J. Symbolic Computation (1998) **11**, 1–000

Computing automatic coset systems and subgroup presentations

DEREK F. HOLT AND DARREN F. HURT

Mathematics Institute, University of Warwick, Coventry CV4 7AL, U.K.

(Received 23 April 1998)

The concept of an automatic group can be generalized to a group that is automatic with respect to a specified subgroup. This means that there is a finite state automaton that recognizes a unique word in each coset of the subgroup, and others that essentially recognize the permutation action on these cosets induced by multiplying by a group generator. These automata make it possible to enumerate coset representatives as words in the generators, and to solve the generalized word problem for the subgroup efficiently. Algorithms to construct these automata have been described previously by Redfern. Here we describe improved versions, together with implementation details and some examples of successful calculations. A related algorithm to compute a finite presentation of the subgroup is also described.

1. Introduction

The concept of an automatic coset system was introduced by Ian Redfern in his PhD Thesis (Redfern, 1993). Such a system consists of a group G with finite monoid generating set A, a subgroup H, and finite state automata W (the coset word acceptor) and M_x (the coset multipliers), where there is one M_x for each $x \in A$, and one for $x = \varepsilon$ (which denotes the empty word or string). The alphabet of W is A, and that of the M_x is $A^\dagger \times A^\dagger$, where $A^\dagger = A \cup \{\$\}$, and \$ is used to pad the shorter of the two words being read, to make them of equal length. Let A^* denote the set of words in A and, for each $w \in A^*$, let $\overline{w}$ be its image in G. The defining properties of an automatic coset system are that, for each right coset Hg of H in G, W accepts at least one word w with $\overline{w} \in Hg$, and each M_x accepts a pair of words $(w_1, w_2) \in A^* \times A^*$ if and only if w_1 and w_2 are both accepted by W, and $H\overline{w_1 x} = H\overline{w_2}$. In particular, if $H = 1$, then we get the usual definition of an automatic structure for G.

If such an automatic coset system exists, then we say that G is *coset automatic* with respect to the subgroup H. Redfern shows in his thesis that this property is independent of the generating set A. (The proof is a straightforward adaptation of that for automatic groups (the case $H = 1$), given in Section 2.4 of Epstein, Cannon, Holt, Levy, Paterson and Thurston, 1992.) One reason that automatic coset systems might be useful in practice is that they can be used to solve the *generalized word problem* for H in G (that is, to decide whether an arbitrary word w in the generators of G lies in H) in time quadratic in the length of w. Again the proof carries over essentially without change from the corresponding result for $H = 1$, which is of course just the word problem for G. This is Theorem 2.3.10 of (Epstein, Cannon, Holt, Levy, Paterson and Thurston, 1992).

Apart from these theorems, very few theoretical results have been proved about coset automatic groups. This is in sharp distinction to the situation for automatic groups, for which a substantial and continually growing theory has been developed. Since G is always coset automatic with respect to itself, we cannot say anything about G in full generality, but one might hope to prove results about the quotient group of G modulo the core of H in G (that is, the largest normal subsgroup of G that is contained in H). Is it true that this quotient is necessarily finitely presented, for example?

A *shortlex* automatic coset system is one in which W accepts precisely one word w for each coset Hg, and this is the least word w in a shortlex ordering of A^* such that $\overline{w} \in Hg$. (In a shortlex ordering, $w < x$ if either w is shorter than x, or w and x have equal lengths and w precedes x in the lexicographical ordering of A^* induced by a specified total ordering of A.) The property of being shortlex coset automatic can depend on the choice of A and on the specified ordering of A; indeed this is the case even when $H = 1$ (see Section 3.5 of Epstein, Cannon, Holt, Levy, Paterson and Thurston, 1992 for an example).

The bulk of Redfern's thesis is devoted to a description of an algorithm for calculating shortlex automatic coset systems, and to some applications of an implementation of this algorithm (by Redfern himself) to drawing the limit sets of Kleinian groups. (This application is also described in McShane, Parker and Redfern 1994.) He assumes that G is a finitely presented group and that H is a finitely generated subgroup, and the algorithm takes a finite presentation of G together with generators of H as words in the generators of G as input. In principle, it should succeed eventually, subject to these assumptions, whenever G is coset automatic with respect to H. A serious weakness of his algorithm, however, is that he is unable to prove formally that the coset systems he computes are correct.

In this paper, we describe an alternative algorithm, in which we can prove correctness, and with the extra feature that we can compute a finite presentation of H whenever the main algorithm succeeds. It is not so general as Redfern's, in that it requires extra assumptions (to be discussed later) for it to succeed. On the other hand, it can be proved that it always succeeds in the important case of quasiconvex subgroups of word-hyperbolic groups (see, for example, Gersten and Short, 1994), and we have not found any specific examples in which Redfern's algorithm succeeds but ours does not. Furthermore, it seems to be considerably faster than Redfern's in the examples used for the applications to Kleinian groups.

The theory of this algorithm is also described in Chapter 3 of the PhD Thesis of the second author (Hurt, 1996). It has been implemented by the first author as part of his package KBMAG for the Knuth-Bendix algorithm on monoids, and the calculation of automatic structures. This is freely available by anonymous **ftp**; please e-mail the first author for details.

We shall assume for the remainder of the paper that G is a group defined by a finite monoid presentation on a fixed finite ordered monoid generating set A, with relator set R. We assume also that A is closed under inversion; that is, for each $x \in A$, there is an $x' \in A$ such that xx' and $x'x$ are in R. Then the words in A^* are totally ordered by the shortlex ordering with respect to the specified ordering of A, and whenever we write $u < v$, etc., for $u, v \in A^*$, we shall be using this ordering. The image of a word u in G is denoted by $\bar{u}$. A word u is called *irreducible* if it is the least word that maps onto $\bar{u}$. We assume further that a finite generating set Y of a subgroup H of G is given, where the elements of Y are words in A^*. We shall say that a word $u \in A^*$ is *H-irreducible* if it is the least word of which the image in G lies in the right coset $H\bar{u}$. We denote the identity element of G by 1 and the empty word in A^* by ε.

For the general theory of automatic groups, the reader should consult (Epstein, Cannon, Holt, Levy, Paterson and Thurston, 1992). The algorithms for shortlex automatic groups, of which the methods to be presented in this paper are generalizations, are described in (Epstein, Holt, Rees, 1991) and (Holt, 1996). Our notation, and the basic structure of the algorithm descriptions will follow those in (Holt, 1996) fairly closely. We shall assume that the reader has some familiarity with automatic groups and their associated machinery, such as finite state automata, and the use of the padding symbol \$ in two-variable automata. As in the earlier papers, if u and v are words in A^*, then we use $(u,v)^\dagger$ to denote the pair of words in which the shorter of u and v (if any) has had the padding symbol appended to its end to make the two words have equal length. Then $(u,v)^\dagger$ can be regarded as a word in $A^\dagger \times A^\dagger$, which is the input language of the multiplier automata. In general, a finite state automaton with input alphabet $A^\dagger \times A^\dagger$ will be referred to as a *two-variable automaton over A*. We shall call it a *padded* automaton if it accepts only words of the form $(u,v)^\dagger$ for $u, v \in A^*$. When considering images in G, \$ will always map onto the identity element, 1.

If $u, v \in A^*$, then we shall use $u =_G v$ to mean that u and v are equal as elements of G (i.e. $\bar{u} = \bar{v}$), whereas $u = v$ will mean that u and v are equal as words. Similarly, we shall use $Hu =_G Hv$ for the coset equality $H\bar{u} = H\bar{v}$. We denote the length of $u \in A^*$ by $l(u)$ and, for $t \geq 0$, we define $u(t)$ to be the prefix of u of length t for $t \leq l(u)$, and $u(t) = u$ for $t \geq l(u)$. The length $l(g)$ of an element $g \in G$ is defined to be the length of the shortest word $u \in A^*$ with $\bar{u} = g$.

We need to extend the notion of the *word-differences* associated with an equation in G, introduced in (Epstein, Holt, Rees, 1991) and (Holt, 1996), to a coset equation. Suppose that u and v are words in A^* satisfying $u =_G hv$ for some $h \in H$. We define the set of word-differences associated with this equation to be the set of elements $\overline{u(t)}^{-1}\overline{hv(t)}$ of G, for all $t \geq 0$. (Since $u(t)$ and $v(t)$ are ultimately constant, this set is finite.) In particular, the element $h \in H$ lies in the word-difference set. The set of word-differences associated with a set of such equations is the union of the sets associated with the individual equations.

