ENUMERATING WORDS IN FINITELY PRESENTED MONOIDS

Miguel A. Borges-Trenard and Hebert Pérez-Rosés[1], Department of Mathematics, Faculty of Sciences, University of Oriente, Santiago de Cuba, Cuba

ABSTRACT
An efficient algorithm is given, to compute the order of a finitely presented group or monoid by enumerating all the elements in a suitable sequence, so as to save space. It is shown that only one element of the group or monoid needs to be kept in memory at each iteration.

Key words: Graphs, efficient algorithm.

AMS: 05AIS

RESUMEN
Un algoritmo eficiente es presentado, para computar el orden de un grupo dado o monoide mediante la enumeración de todos los elementos en una sucesión adecuada, para salvar espacio. Es demostrado que solo un elemento del grupo o monoide necesita ser mantenido en la memoria en cada iteración.

Palabras clave: gráficos, algoritmo eficiente.

INTRODUCTION

In this paper we give a space-efficient algorithm to enumerate the elements of a monoid (or group) *M* that is given by a finite set of generators and defining relations; in other words, we want to compute the order *M* using as little memory space of the computer as possible. We show that for doing that, it suffices to keep only one word in memory, so that the maximum space needed is roughly proportional to the length of the maximal irreducible word.

If *M* is a group, a standard method to compute its order is coset enumeration. If we know the order of a nontrivial subgroup *H* of *M*, generated by a set of words in the generators of *M*, then we can compute the index of *H* in *M* by means of the classical Todd-Coxeter coset enumeration procedure [6,2]. If, on the other hand, a nontrivial subgroup *H* is not available, then we could also perform the same computation with the trivial subgroup; in this case, enumerating the cosets is equivalent to enumerating all group elements. This method has been employed by several Computer Algebra systems, like GAP, for instance. In principle, it can be applied to any group presentation, but it has the disadvantage that it requires a relatively high amount of memory space to store a table of cosets.

If the presentation defining *M* posesses certain nice properties (completeness), there is a better algorithm to compute its order, which was proposed by Gilman in 1979 [4]. This algorithm is based on the enumeration of all paths from a single source in a special graph constructed from the input presentation; each path corresponds to an irreducible word in the generators, which, in turn corresponds to one, and only one element of *M* (here, *M* could be either a monoid or a group). Gilman's method still requires space to store the graph, and some additional space to keep track of the paths traversed; but that spaces is usually less than what is needed for the coset table.

Our method is a simple modification of Gilman's; we arrange the irreducible words in a tree structure, and then we perform a backtracking or depth-first traversal of that tree. The tree of irreducible words is just a virtual tree, a conceptual model: there is no need to store it in memory in order to traverse it. The traditional depth-first traversal algorithm would certainly need to keep a record of the path that is being traversed in order to be able to backtrack; i.e. it would need to store a sequence of words; however, in the way that we have defined our tree, that record-keeping process can be done by just one word.

[1]Email: mborges@csd.uo.edu.cu
hebert@csd.uo.edu.cu

This paper is organized as follow: In the first section we give an overview of string-rewriting concepts applied to group presentations. In the second section we show how to construct Gilman's graph and review a couple of basic facts related to it. Section 3 is devoted to our tree of irreducible words and the space-efficient algorithm to traverse it.

1. COMPLETE PRESENTATIONS

Let T be a finite set of symbols; we shall denote by (T) the free monoid generated by T. The identity element (the empty word) will be denoted as 1. If we have a binary relation R on $\langle T \rangle$, then $\langle R \rangle$ will stand for the congruence generated by R, i.e. the reflexive-symmetric-transitive closure of R, which is compatible with the concatenation operation. To make it simpler, we shall write the pairs of words $(\alpha,\beta) \in R$, as rewriting rules of the form $\alpha \rightarrow \beta$, or equalities of the form $\alpha = \beta$, and $\overline{\alpha}$ will be the equivalence class of α modulo $\langle R \rangle$. In $\alpha \rightarrow \beta$, α is the **left-hand side**, or **head** of the rule, and β is its **right-hand side**, or tail.

Let T^{-1} be the set of inverse symbols of T:

$T^{-1} = \{x^{-1} | x \in T\}$, and $I = \{xx^{-1} = x^{-1}x = 1 | x \in T\}$.

The set I is a binary relation on $\langle T \cup T^{-1} \rangle$ and the quotient monoid

$$F = \frac{\langle T \cup T^{-1} \rangle}{\langle I \rangle}$$

is the free group generated by T. Now, if R is any binary relation on $\langle T \cup T^{-1} \rangle$, then it is said that $\langle T; R \rangle$ is a presentation of the group

$$G = \frac{\langle T \cup T^{-1} \rangle}{\langle R \cup I \rangle},$$

or any group canonically isomorphic to G.

