, (1990), 481–498.

[19] E. Lusk and R. McFadden, Using Automated Reasoning Tools; A study of the Semigroup F_2B_2, *Semigroup Forum* **36**, (1987), 75–87.

[20] M. Pelletier and J. Sakarovitch, Easy multiplications II. Extensions of Rational Semigroups, *Information and Computation/* **88**, (1990), 18–59.

[21] T.S. Motzkin and J.L. Selfridge, Semigroups of order five, *Bull. Amer. Math. Soc.* **62**, (1956), 14.

[22] J. F. Perrot, *Contribution à l'étude des monoïdes syntactiques et de certains groupes associés aux automates finis*, Thèse de doctorat, Univ. de Paris, France, (1972).

[23] R. J. Plemmons, Cayley tables for all semigroups of order ≤ 6, Auburn Univ., (1966).

[24] R. J. Plemmons and M. T. West, On the semigroup of binary relations, *Pacific Jour. of Math.* **35**, (1970), 743–753.

[25] J.-E. Pin, *Variétés de langages formels*, Masson, Paris, (1984). English translation: *Varieties of formal languages*, Plenum, New-York, (1986).

[26] J.-E. Pin, Finite semigroups and recognizable languages : an introduction, in NATO Advanced Study Institute *Semigroups, Formal Languages and Groups*, J. Fountain (ed.), Kluwer academic publishers, (1995), 1–32.

[27] J.-E. Pin, A variety theorem without complementation, *Izvestiya VUZ Matematika* **39**, (1995), 80–90. English version, *Russian Mathem. (Iz. VUZ)* **39**, (1995), 74–83.

[28] J. Sakarovitch, Description des monoïdes de type fini, *Elektronische Informationsverarbeitung und Kybernetik EIK* **17** (1981), 417–434.

[29] J. Sakarovitch, Easy multiplications I. The Realm of Kleene's Theorem, *Information and Computation/* **74**, (1987), 173–197.

[30] S. Sato, K. Yama and M. Tokizawa, Semigroups of order 8, *Semigroup Forum* **49**, (1994), 7–29.

[31] I. Simon, On semigroups of matrices over the tropical semiring, *Informatique Théorique et Applications* **28**, (1994), 277–294.

[32] H. Straubing, The Burnside problem for semigroups of matrices, in *Combinatorics on Words, Progress and Perspectives, L.J. Cummings (ed.)*, Acad. Press, (1983), 279–295.

[33] K. Sutner, Finite State Machines and Syntactic Semigroups, *The Mathematica Journal* **2**, (1991), 78–87.

[34] T. Tamura, Some remarks on semigroups and all types of order 2, 3, *J. Gakugei Tokushima Univ.* **3**, (1953), 1–11.

[35] T. Tamura, Notes on finite semigroups and determination of semigroups of order 4, *J. Gakugei Tokushima Univ.* **5**, (1954), 17–27.

[36] K. Tetsuya, T. Hashimoto, T. Akazawa, R. Shibata, T. Inui and T. Tamura, All semigroups of order 5, *J. Gakugei Tokushima Univ.* **6**, (1955), 19–39 and Erratum.

[37] R. E. Tarjan, Depth first search and linear graph algorithms, *SIAM Journal of Computing* **1** (1972) 146–160.