The principal extra assumption on which our algorithms depend is the following. There is a global positive constant K (depending only on the fixed input data G, A, and Y), such that whenever $u, v \in A^*$ are H-irreducible, $x \in A$, and $ux =_G hv$ for some $h \in H$, then the word-differences associated with this equation have length at most K. (Equivalently, there is a finite set D of elements of G, such that all word-differences of all coset equations of this type lie in D.) We shall refer to this condition as the *coset fellow traveller* property, and we shall prove in the next section that it implies that G is coset automatic with respect to H.

In practice, our programs will not work unless G itself is shortlex automatic. This is not such a big restriction as it may sound, since it is probable that most or all interesting examples have this property anyway. It is proved by Redfern in Chapter 10 of (Redfern, 1993) that both of these assumptions hold for quasiconvex subgroups of word-hyperbolic groups, and in practice they hold for many other examples. The programs can verify the correctness of any automatic coset system that they calculate, and so if this process is successful, then we know that the example is shortlex coset automatic, independently of any general theoretical results.

Chapter 1 of (Epstein, Cannon, Holt, Levy, Paterson and Thurston, 1992) is probably the best reference for general results about finite state automata (fsa) in the context of automatic group theory. (Note that the abbreviation fsa will be used for both the singular and the plural.) Two types of fsa will be used in this paper. The first is the standard *partial deterministic automaton* (pda) (cf Definition 1.2.5 of Epstein, Cannon, Holt, Levy, Paterson and Thurston, 1992), in which there is at most one initial state, no epsilon transitions, and at most one transition from a given state with a given alphabet symbol. The second, which we shall call a *multiple initial-state partial deterministic automaton* (mipda), is the same except that there may be more than one initial state. Of course, for any mipda, one can construct an equivalent pda accepting the same language. This construction is theoretically less expensive than the general determinization process for a non-deterministic automaton, since the state-set of the equivalent pda consists of subsets of the original state-set of size at most the number of initial states, as opposed to all subsets in the general case. In general, for an fsa M, $\mathcal{S}(M)$ denotes the state-set of M, and $\mathcal{A}(M)$ the set of accept states. The set of initial states will be denoted by $\mathcal{I}(M)$, and $\iota(M)$ will be the unique initial state of a deterministic automaton. The image of the transition (if any) from a state τ, with label x is denoted by τ^x, and $L(M)$ denotes the language accepted by M.

Mipda will arise in the following context, which is a direct generalization of similar constructions of (deterministic) word-difference automata that are described in (Epstein, Holt, Rees, 1991) and (Holt, 1996). In general, a *word-difference automaton* WD for G is defined to be a two-variable pda or mipda over A for which there is a map $\alpha : \mathcal{S}(WD) \to G$ such that, for all $x, y \in A^\dagger$ and all $\tau \in \mathcal{S}(WD)$ for which $\tau^{(x,y)}$ is defined, we have $\alpha(\tau^{(x,y)}) = \overline{x}^{-1}\alpha(\tau)\overline{y}$.

Let D be the set of word-differences associated with a set of coset equations $u =_G hv$ as above, and suppose that D is finite. For such an equation $u =_G hv$, let $u = x_1 \ldots x_n$, and $v = y_1 \ldots y_n$, where $x_i, y_i \in A^\dagger$, and the shorter of u and v has been padded by \$ symbols to make their lengths equal. Then we define an associated word-difference mipda WD, in which D is the state-set and α is the identity map as follows. The initial states of WD are 1 and the elements $h \in H$ that occur in the equations, the (unique) accept state of WD is 1, and the transitions are

$$\overline{u(i-1)}^{-1}\,\overline{hv(i-1)}^{(x_i,y_i)} = \overline{u(i)}^{-1}\,\overline{hv(i)},$$

for $1 \leq i \leq n$, for each of the equations $u =_G hv$ under consideration. It is also technically convenient to include the transitions $1^{(x,x)} = 1$ for each $x \in A$. This automaton has the property that it accepts $(u,v)^\dagger$, for each of the equations $u =_G hv$. Conversely, for any padded pair of words $(u,v)^\dagger$ that WD accepts, we have $u =_G hv$ for one of the initial states h. We shall call WD the *basic* mipda associated with the set of coset equations.

We can also extend WD as follows, without disturbing any the properties mentioned so far. For all $x \in A$, we adjoin the states $\bar{x}$ to $\mathcal{S}(WD)$ if they are not there already. Then, for all $\tau \in \mathcal{S}(WD)$, and (x,y) in $A^\dagger \times A^\dagger \setminus \{(\$,\$)\}$, we adjoin the transition $\tau^{(x,y)} = \bar{x}^{-1}\tau\bar{y}$ whenever $\bar{x}^{-1}\tau\bar{y} \in \mathcal{S}(WD)$. We call the resulting fsa the *extended* mipda associated with the set of coset equations.

In practice, during the course of the algorithms, we will not necessarily be able to reduce words $w \in A^*$ to the irreducible representatives of the corresponding elements

$\overline{w} \in G$. This means that the states of our word-difference automata will not actually be elements of G but will be words in A^*, and α will be the natural map onto G.

As in (Holt, 1996), if X and Y are two fsa with the same input alphabet, then we denote by $X \cdot Y$, X', $X \wedge Y$ and $X \vee Y$, fsa with accepted languages $L(X)L(Y)$, $A^* \setminus L(X)$, $L(X) \cap L(Y)$ and $L(X) \cup L(Y)$, respectively. If Z is a two-variable fsa over A then there is an fsa $E(Z)$, with input alphabet A, that accepts $u \in A^*$ if and only if there exists $v \in A^*$ with $(u,v)^\dagger \in L(Z)$. There are fairly straightforward algorithms for effectively constructing each of these new automata (and they are all provided in the KBMAG package, both as functions and as standalone programs). See (Epstein, Cannon, Holt, Levy, Paterson and Thurston, 1992) for details of these constructions. Again following (Holt, 1996), we denote by GT ("greater than") a two-variable automaton which, for $u, v \in A^*$, accepts $(u,v)^\dagger$ if and only if $u > v$ in our fixed shortlex ordering on A^*.

The minimization procedure for finite state automata plays an important rôle in the implementation of our algorithms. This is a procedure which, given a pda X constructs a pda Y with a minimal number of states such that $L(Y) = L(X)$. (See, for example, Chapter 4.13 of (Aho et. al., 1974) for a description of this procedure.) In fact, it generalizes easily to automata with more than the standard two categories of states (accepting and non-accepting). In our implementation, we have the possibility of defining a finite set L of labels, and attaching a label in L to some of the states in a pda or mipda X over an alphabet A. In fact L is the (not necessarily disjoint) union of L_I and L_A, the initial and accepting labels. We can construct a pda or mipda Y with the same labelling set, and a minimal number of states subject to the property that, for any word w in A^* and any $l_i \in L_I, l_a \in L_A$, there is a path for w in Y beginning at a state with label l_i and ending at a state with label l_a if and only if there is such a path in X. In the applications described in this paper, X and Y will be pda if $L_I \leq 1$ and mipda if $L_I > 1$, and the distinct initial states will always have distinct labels.

The paper is organized as follows. In Sections 2 and 3, we prove the theoretical results on which the algorithms for computing, respectively, automatic coset systems, and subgroup presentations depend. In Section 4, we describe our algorithm for computing and verifying the correctness of automatic coset systems, and in Section 5, we comment on some of the issues involved in their implementation. In Section 6, we describe our algorithm for computing a presentation of the subgroup H, once the automatic coset system is known. Our current implementation computes this presentation on a set of (monoid) generators of H that is, so to speak, chosen by the computer rather than by the user. More precisely, this is the set of elements $h \in H$ that arise in the coset equations $ux =_G hv$, for H-irreducible words $u, v \in A^*$ and $x \in A$. It should not be difficult to calculate an alternative presentation on the generators Y of H provided by the user, and we describe briefly how this could be done, but we have not yet implemented this. In the final section, we discuss the performance and results of the programs on some specific examples.

2. Theoretical results on automatic coset systems

We start by proving the result mentioned in the introduction that the coset fellow traveller property guarantees the existence of a shortlex automatic coset system.