Clearly, if we let $T := T \cup T^{-1}$, and $R := R \cup I$, then we have a monoid presentation, so we can restrict ourselves to presentations of this kind.

If $(\alpha,\beta) \in R$, and v,w are arbitrary words of $\langle T \rangle$, then we say that $v\,\alpha w$ reduces or rewrites to $v\,\beta w$ (with respect to R), and we also write $v\,\alpha w \rightarrow v\,\beta w$. In this case, the word $v\,\alpha w$ is called reducible. On the other hand, if the word δ does not contain any left-hand side of a rule as a subword, δ is said to be irreducible or reduced. Notice that $\rightarrow$ now stands for a new binary relation R' on $\langle T \rangle$, such that $(v\,\alpha w, v\,\beta w) \in R'$ iff $(\alpha,\beta) \in R$).

In general, two reduced words may be in the same class modulo $\langle R \rangle$, which means that they represent the same element of the monoid. That is not the case if we have a complete presentation, defined below.

We write $\overset{*}{\rightarrow}$ for the reflexive transitive closure of R ' and we say that the words α,β are joinable (denoted $\alpha \downarrow \beta$) if there exists a word γ such that $\alpha \overset{*}{\rightarrow} \gamma$ and $\beta \overset{*}{\rightarrow} \gamma$. R is called terminating, well-founded or noetherian if there is no infinite sequence $\alpha_1 \rightarrow \alpha_2 \rightarrow ... \rightarrow_n \rightarrow ...$; R is called confluent if $\alpha \overset{*}{\rightarrow} \beta$ and $\alpha \overset{*}{\rightarrow} \gamma$ implies $\beta \downarrow \gamma$, and it is called locally confluent if $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ implies $\beta \downarrow \gamma$. If R is terminating then local confluence amounts to confluence (Diamond Lemma or Newman's Lemma).

R is said to have the Church-Rosser property if $\alpha \equiv \beta \pmod{\langle R \rangle}$ implies $\alpha \downarrow \beta$. If R is both terminating and Church-Rosser, it is said to be complete, canonical or convergent. Completeness guarantees the existence of a unique irreducible word in each equivalence class modulo $\langle R \rangle$, so that the number of irreducible words corresponds exactly to the order of the group; for an arbitrary presentation, the number of irreducible words is an upper bound of the order of the group [4]. This upper bound is not tight, in general. As an example, take the presentation $\langle a,b,c; a^2 = b^2 = c^2 = abc \rangle$, of the quaternion group, of order 8; the set of irreducible words associated with this presentation is infinite.

From now on, *Irr*(*R*) will stand for the set of irreducible words modulo *R*. *R* is called normalized if for every rule $\alpha \to \beta$ of *R*, $\beta \in Irr(R)$, and $\alpha \in Irr(R \setminus \{\alpha \to \beta\})$. The underlying idea is that normalized presentations do not contain relations that are obviously redundant.

2. GILMAN'S GRAPH

A useful tool for studying the monoid presented by $\langle T ; R \rangle$ is Gilman's graph [4], denoted $\Gamma(R)$, whose construction we are ready to describe now. Let $H = \{\alpha \in \langle T \rangle \mid (\alpha,\beta) \in R\}$ (the set of rule heads); the vertices of $\Gamma(R)$ are the proper prefixes of these rule heads (including 1). For any two vertices w_1 and w_2, and for $x \in T$, there is an arc going from w_1 to w_2, labeled x, if, and only if, $w_1x \in Irr(R)$, and one of the following conditions holds.

i) $w_2 = w_1x$, or else

ii) w_2 is the longest suffix of w_1x that is a vertex of $\Gamma(R)$.

A path in $\Gamma(R)$ starting at 1 defines a word in $\langle T \rangle$ in a straightforward manner: by succesively adjoining the labels of the arcs traversed. $\Gamma(R)$ can be easily transformed into a non-deterministic finite automaton A (*R*), that accepts precisely the set *Irr* (*R*). As usual, vertices will represent states of the automaton, and labeled arcs represent transitions between states; a new state, $\clubsuit \notin \langle T \rangle$, is added, as well as an arc (w, ♣), labeled x, for every vertex w and every $x \in T$ such that wx is reducible. We also add an arc (♣, ♣), labeled x, for every $x \in T$. All the states of A(*R)* , except ♣, are final states (i.e. accepting states), whereas 1 is the initial state.

2.1 Theorem. The set accepted by A(R) is *Irr* (*R*) (hence, *Irr* (*R*) is regular)

Proof. [1] (lemma 2.1.3). □

Given this automaton, there is an effective procedure to decide whether *Irr* (*R*) is empty, finite, or infinite; as shown by the following well-known theorem from the theory of languages and automata.