THEOREM 2.1. *Suppose that there is a finite set D of elements of G with the property that, whenever $u, v \in A^*$ are H-irreducible, $x \in A$, and $ux =_G hv$ with $h \in H$, then* *the word-differences associated with this equation all lie in D. Then G is shortlex coset automatic with respect to H.*

Proof. Let WD be the extended mipda (as defined in Section 1) associated with the set of all coset equations $ux =_G hv$ with u, v H-irreducible and $x \in A$. For any H-reducible word $w \in A^*$, if u is its longest H-irreducible prefix and ux its shortest H-reducible prefix, then $(ux, v)^\dagger \in L(WD)$ for some H-irreducible word $v \in A^*$. By definition of H-irreducible, we have $ux > v$, and so $(ux, v)^\dagger$ is accepted by the fsa $WD \wedge GT$, and $w \in L(E(WD \wedge GT) \cdot A^*)$. Conversely, if $w \in L(E(WD \wedge GT) \cdot A^*)$, then $(ux, v)^\dagger$ is accepted by $WD \wedge GT$ for some prefix ux of w and some $v \in A^*$, and therefore $ux > v$, and $ux =_G hv$ for some initial state h of WD. Since these initial states are all elements of H, this implies that ux is H-reducible, and therefore w is H-reducible.

We have shown that $E(WD \wedge GT) \cdot A^*$ accepts precisely the set of H-reducible words in A^*, and so its complement $W := (E(WD \wedge GT) \cdot A^*)'$ accepts the set of H-irreducible words, and is the correct word-acceptor for the automatic coset system.

The multiplier automata M_x for $x \in A$ are constructed in much the same way as for automatic groups (see Definition 2.3.3 of Epstein, Cannon, Holt, Levy, Paterson and Thurston, 1992, Epstein, Holt, Rees, 1991 or Holt, 1996). (Note that, since W accepts a unique word for each coset, we must have $L(M_\varepsilon) = \{(w, w) | w \in L(W)\}$.) We first make a slight modification to W, and define a pda $W^\dagger$ which is identical to W, except that its input language is $A^\dagger$ rather than A, and it has an extra state τ, say, that is accepting, and transitions $\sigma^\$ = \tau$ for all $\sigma \in \mathcal{A}(W^\dagger)$ (including $\sigma = \tau$). The mipda M_x is defined to have state set $\mathcal{S}(W^\dagger) \times \mathcal{S}(W^\dagger) \times \mathcal{S}(WD)$. The initial states are $(\iota(W^\dagger), \iota(W^\dagger), h)$ for $h \in \mathcal{I}(WD)$. For $x, y \in A^\dagger$ and $(\sigma_1, \sigma_2, \rho) \in \mathcal{S}(M_x)$, the transition $(\sigma_1, \sigma_2, \rho)^{(x,y)}$ is defined to be equal to $(\sigma_1^x, \sigma_2^y, \rho^{(x,y)})$ provided all three components are defined, and undefined otherwise. The M_x for different x differ from each other only in their accept states; the accept states of M_x are all $(\sigma_1, \sigma_2, \overline{x})$ such that $\sigma_1, \sigma_2 \in \mathcal{A}(W^\dagger)$, and at most one of σ_1 and σ_2 is equal to τ. (Our definition of WD as the extended mipda associated with a set of coset equations ensures that $\overline{x} \in \mathcal{S}(WD)$ for all $x \in A$.) Note that our definition of $W^\dagger$ ensures that M_x is a padded two-variable mipda; that is, it accepts only padded words of the form $(u, v)^\dagger$ for $u, v \in A^*$.

It is clear from the construction that, if M_x accepts $(u, v)^\dagger$ for $u, v \in A^*$, then u and v are accepted by W, and $ux =_G hv$ for some initial state h of WD, and so $Hux =_G Hv$. Conversely, suppose that $u, v \in L(W)$ and $Hux =_G Hv$. Then u and v are H-irreducible and $ux =_G hv$ for some $h \in H$ so, by assumption, $(ux, v)^\dagger$ is accepted by WD, by a path beginning with the initial state h. Since we are using the shortlex ordering we must have $l(v) \leq l(ux)$. Let $l(ux) = n$, and $v' = v(n-1)$. Then, when WD reads the word $(ux, v)^\dagger$ starting with the state h, let g be the state reached after reading the maximal proper prefix (u, v'). Then, either $l(v) < l(ux)$, in which case $v = v'$ and $g = \bar{x}$, or $l(v) = l(ux)$, $v = v'y$ for some $y \in A$, and $g = \overline{xy}^{-1}$. In the latter case, by the definition of the extended mipda, WD will have a transition $g^{(\$, y)} = \bar{x}$. In either case, starting from the state h of WD, we reach the state $\bar{x}$ after reading $(u, v)^\dagger$. It follows that M_x accepts $(u, v)^\dagger$, and so the word-acceptor W and the multipliers M_x fulfil the conditions for a automatic coset system, and the proof is complete. □

Note that the initial states of each multiplier automaton M_x constructed in the above proof can be labelled by distinct elements $h \in H$, and that if $(u, v)^\dagger$ is accepted by M_x by a path beginning with an initial state labelled h, then $ux =_G hv$ in G. This fact will be important in the construction of the subgroup presentation of H, which will have this set of state-labels as its (monoid) generating set.

Our next result is used to justify the correctness verification (axiom checking) procedure that we use on candidates for our automatic coset system that we construct in the algorithm to be described in Section 3. It is a fairly straightforward generalization of Theorem 2.5 in (Epstein, Holt, Rees, 1991). We recall that G is a finitely presented group with monoid presentation $\langle A|R\rangle$, and H is a subgroup generated by the set Y of words in A^*.

THEOREM 2.2. *Let W and $M_x(x \in A)$ be finite state automata satisfying the following conditions :*

(i) W has input alphabet A, and each M_x is a two-variable fsa *over A.*
(ii) If $(v,w)^\dagger \in L(M_x)$ for some $x \in A$, then $v, w \in L(W)$ and $Hvx =_G Hw$.
(iii) $L(W)$ is nonempty and, if $x_1 \dots x_n \in L(W)$ (where $x_i \in A$), then $x_1 \dots x_{n-1} \in L(W)$ and $(x_1 \dots x_{n-1}\$, x_1 \dots x_n) \in L(M_{x_n})$.
(iv) Let $w(= w_0) \in L(W)$ and $x_1 \dots x_n \in R$. Then for any $w_n \in L(W)$, $w_0 = w_n$ if and only if there exist words $w_1, \dots, w_{n-1} \in L(W)$ for which $(w_{i-1}, w_i)^\dagger \in L(M_{x_i})$ for $1 \le i \le n$.
(v) Let $x_1 \dots x_n \in Y$. Then for any $w_n \in L(W)$, $w_n = \varepsilon$ if and only if there exist $w_1, \dots, w_{n-1} \in L(W)$ with $(\varepsilon, w_1)^\dagger \in L(M_{x_1}), \dots, (w_{n-1}, w_n)^\dagger \in L(M_{x_n})$ (by (iii), $L(W)$ is nonempty and is prefix closed, so $\varepsilon \in L(W)$).

Then G is coset automatic with respect to H, and W and the M_x form an automatic coset system in which $L(W)$ is prefix closed, and each right coset of H in G has a unique representative in $L(W)$.

Conversely, if the automata W, M_x form an automatic coset system for G with respect to H, and $L(W)$ is prefix closed with unique representatives for each right coset, then W, M_x satisfy conditions $(i) - (v)$.

REMARK: Since $L(W)$ accepts a unique word for each coset of H in G, the equality multiplier M_ε of the automatic coset system, which we have not mentioned explicitly in the statement of the theorem, has language $\{(w,w)|w \in L(W)\}$.

PROOF. Suppose that W and M_x satisfy the conditions (i)-(v). As stated in the introduction, we are assuming throughout that A is inverse closed; that is, that for each $x \in A$, there is an $x' \in A$ such that $xx', x'x \in R$. Let $w \in L(W)$. Then, given $x \in A$, by (iv), there exists $w' \in L(W)$, with $(w,w')^\dagger \in L(M_x)$ and $(w',w)^\dagger \in L(M_{x'})$. Let w'' be any element of $L(W)$ with $(w,w'')^\dagger \in L(M_x)$. Then we have $(w',w)^\dagger \in L(M_{x'})$ and $(w,w'')^\dagger \in L(M_x)$. But $x'x \in R$. Hence by (iv), we must have $w'' = w'$. Thus, given $w \in L(W)$, there exists a unique $w' \in L(W)$ such that $(w,w')^\dagger \in L(M_x)$. Hence we have a well-defined map $\mu(x) : L(W) \to L(W)$, such that $w\mu(x) = w'$. Clearly, from the above, we have $w'\mu(x') = w$, and consequently $\mu(x') = \mu(x)^{-1}$, and $\mu(x)$ is a permutation of $L(W)$. We can immediately extend μ to a monoid homomorphism $\mu : A^* \to Sym(L(W))$ (the symmetric group on $L(W)$).

Let $x_1 \dots x_n \in R$ (with $x_i \in A$) and $w \in L(W)$. Then $w\mu(x_1 \dots x_n) = w\mu(x_1) \dots \mu(x_n)$. Let $w_i = w\mu(x_1 \dots x_i)$. Then $(w_{i-1}, w_i)^\dagger \in L(M_{x_i})$ and $w_n = w\mu(x_1 \dots x_n)$. From (iv), we deduce that $w_n = w(= w_0)$. Thus $\mu(r)$ is the identity permutation for all $r \in R$.

Consequently μ induces a well defined homomorphism $\bar{\mu} : G \to Sym(L(W))$. By (v), we have $\varepsilon\mu(w) = \varepsilon$ for all $w \in Y$, and so $\varepsilon\bar{\mu}(h) = \varepsilon$ for all $h \in H$.

Let Ω be the set of right cosets of H in G, and define a map $\alpha : L(W) \to \Omega$ by $\alpha(w) = H\overline{w}$. Suppose $Hg \in \Omega$, and let $x_1 \ldots x_n \in A^*$, with $x_1 \ldots x_n =_G g$. Let $v_i = \varepsilon\mu(x_1 \ldots x_i)$. Then $(v_{i-1}, v_i)^\dagger \in L(M_{x_i})$. By (ii), $Hv_{i-1}x_i =_G Hv_i$. Thus $Hv_n =_G Hx_1 \ldots x_n =_G Hg$, and so $\alpha(v_n) = Hg$ and α is a surjective map.

Suppose that $x_1 \ldots x_n$ and $y_1 \ldots y_m$ are elements of $L(W)$ with the same image Hg under α. By (iii), we have ε, x_1, x_1x_2,…,$x_1 \ldots x_n \in L(W)$ and $(x_1 \ldots x_{i-1}, x_1 \ldots x_i)^\dagger \in L(M_{x_i})$ for $i = 1, \ldots, n$. Thus $\varepsilon\mu(x_1 \ldots x_n) = x_1 \ldots x_n$. Similarly, we get $\varepsilon\mu(y_1 \ldots y_m) = y_1 \ldots y_m$.

Now $Hx_1 \ldots x_n =_G Hg =_G Hy_1 \ldots y_m$. Thus $(\overline{x_1 \ldots x_n})(\overline{y_1 \ldots y_m})^{-1} \in H$. As we saw above, $\varepsilon\bar{\mu}(h) = \varepsilon$ for all $h \in H$, so we have

$$x_1 \ldots x_n = \varepsilon\bar{\mu}(\overline{x_1 \ldots x_n}) = \varepsilon\bar{\mu}(\overline{y_1 \ldots y_m}) = y_1 \ldots y_m.$$

Consequently α is a bijection and $L(W)$ contains a unique representative for each coset, and is prefix closed due to (iii). Now, if $v,\ w \in L(W)$ and $Hvx =_G Hw$ for some $x \in A$, we know that there exists a unique $w' \in L(W)$ with $(v, w')^\dagger \in L(M_x)$. For this w', we have (by (ii)) $Hvx =_G Hw'$. So w' is a representative in $L(W)$ of the coset $H\overline{vx}$. By uniqueness, we deduce that $w = w'$, so that $(v, w)^\dagger \in L(M_x)$ as desired. Thus we have proved that the automata form an automatic structure for G with respect to H.

The fact that an automatic coset system for G with respect to H which is prefix-closed and has unique representatives satisfies (i)-(v) follows easily from the definition of an automatic coset system. □

3. Theoretical results on subgroup presentations

3.1. the composite automaton

The definition of the *composite* of two or more (padded) two-variable automata over the same alphabet is given in (Epstein, Holt, Rees, 1991) and (Holt, 1996). Composite automata are used in our algorithms in the verification of hypothesis (iv) of Theorem 2.2, and in the construction of defining relators for the subgroup H. In order to justify the latter process, we need to consider precisely how these composites are constructed, specifically in the case where the given automata are mipda.

Let $M_1, \ldots, M_k$ be padded two-variable pda or mipda over A. We define the composite language of $M_1, \ldots, M_k$ to be the set of all padded pairs $(u, v)^\dagger$ such that there exist $u_1, \ldots, u_{k-1} \in A^*$ with $(u, u_1)^\dagger \in L(M_1)$, $(u_1, u_2)^\dagger \in L(M_2)$,…, $(u_{k-1}, v)^\dagger \in L(M_k)$. A corresponding composite fsa C that accepts this language is constructed as a non-deterministic automaton, as follows. (For the theoretical treatment, it is easier to allow C to be non-deterministic. In the implementation it will need to be made deterministic, and we will discuss this question in Sections 5 and 6.)

We start by adjoining transitions labeled $(\$, \$)$ from σ to σ, for all accept states σ of each M_i. The state set of C is defined to be $\mathcal{S}(M_1) \times \mathcal{S}(M_2) \times \ldots \times \mathcal{S}(M_k)$, with initial state set $\mathcal{I}(M_1) \times \mathcal{I}(M_2) \times \ldots \times \mathcal{I}(M_k)$, and accept state set $\mathcal{A}(M_1) \times \mathcal{A}(M_2) \times \ldots \times \mathcal{A}(M_k)$. The transitions are defined as follows. For $(x, y) \in A^\dagger \times A^\dagger$, and states $\sigma_i, \tau_i \in \mathcal{S}(M_i) (1 \le i \le k)$, there is a transition $(\sigma_1, \ldots, \sigma_k)^{(x,y)} = (\tau_1, \ldots, \tau_k)$, whenever there exist $x_i \in A^\dagger (0 \le i \le k)$ with $x_0 = x$ and $x_k = y$, and transitions $\sigma_i^{(x_{i-1}, x_i)} = \tau_i$ for $1 \le i \le k$. In order to make C accept the correct padded words in $A^\dagger \times A^\dagger$, we need to make a slight modification to its set $\mathcal{A}(C)$ of accept states. First we adjoin to $\mathcal{A}(C)$ all states from which a state in $\mathcal{A}(C)$ can be reached by repeated application of $(\$, \$)$ (and we need to do this repeatedly until no further accept states are found). We then remove from C all transitions labeled $(\$, \$)$. With this modification, it is easy to see that C accepts precisely the required composite language.

For example, in the case $k = 2$, if $(x, xy) \in L(M_1)$ and $(xy, x) \in L(M_2)$, then the modification to the accept states of C is necessary to ensure that $(x, x) \in L(C)$. Similarly, if $(xy, x) \in L(M_1)$ and $(x, x) \in L(M_2)$, then the initial $(\$, \$)$ transitions adjoined to the M_i are necessary to ensure that $(xy, x) \in L(C)$.

3.2. subgroup presentations

Assume now that G satisfies the hypotheses of Theorem 2.1 (and so it is coset automatic with respect to H), and that W is the coset word-acceptor, and M_x are the coset multiplier mipda, as constructed in the proof of that theorem. Then, if $w = x_1 \ldots x_k$ is a word in A^*, we shall denote the composite automaton of $M_{x_1}, \ldots, M_{x_k}$ by M_w. It is easy to see that, for $u, v \in A^*$, M_w accepts $(u, v)^\dagger$ if and only if $u, v \in L(W)$ and $Huw =_G Hv$. Furthermore, from the remark following the proof of Theorem 2.1, we see that if this is the case, and $u_i \in A^*(0 \le i \le k, u_0 = u, u_k = v)$ are the (necessarily unique) words such that M_{x_i} accepts $(u_{i-1}, u_i)^\dagger$, for $1 \le i \le k$, where an accepting path in M_{x_i} begins with an initial state of M_{x_i} labelled h_i, then $uw =_G h_1 h_2 \ldots h_k v$.