2.2 Theorem. Let *A* be a finite automaton with *N* states. The set accepted by *A* is

i) non empty iff A accepts some word of lenth $\ell < N$.

ii) infinite iff A accepts some word of length ℓ, with $N \le \ell \le 2N$.

Proof. [5] (theorem 3.7). □

From this theorem and the previous construction, Gilman's proposition 3(a) [4] follows as a corollary.

2.3 Corollary. The number of irreducible words determined by *R* is finite iff $\Gamma(R)$ does not have any directed cycle. If it is finite, the number of irreducible words is equal to the number of paths in $\Gamma(R)$ starting at 1.

Proof. If $\Gamma(R)$ has a directed cycle, *A*(*R*) will obviously accept words of any prescribed length. Now, if the language accepted by *A*(*R*) is infinite, then, by Theorem 2.2. *A*(*R*) accepts a word of length at least *N*. The accepting path of this word must include at least *N* nodes, none of which is ♣; hence, this path must form a cycle in $\Gamma(R)$, for $\Gamma(R)$ has *N* - 1 vertices. The rest is obvious. □

The existence of cycles in $\Gamma(R)$ can be easily checked in *O* (*N* + *E*) steps, where *E* is the number of arcs, $E \le N|T|$. The number *N* depends not only on the number of rules and the lengths of the rule heads, but on the rules themselves as well. Let $n = |T|$ and $M = |H|$; and let a_i be the length of the i-th rule head α_i, for i = 1,2,...,it is very straightforward to derive the following upper bound for *N*

2.4 Proposition. $N \le \sum_{i=1}^{M} a_i - 2M + n + 1$.

Proof. The number of non-empty (not necesarily different) proper head prefixes is $N \leq \sum_{i=1}^{M}(a_i - 1)$. plus the empty word; but the number of length-one prefixes is at most n, which makes $\sum_{i=1}^{M}(a_i - 2) + n + 1$. □

Now, let *L* denote the length of the longest rule head, and *f* and *g* be two integer-valued functions describing the complexity of testing set membership, and adding a new element to a set, respectively. Given a vertex v of $\Gamma(R)$, and $x \in T$, such that vx is known to be irreducible, the construction of the arc starting at v and labeled x can be done in at most $O(Lf(N)+g(E))$ steps. To decide that $vx \in Irr(R)$, requires up to $O(LM)$ extra steps. Hence, we have

2.5 Proposition. The construction of $\Gamma(R)$ may be done in no more than $O(N|T|(LM + Lf(N)+ g(E)))$ steps.

3. THE DEGLEX TREE OF IRREDUCIBLE WORDS

Again, $\langle T;R \rangle$ is a finite presentation of the monoid G, with $|T| = n$; let us assume that there is a linear ordering among the generators: $x_1 < x_2 < ... < x_n$. We construct the deglex tree of irreducible words modulo R, $\tau(R)$, as follows. The nodes of the tree are words in the alphabet *T*, and the root of the tree is the empty word 1. Let w be an arbitrary node of $\tau(R)$, and consider the set of words $\{wx: x \in T\}$; from this set of words we take the subset C_{w_1} consisting of the irreducible ones: $C_w = \{wx_{i1}, wx_{i_2},..., wx_{i_r}\}$ (where $1 \leq i_1 < i_2 < ... < i_r \leq n$). The words in C_w will be the children of w in our tree of irreducible words.

We call wx_{i_1} the leftmost child of w, and wx_{i_k}, consequently, its rightmost child. More generally, if $i_r < i_s$ then we say that wx_{i_s} is at the left of wx_{i_s}. A path (or branch) in the tree is a sequence $w_0 = 1$, w_1, w_2, ..., where w_i is a child of w_{i-1}. The leftmost path of the tree is the sequence $w_0 = 1, w_1 = x_1, w_2,...$, where w_i is the leftmost child of w_{i-1}; the right-most path is defined in an obvious manner. The length of the finite path $w_0 = 1$, w_1, w_2, ...,w_t is ℓ.

The set C_w may be empty for some nodes w; we call such a w a leaf, otherwise it is an inner node. The root of the tree is on level zero, and the generators x_i are on level one of the tree; if w is an irreducible word of length ℓ, then it is on level ℓ of the tree, and conversely. If w_1 and w_2 are two words on the same level, that are the children of v_1 and v_2 respectively ($v_1 \neq v_2$), then we say that w_1 is at the left of w_2 if v_1 is at the left of v_2. If we traverse the nodes of the tree starting at level 0, then 1,2,... etc., and each level from left to right, then we are just enumerating the irreducible words in degree lexicographic order (whence the name deglex tree); algorithm DEGLEX-ENUMERATE works that way. This level traversal (or breadth-first traversal) is, however, an inefficient way to carry out the enumeration, because we must keep a whole level in memory in order to compute the next one.