We call an initial state of a mipda *active*, if some path in the mipda starting from that state leads to an accept state. Let K be the set of labels of all active initial states of all of the M_x, except for 1. An active initial state of a composite multiplier M_w, with w as above, has the form $(\sigma_1, \ldots, \sigma_k)$, where σ_i is an active initial state of M_{x_i}. Let h_i be the label of σ_i. Then the label in K^* for the initial state $(\sigma_1, \ldots, \sigma_k)$ is computed from the sequence $(h_1, h_2, \ldots, h_k)$ by omitting any $h_i = 1$ and then multiplying the remaining h_i. Let S be the set of all labels of all active initial states of the automata M_w for $w \in R$ (the set of defining monoid relators of G).

Theorem 3.1. *With the above notation, $\langle K | S \rangle$ is a monoid presentation of H.*

Proof. This is proved by applying the standard Reidemeister-Schreier process as described, for example, in Section 1 of Chapter 9 of (Johnson, 1990), to the subgroup H of G. We apply this process to the given presentation $\langle A | R \rangle$ of G. We are assuming that this is both a group and a monoid presentation of G, but for the purpose of the Reidemeister-Schreier application, we are treating it as a group presentation. The language $L(W)$ of the word-acceptor W is a Schreier transversal for H in G, and so our generating set

$$K = \{\overline{ux}\,\overline{v}^{-1} | u, v \in L(W), x \in A, \overline{ux} \in H\bar{v}, \overline{ux} \ne \overline{v}\}$$

is precisely the set of Schreier generators for H used in the Reidemeister-Schreier presentation. (This is not strictly speaking correct, since the set of Schreier generators is really the set of formal symbols $g_{u,x,v}$, with $u, v \in L(W), x \in A, \overline{ux} \in H\bar{v}, ux \ne v$. This set will be infinite if $|G : H|$ is infinite. However, we are using the set of images in H of the $g_{u,x,v}$ as generators, and we are assuming that this set is finite. This merely amounts to applying Tietze transformations to the formal Reidemeister-Schreier presentation to replace all generators that are equal in H by a single generator.)

We obtain the relators in the Reidemeister-Schreier presentation by rewriting words of the form uwu^{-1}, for $u \in L(W)$ and $w \in R$, as words in the Schreier generators. To do this, we let $w = x_1 \ldots x_k$ with $x_i \in A$ and, for $0 \leq i \leq k$, let u_i be the representative of the coset $H\overline{uw(i)}$ in $L(W)$. (So $u_0 = u$ and, since w is a relator, $u_k = u$.) Then the resulting relator is formally the word

$$g_{u_0,x_1,u_1} g_{u_1,x_2,u_2} \cdots g_{u_{k-1},x_k,u_k},$$

where a term is omitted if $u_{i-1}x_i = u_i$. However, since we are using the images in H as generators, we are writing this as $h_1 h_2 \ldots h_k$, where $u_{i-1}x_i =_G h_i u_i$, and we omit h_i if it is equal to 1.

It is clear from the discussion above, that this word $h_1 h_2 \ldots h_k$ is precisely the label of the active initial state from which (u, u) is accepted by the composite automaton M_w. Conversely, for any active initial state of M_w there is, by definition, some word-pair $(u, v)^\dagger$ accepted by M_w that starts at this state and, since w is a relator, we must have $u = v$. Then rewriting the word uwu^{-1} will yield the label of this state as a relator. Thus $\langle K | S \rangle$ is a group presentation of the subgroup H.

To show that it is a monoid presentation, we make use of our assumption that, for any generator $x \in A$, there is an $x' \in A$ such that $xx', x'x \in R$. Thus, for each generator $h \in K$, we have $ux =_G hv$ for some $u, v \in L(W), x \in A$, and then $vx' =_G h'u$, where $h' \in K$, and h' is equal to the inverse of h in H. Then rewriting the words $uxx'u^{-1}$ and $vx'xv^{-1}$ yields the relators $hh', h'h \in S$. So it is easy to obtain a group presentation in the usual sense (not closed under inversion) of H if required, by removing generators h' and replacing them by h^{-1} wherever they occur in relators. □

4. The algorithm to calculate a shortlex automatic coset system

Roughly speaking, the idea of the algorithm is to construct a finite set D of elements of G that we hope satisfies the hypotheses of Theorem 2.1, and then use D to construct candidates for the word-acceptor and multiplier automata for G, following the method described in the proof of Theorem 2.1. Finally, we check the correctness of these automata by using Theorem 2.2.

There are three problems with this basic idea. Firstly, our method of finding word-differences, which uses the Knuth-Bendix procedure, cannot always produce a set D satisfying the required hypotheses, for reasons that will become clear shortly. Secondly, we need some practical methods of finding new elements to adjoin to D if the resulting automata turn out to be incorrect. Thirdly, our elements of D will have to be words in A^* rather than elements of G.

We shall assume that the reader has some familiarity with the Knuth-Bendix algorithm as applied to monoid presentations. There is some discussion of this in (Epstein, Holt, Rees, 1991) and a very detailed account, together with discussions of implementations, in Chapters 1–3 of (Sims, 1994). In particular, the application of Knuth-Bendix to finitely presented groups with a specified subgroup, which is what is required in our algorithm, is described in Section 2.8 of that book.

The Knuth-Bendix algorithm works with equations, or rewriting rules $u \to v$, in the group, and deduces new equations from the initial ones. In such a rule, we have $u > v$ in some specified ordering on strings of the input alphabet. We use the input alphabet $A \cup \{\#\}$, where # is a symbol that represents the subgroup H of G. The most convenient ordering on words to use is a so-called wreath product ordering (see pages 46–50 of Sims, 1994), in which # has a smaller level than the symbols in A, all of which have the same level, and the shortlex ordering is used for words in A^*. The rules that occur are of two types, the group rules, $w \to z$, which represent equalities $w =_G z$ between group elements, and the coset rules, $\#u \to \#v$, which represent coset equalities $Hu =_G Hv$. Here u, v, w and z are all words in A^*. As initial rules, we take $w \to \varepsilon$, for the defining monoid relators $w \in R$ of G, and $\#u \to \#$ for the words $u \in Y$ that generate H. Note that the generators in A have inverses, and so can be cancelled on the left or right in rules, but # has no inverse, and does not cancel.

A word $w \in A^*$ or $\#u \in \#A^*$ is called *irreducible* if w or u is, respectively, the least word in the ordering that maps onto $\overline{w}$ in G, or the least word such that $\bar{u}$ lies in the coset $H\bar{u}$. (In other words, using the terminology introduced in the introduction, w or u is respectively irreducible or H-irreducible.) A group rewriting rule $w \to z$ is called *minimal* if z is irreducible, and w is reducible, but all of its proper prefixes and suffixes are irreducible. Similarly, a coset rewriting rule $\#u \to \#v$ is called minimal if v is H-irreducible, and u is H-reducible, but all of its proper prefixes are H-irreducible and all of its proper suffixes are irreducible. It can be shown (see Section 2.8 of Sims, 1994 for details) that the Knuth-Bendix algorithm eventually computes the set of all minimal rules. Since this set is usually infinite, the algorithm will not terminate in general, but it is guaranteed that any particular minimal rule will eventually be output if it is run for long enough. The procedure may also produce some non-minimal rules on the way. These will eventually be recognized as being non-minimal, and discarded, but at any given time, the current set of live rules may contain some non-minimal rules.

The problem with this approach arises from the fact that, to generate the complete set D of word-differences required for Theorem 2.1, we need all equations of the form $Hux =_G Hv$, where u and v are H-irreducible. Unfortunately, the Knuth-Bendix algorithm will not necessarily produce all of the related rewriting rules $\#ux \to \#v$, since the word ux may have proper suffixes that are reducible in G, which would mean that the rule was not minimal.

Following (Holt, 1996), we shall say that a word-difference pda or mipda WD for G satisfies the property $\mathcal{P}_1$, if it accepts all pairs $(twx, tz)^\dagger$ and $(ux, v)^\dagger$ where, respectively, $wx \to z$ and $\#ux \to \#v$ are minimal rewriting rules, and t is any word in A^*. Our algorithm will only work if this associated set of word-differences is finite. If we assume that G and H satisfy the coset fellow traveller property, as defined in the introduction, then there will only be finitely many word-differences coming from the minimal coset equations $Hux =_G Hv$. To be certain that there will be only finitely many from the minimal group equations $wx =_G z$, we need to make the additional assumption that G itself is shortlex automatic. From what we have already said, it is clear that, if this set of word-differences is indeed finite then, if we run the Knuth-Bendix procedure for long enough, sufficient rewriting rules will be output such that the associated set of word-differences will contain this complete set, and then the basic word-difference mipda associated with these equations, as defined in the introduction, will satisfy $\mathcal{P}_1$. (It may also contain extra word-differences, since, at any given time, some of the rewriting rules may not be minimal.)

We shall say that a word-difference pda or mipda WD for G satisfies the property $\mathcal{P}_2$, if it accepts all pairs $(u, v)^\dagger$ for which $ux =_G hv$ with $h \in H$, u and v are H-irreducible, and $x \in A$ or $x = \varepsilon$. The extended word-difference mipda associated with the set of all such equations $ux =_G hv$ (cf proof of Theorem 2.1) satisfies $\mathcal{P}_2$.

We can now describe the complete algorithm, which is a direct generalization of that described in (Holt, 1996) for shortlex automatic groups G.

Step 1. Construct word-difference mipda WD_1 and WD_2 for G, which we hope satisfy $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively.

Step 2. Construct a candidate for the coset word-acceptor W.

Step 3. Construct mipda candidates for the multipliers M_x, for all $x \in A$. If, during this construction, we find distinct words u and v in A, both accepted by W, with $Hu =_G Hv$, then modify WD_1 and return to Step 2.

Step 4. Determinize the M_x; that is, construct pda that accept the same language.

Step 5. Check correctness of the M_x assuming the correctness of W. If incorrect, modify WD_2 and return to Step 3.

Step 6. Check correctness of W and M_x. If incorrect, return to Step 1 (or give up!).

We shall now give more details of each of these steps, from a theoretical viewpoint. Implementation issues will be discussed in the following section. Step 1 is just the Knuth-Bendix procedure, which we have already discussed. Since this procedure normally runs forever by default, it needs to be interrupted at some stage, and the word-difference machines WD_1 and WD_2 computed and output. These are essentially the basic and extended mipda calculated from the current set of reduction rules. However, since we cannot be certain that a word is irreducible or H-irreducible at this point, their states will not necessarily correspond to distinct elements in G. In fact, their states are words in A^* that cannot be reduced by using the current set of reduction rules, and the map α (see definition of word-difference machine in the introduction) is the natural map onto G. Of course ε will always be one of these states (and indeed an initial state and the unique accept state), and to conform with the technical condition of property $\mathcal{P}_1$ that WD_1 should accept the group reduction rules preceded by the prefix (t, t) for any $t \in A^*$, we adjoin transitions $\varepsilon^{(x,x)} = \varepsilon$ for all $x \in A$ to WD_1 if necessary.

In Step 2, we construct W using the standard finite state automaton operation

$$W = (E(WD_1 \wedge GT) \cdot A^*)',$$

where GT is the “greater than” automaton for the shortlex ordering, as defined in the introduction. Expressed in words, W accepts $w \in A^*$ if and only if it does not have a prefix u such that there exists $v \in A^*$ with $u > v$ and $(u, v)^\dagger$ accepted by WD_1. Note that our construction of WD_1 (and formulation of the property $\mathcal{P}_1$) ensure that if $(w, z)^\dagger$ is accepted by WD_1 where $w \to z$ is a reduction in G, then $(tw, tz)^\dagger$ is also accepted by WD_1 for any word $t \in A^*$, and so reductions in G occurring in the middle of words are recognized by WD_1. (The coset reductions of the form $\#ux \to \#v$ can only occur on prefixes of words.) It is thus clear that, if WD_1 really does have property $\mathcal{P}_1$, then W will accept precisely the set of H-irreducible words; in other words, it will be correct. (In practice, we often use WD_2 in place of WD_1 in the construction of W. This will be discussed in the next section.)

Step 3 is done following the recipe described in the proof of Theorem 2.1. The words u and v referred to in Step 3 can arise if there is some minimal reduction rule $w \to z$ or $\#u \to \#v$ that was not found by the Knuth-Bendix procedure, so $(w, z)^\dagger$ or $(u, v)^\dagger$ is not accepted by WD_1, but it does happen to be accepted by WD_2. (There is no particular reason to expect this to occur, but in practice it seems to be quite common.) In that case, we simply modify WD_1 so that it does accept this missing reduction rule, and return to Step 2. (If, however, we are using WD_2 rather than WD_1 to construct W, then this situation will never occur when Step 3 is executed immediately after Step 2. It can still occur though, if Step 3 is executed after returning from Step 5 with an updated version of WD_2. In that case, we simply return to Step 3 without updating WD_1.)

Step 4 is a routine operation on finite state automata. We observe at this stage that we can use WD_1 or WD_2 to reduce any word $u \in A^*$ to a word $v \in L(W)$ with $Hu =_G Hv$. To do this, we examine all prefixes u' of u, and by using a systematic search of the relevant WD_i, we look for words v' with $(u', v')^\dagger \in L(WD_i)$ and $u' > v'$. If we locate such pairs, then we substitute v' for u' in u and repeat until we find no new reductions. We must then have $u \in L(W)$. We call this process coset word reduction using WD_i. If we used WD_2 to construct W, then we must also use it for coset word reduction (since some of the required substitutions might not be accepted by WD_1). However, word reduction is often quicker using WD_1.

To explain Step 5, suppose that W is correct but one or more of the M_x is not correct. From the construction of M_x, it is clear that, if M_x accepts $(u, v)^\dagger$, then W accepts u and v, and $Hux =_G Hv$. Thus, there exist words u and v in A^* such that W accepts u and v and $Hux =_G Hv$, but M_x does not accept $(u, v)^\dagger$. But then, since v is the unique word accepted by W that lies in the coset Hux (given that W is correct), M_x cannot accept $(u, w)^\dagger$ for any $w \in A^*$. So we can test for this, by constructing the automata $W \wedge E(M_x)'$ for each $x \in A$, and we do this in Step 5. If all of these automata have empty accepted language and W is correct, then the M_x must also be correct, and we proceed to Step 6 for the complete verification. Otherwise, we find some explicit words u that are not accepted by some $W \wedge E(M_x)'$, and we let v be the reduction of ux (using coset word reduction, as described above); then W accepts v and $Hux =_G Hv$. Since M_x does not accept (u, v), there must be prefixes $u(k)$ and $v(k)$ of u and v of the same length, such that there is no word representing $\overline{u(k)}^{-1} h \overline{v(k)}$ in $\mathcal{S}(WD_2)$, where $h = \overline{ux}\,\overline{v}^{-1}$. We therefore adjoin reduced words for all such elements and their inverses to $\mathcal{S}(WD_2)$, and then recalculate WD_2 itself, and return to Step 3.

The final correctness verification (Step 6) is performed by verifying the hypotheses (i)-(v) of Theorem 2.2. If we succeed in this, then the correctness of W and the M_x follows from this theorem. In fact our constructions of W and the M_x ensure that the hypotheses (i)-(iii) hold automatically. Hypothesis (iv) is checked by constructing the composite multiplier automata M_w (as defined at the beginning of Section 3) for the relators $w \in R$ of G, and verifying that they are equal to the identity multiplier M_ε, which has language $\{(w, w) | w \in L(W)\}$.

For hypothesis (v), we only need to check that the composite automata M_y for $y \in Y$ accept $(\varepsilon, \varepsilon)$. (If hypotheses (i)-(iv) are true, then we already know that, for any $u \in A^*$, there is a unique word $v \in A^*$ such that $(u, v)^\dagger \in L(M_y)$.) This almost follows from the constructions, given that $\#y \to \#$ was one of the initial reduction rules, in the Knuth-Bendix procedure, but it seems difficult to prove it formally. However, to verify it, we do not need to construct M_y explicitly. Suppose $y = x_1 \ldots x_n$, let $w_0 = \varepsilon$ and, for $1 \leq i \leq n$, let w_i be the unique word in $L(W)$ such that $(w_{i-1}, w_i)^\dagger \in L(M_{x_i})$. Then we can calculate the w_i easily by carrying out coset reduction on $w_{i-1}x_i$, as described above, and to verify hypothesis (v) we need only check that $w_n = \varepsilon$ for each $y \in Y$.

5. Implementation Issues

In this section, we shall discuss some of the issues involved in implementing the six steps of the algorithm described in Section 4, and mention some of the decisions taken in the KBMAG package.