A good way to reduce the memory requirements is to traverse the tree in a backtracking or depth-first (DFS) fashion, which, in our case, is equivalent to enumerating the irreducible words in pure lexicographic ordering. A recursive definition of backtracking could be the following: given a tree with root x, first output x; then take the first child of x, say y_1 and search the whole subtree with root y_1 using the same bactracking procedure; afterwards, proceed with the second child y_2, etc. In practice, the procedure follows the leftmost path up to the end, then backtracks to the nearest crossroads and takes the second path up to its end, and so on.

The implementation of this procedure requires that we be able to backtrack, hence we have to keep a record of the path that has been traversed so far.

Input**:** <*T;R*>, and *N*.
Output**:** The words of *Irr*(*R*) in degree lexicographic order.

```
i: = 0; (* indicates current level *)
L:= {1}; (* set of irreducible words on level i*)
c: = 1; (* word count *)
repeat
      i :=i+1;
      L':=Ø;
      for w ∈ L do
          for j:=1 to n do
              if w . x_j ∈ Irr(R ) then begin
                 Output w . x_j;
                 L':= L' ∪ {w . x_j} ;
                 c:=c+1;
                 end;
      L:=L';
until L = Ø or i = N ;
```

Figure 1: Algorithm DEGLEX ENUMERATE

In our case, a key observation is that, given an irreducible word w, we can reconstruct the path that leads to w from the root 1 (it is just the sequence of prefixes of w); hence, we do not have to store the whole path in memory: it suffices to have w. Algorithm, LEX-ENUMERATE is a non-recursive formulation of the above procedure.

So, there is a one to-one correspondence between paths in $\Gamma(R)$ starting at 1, irreducible words modulo R, and paths in the deglex tree $\tau(R)$. This correspondence implies that if $Irr(R)$ is infinite, then there is at least on cycle in $\Gamma(R)$, and consequently, at least one infinite branch of $\tau(R)$ with a "cyclic" structure, i.e. the words on that branch have the form $\alpha\beta\beta$ ...β δ, where $\alpha,\beta,\delta \in <T>$, and δ is a prefix of β. To conclude that $Irr(R)$ is infinite, it suffices to find either such a cyclic branch or a word with length greater than or equal to N. LEX-ENUMERATE halts after finding a word of length N, or it stops normally when it tries to backtrack past the empty word 1, which means that the whole tree has been searched. In the latter case, the variable c contains the value $|Irr(R)|$.

The algorithm could work with any upper bound of N as input, instead of its exact value, which is more difficult to calculate. As we said before, the maximum space needed is proportional to the maximum word length (that is, N).

```
Input: <T;R >, and N.
Output: The words of Irr(R ) in lexicographic order.

w:=1; (* current irreducible word *)
Output 1;
c:=1; (* word count *)
i:=1; (* an index in the set T of generators *)
repeat
   Find the smallest j such that w. x_j ∈ Irr(R );
   if it exists then
      if length (w)=N – 1 then
           halt (*Irr(R ) is infinite *)
      else begin
            w:=x.x_j;
           Output w;
           c:=c+1;
            i:=1;
           end

   else
      if w= 1 then
         return c
    else
         Delete the last letter, x_j, of w,
          and let i:=j + 1;
until false;
```

Figure 2: Algorithm LEX-ENUMERATE.

REFERENCES

[1] BOOK, R.V. and O. FRIEDRICH (1993): **String Rewriting Systems**. Texts and Monographs in Computer Science, Berlin: Springer-Verlag.

[2] CANNON, J.J.; L.A. DIMINO; G. HAVAS and J.M. WATSON (1973): "Implementation and Analysis of the Todd-Coxeter Algorithm". **Mathematics of Computation 27**, 463-490.

[3] FUST, M.L.; J. HOPCROFT and E.M LUKS (1980): "Polynomial Time algorithms for Permutation Groups". **Procs. of the 21st IEEE Symp. on the Foundations of Computer Science**, 36-41.

[4] GILMAN, R.H. (1979): "Presentations of Groups and Monoids". **Journal of Algebra 57**, 544-554.

[5] HOPCROFT, J. and J.D. ULLMAN (1979): **Introduction to Automata Theory, Languages and Computation**. Reading: Addison-Wesley.

[6] TODD, J.A. and H.S.M. COXETER (1936): "A Practical Method for Enumerating Cosets of a Finite Abstract Group". **Procedings of the Edinburgh Mathematical Society** (2) 5, 25-34.