We shall not go into details about implementing the Knuth-Bendix procedure, since this question has been extensively treated in Chapters 1–3 of (Sims, 1994), and most of the nontrivial ideas involved in our own program are derived from that source. The only additional point worth making is that it appears to be definitely advantageous to give priority to those reduction rules that give rise to new word-differences, when deciding which rules to use for overlap searching. Giving priority to shorter rules, on the other hand, does not seem to make much difference. These are purely heuristic remarks, based on the examples that we have run.

The main problem for us in Step 1 is to decide when to halt the Knuth-Bendix procedure. Roughly speaking, we want to stop when no new word-differences have appeared for some time. This decision is complicated by the fact that in some larger examples there can be very long gaps between successive appearances of new word-differences. Another technical difficulty is that the existence of some non-minimal reduction rules in the current set of rules can mean that some word-differences can appear and then disappear again, sometimes repeatedly. In our implementation, the program can either be run in verbose mode, when it delivers regular reports on the current number of word-differences, and the user can then halt it interactively when this appears to have stabilized, or it can be given its own internal halting criteria. The default for this is that the total number of reduction rules should have doubled, with no new word-difference appearing, but this factor can be set by the user using a command-line option. If the procedure is halted with a few essential word-differences still missing, then this is not always disastrous, since the remaining steps in the procedure have the capability of finding missing word-differences. However, this usually seems to have the effect that they take longer to run and require more space than when run with the correct complete set of word-differences. (In other words, experience shows that these procedures run most smoothly when given the correct input data. There is no obvious theoretical reason why this should be the case, but it is perhaps not surprising. In fact several of the steps involve determinizing non-deterministic automata, which is a process with exponential complexity, and so, in a sense, it is only by good fortune that the algorithms turn out to be practical at all!)

The construction in Step 2 could be performed by piecing together our basic procedures for the logical operations on automata. In practice we have written a single procedure that does it all at once, which turns out to be considerably more efficient. (It can take advantage of a few technical conditions that would not necessarily hold for general automata, such as the fact that, in a minimal reduction rule $u \to v$ or $\#u \to \#v$, we have $l(u) = l(v), l(v)+1$ or $l(v)+2$. This depends of course on the fact that we are using the shortlex ordering on A^*.) We have also discovered through experience that, in many examples, the complete algorithm runs faster if we use WD_2 in place of WD_1 in the construction of W, even though this usually makes Step 2 itself slower, since WD_2 contains many more transitions than WD_1, most of which are theoretically unnecessary for the correct construction of W. The reason that the complete algorithm can become faster with WD_2 is that it often performs fewer returns to Step 2 from Step 3 in that case. In fact KBMAG now uses WD_2 by default, although the user can elect to use WD_1, and there are some examples where the algorithm only completes at all if we use WD_1.

Turning to Step 3, from the proof of Theorem 2.1, we notice that the multiplier automata M_x all have the same states and transitions, and differ only in their accept states. We make use of this fact in the implementation, by computing and storing a single generalized multiplier automaton M_G, that has labeled states, as described in the introduction. The initial labels L_I will be the empty word ε and the reduced words in A^* representing the initial states $h \in H$, and the accepting labels will be the generators $x \in A$ and ε. Then the accept states of M_x will be precisely those with label x. Our implementation ensures that the only word pairs $(u,v)^\dagger$ that lead from an initial state to ε are (u,u) for all $u \in L(W)$. (This is because, whenever we discover such a pair with $u \neq v$, then we abort the construction of M_G, and return to Step 2.) We can then perform the minimization operation on M_G while maintaining these state labels, as described in the introduction. Experience shows that this generalized minimized multiplier has very few more states than the individual minimized multipliers. It is in any case completely straightforward to construct the individual multipliers from M_G if required.

Another point about our implementation of Step 3 is that we do not necessarily abort the construction of M_G with the first appearance of an equation $Hu =_G Hv$ for $u, v \in L(W)$. Instead, we continue and store up to a specified number of such rules (a few hundred works quite well in practice), and then abort the construction and use these stored rules to update WD_1.

Step 4 is performed on the generalized minimized multiplier M_G. We did experiment with an alternative strategy of determinizing WD_2 before constructing the M_x in Step 3, so the M_x were constructed as pda, but this turned out to be very inefficient, since the determinized version of WD_2 tended to have a huge number of states.

For Step 5, we have coded the construction of the $W \wedge E(M_x)'$ as a single function that takes W and the (minimized) generalized multiplier M_G as input, so it is effectively calculating all of these test automata simultaneously, and it will notice (and abort if required) whenever a word is found that is accepted by any of these automata. This is another big advantage that is gained by using M_G rather than the individual M_x. (This feature applies equally well to the calculation of shortlex automatic structures for groups G (with no subgroup involved), and represents an improvement of our latest package KBMAG over the original Warwick Automata package that was described in (Epstein, Holt, Rees, 1991) and (Holt, 1996).) As in Step 3, we store up to a specified number of words accepted by some $W \wedge E(M_x)'$, then abort the construction and update WD_2.

The main computational problem in Step 6 is the construction of composite automata M_w for words $w \in A^*$, as described in Section 3.1. Note that all of the composite automata concerned here have a unique initial state (they are pda), since the mipda versions of the M_x were converted to pda in Step 4. Let us denote the language of M_w by L_w. It is clear that, if we define a multiplication operation on such languages L_w by

$$L_{w_1}L_{w_2} = \{(u,v)^\dagger | \exists t \in A^*, (u,t)^\dagger \in L_{w_1}, (t,v)^\dagger \in L_{w_2}\},$$

then this operation is associative, and the product $L_{w_1}L_{w_2}$ is equal to $L_{w_1w_2}$. Furthermore, $L_\varepsilon = L(M_\varepsilon) = \{(w,w) | w \in L(W)\}$ is an identity element, and so the set of all such languages forms a monoid.

Verifying hypothesis (iv) of Theorem 2.2 involves checking the equality of two such languages. Since we are working with automata rather than languages, we call two fsa equivalent if they have the same language, and we need to find a canonical member of each equivalence class, so that we can use these to check for equality between their languages. Fortunately there is a standard way of doing this. The composite automata in Section 3.1 were defined as nondeterministic automata. First we we need to determinize them, then minimize them, and finally permute their states so that they appear in order when we scan the transition table. This provides the required canonical representative. (In fact, in our implementations, the final step of permuting the states is unnecessary, since the minimization procedure puts the states into the desired order already.)

It turns out to be most efficient to construct only composites of two automata at a time, and to determinize and minimize the result, before repeating the construction. In fact, our implementation is described very simply, recursively, as follows. Let $w = w_1 w_2$, where $|l(w_1) - l(w_2)| \leq 1$. If $l(w_1) > 1$, then construct M_{w_1}, determinize it and minimize it, and similarly for w_2. Then construct M_w as the composite of M_{w_1} and M_{w_2}, determinize it and minimize it. In practice, we perform the construction and determinization together, by storing the states of the composite of automata M_u and M_v as subsets of $\mathcal{S}(M_u) \times \mathcal{S}(M_v)$.

We can make Step 6 a little more efficient as follows. First, we compute M_w for the inverse monoid relators $w = xx' \in R$, and check that they are all equal to M_ε. If this is the case, then we know that the monoid of composite languages described above is a group in which L_x and $L_{x'}$ are mutual inverses. It then follows that, if $w = w_1 w_2$ is a long relator, and w_2' is the word obtained from w_2 by reversing it and replacing every letter x by x', then $L_w = L_\varepsilon$ if and only if $L_{w_1} = L_{w_2'}$. It is usually quicker to check this latter condition, since it avoids the final (and slowest) stage of calculating L_w itself.

6. The algorithm to compute a subgroup presentation

6.1. SUBGROUP PRESENTATIONS ON THE SCHREIER GENERATORS

Assuming that we have already computed and verified the correctness of the automatic coset system for G with respect to H, to construct a subgroup presentation we need only calculate the generators K and relators S as defined in Section 3.2.

The principal operation involved here is the formation of composite automata M_u for words $u \in A^*$. The difference from the construction of composites described above for verifying correctness of the automatic coset system, is that we now need to construct the composites as mipda. In other respects, the implementation is similar, in that we construct composites recursively, by splitting the word u roughly in half as $u = u_1 u_2$. We then construct M_{u_1} and M_{u_2}, determinize their transition tables and minimize them (while maintaining the labels of their initial states, in the manner described in the introduction), and then construct the composite M_u of M_{u_1} and M_{u_2}, and determinize its transition table and minimize it.

To obtain K and S, we need to be able to locate the active initial states of a mipda, as defined in Section 3.2. This is done by a routine search of the transition table, to see whether any path starting from a particular initial state can lead to success. In the implementation, we remove any inactive initial states from the list of initial states. Having done that, we remove all states of the mipda that cannot be reached from a path starting at an active initial state, in order to make the mipda accessible.

We start the procedure by locating the active initial states of our generalized multiplier mipda M_G that was constructed in Step 3 of the main algorithm described in Section 4. (We are assuming that that algorithm was run successfully to completion, so we know that M_G is correct.) We then relabel these states with suitable symbols, since they form the generating set K for H. The initial state set of the composite mipda of the mipda M_u and M_v is, in general, equal to $\mathcal{I}(M_u) \times \mathcal{I}(M_v)$, and we label such an initial state (σ_1, σ_2) as $w_1 * w_2$, where w_1, w_2 are the labels of σ_1, σ_2, respectively. This has the effect that the labels of the (active) initial states of the mipda composites M_w for $w \in R$ are words in K, and so we obtain the required set S of defining relators for H as the set of these labels coming from all $w \in R$.

As is the case with subgroup presentations computed with current Reidemeister-

Schreier implementations for finite index subgroups, the resulting presentation tends to be very highly redundant, but it is easily simplified, by using Tietze transformations. In our implementation, the subgroup presentation $\langle K|S\rangle$ is output in the format of the GAP system, so that it can be easily read into GAP and simplified, if required.

6.2. subgroup presentations on the user generators

It might be desirable to be able to construct a presentation for H using the given set Y as generators, rather than the set K of Schreier generators. In principle, this could be done quite easily, although we have not yet implemented it completely. Since Y is actually a set of words, we need to introduce a new set of symbols Y', disjoint from X and in one-one correspondence with Y.

All that we need to do is to compute sets $E_{K,Y}$ and $E_{Y,K}$ of equations in G that respectively express the generators in K as words in Y, and vice versa. We then use $E_{K,Y}$ to rewrite the words in S and the equations $E_{Y,K}$ as words (or equations) in the generators Y, and it follows directly from the theory of Tietze transformations that the resulting relators and equations define a presentation for H on Y.

To compute $E_{Y,K}$ we simply carry out coset reduction using WD_1 or WD_2, as described in Section 4, on the words $y \in Y$. Since $y \in H$, y will reduce to the empty word, but we can keep track of the initial states of WD_i (which correspond to the generators K) that occur in the course of the reduction, and the result will be to rewrite y as a word in K^*.

Computing $E_{K,Y}$ is less straightforward, but we have already implemented this part. We need to run the Knuth-Bendix procedure in Step 1 of the main algorithm using an extended generating set, which includes the set Y' mentioned above, of new symbols, in one-one correspondence with Y. These all have the same level as $\#$, which is less than that of the symbols in A, in the wreath product ordering on the complete set of symbols. The initial coset reduction rules $\#y \to \#$, for $y \in Y$, are replaced by $\#y \to y'\#$, where $y' \in Y'$ corresponds to $y \in Y$ and, in the coset reduction rules computed by the Knuth-Bendix process, $\#u \to \#v$ is replaced by $\#u \to t\#v$, for some word t in Y'^*. Since the elements $\overline{uv}^{-1} \in H$ coming from these rules are precisely the generators K derived from the M_x, the words t provide the required rewriting of the elements of K as words in Y'.

There is an additional complication in that some of the generators in K may not come directly from the Knuth-Bendix procedure, but may appear during the iterations through Steps 2–5 of the main procedure. However, at the point when a new generator in K is introduced, there is always enough information present to write it as a word in the existing generators, and so this should not present a problem. Since this process has not yet been implemented, however, there seems little to be gained by going into further details at this stage.

7. Examples

In this final section, we describe the performance of our algorithms on three examples. The programs were run on a SparcStation 20 running Solaris 2, with 256 Megabytes RAM, and using the C-compiler gcc. In general, the axiom checking process (Step 6) requires the most time and space in the algorithm for computing automatic coset systems, but calculating the subgroup presentation usually requires even more resources. The reason is that, in both cases, the bulk of the time is taken up with computing composite multipliers M_w for words $w \in A^*$ and, in general, the longer the word w, the more resources this will require. When we compute a subgroup presentation, we have to calculate the full composite multipliers M_w for the group relators w, whereas, as we saw in Section 5, to verify a group relator w in Step 6, we do not need to compute M_w itself, but only M_{w_1} and M_{w_2}, where w_1 and w_2 are about half as long as w.

The examples described in (Redfern, 1993) that were used for the drawings of circle packings are all either subgroups of free groups or of various Coxeter groups. All of these completed very quickly for us. For example, for the group

$$G = \langle a, b, c, d \mid a^2, b^2, c^2, d^2, (ab)^4, (ac)^3, (ad)^2, (bc)^4, (bd)^4, (cd)^4 \rangle$$

with the subgroup $H = \langle b, c, d \rangle$, which Redfern refers to as "Full Tetrahedron", Steps 1–5 of the algorithm in Section 4 took a total of 4 seconds (it goes once back round the loop from Step 5 – Step 3), Step 6 (axiom checking) took 6 seconds, and calculating the subgroup presentation took 7 seconds. The word-difference machine WD_2 has 40 states, the word-acceptor W has 46 states, and the generalized multiplier M_G has 192 states as a mipda and 185 states after determinization. The subgroup presentation comes out on three generators, which are in fact b, c and d, with the same relators as those in the original presentation of G that involve b, c and d. (This could have been predicted from the general theory of Coxeter groups.)

The other two examples are not quite so easy. For our second example, we take the Fibonacci group $F(2, 8)$ with the presentation

$$\begin{aligned} G = \langle a, b, c, d, e, f, g, h \mid \\ ab = c, bc = d, cd = e, de = f, ef = g, fg = h, gh = a, ha = b \rangle \end{aligned}$$

which, like all $F(2, 2n)$ for $n \geq 4$ is known to be word-hyperbolic, with the subgroup $H = \langle a, e \rangle$. Step 1 of the algorithm took 30 seconds, and produced WD_2 with 89 word-differences. Steps 2–5 took 20 seconds, with no looping, where W has 228 states, and M_G has 1978 states as mipda and 1944 states as pda. Step 6 took 173 seconds, requiring about 5.5 Megabytes of space. Computing the subgroup presentation took about 4300 seconds, and about 15 Megabytes of memory. The presentation comes out on 28 monoid generators (that is, 14 group generators), which simplifies to a free group of rank 2, and so in fact H must be a free group on its defining generators a and e.

For our final example, we take the group

$$G = \langle x, y, z \mid [x, [x, y]] = z, [y, [y, z]] = x, [z, [z, x]] = y \rangle,$$

where the commutator $[u, v]$ is defined to be $u^{-1}v^{-1}uv$. This was proposed by Heineken several years ago as a possible candidate for a finite group with a balanced cyclic presentation. However, we were able to show, using KBMAG, that it is shortlex automatic (in fact, it is even word-hyperbolic), and its generators have infinite order, so it cannot be finite. We used this group as input to our subgroup programs, with the subgroup $H = \langle [x, y], [y, z], [z, x] \rangle$. Step 1 took 105 seconds, and WD_2 has 279 states. Steps 2–5 took 58 seconds, with no looping, where W has 58 states, and M_G has 2520 states as mipda and 2536 states as pda. Step 6 took 399 seconds, using 7 Megabytes of space. Computing the subgroup presentation took about 2800 seconds, and used up to 27 Megabytes. The presentation has 10 monoid generators (that is, 5 group generators), which simplifies to a free group of rank 3, and so again H must be a free group on its defining generators.

We have run the program on many other examples, including some where $|G : H|$ is finite. This typically takes rather longer than the straightforward application of the Reidemeister-Schreier algorithm, but it serves as a useful check on the correctness of our programs, since we have an alternative method of doing the same computation in these cases.

References

Aho, A.V., Hopcroft, J.E., Ullman, J.D. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA.

Epstein, David B.A., Cannon, J.W., Holt, D.F., Levy, S., Patterson, M.S., Thurston, W. (1992). *Word Processing in Groups*, Jones and Bartlett.

Epstein, D.B.A., Holt, D.F., Rees, S.E. (1991). The use of Knuth-Bendix methods to solve the word problem in automatic groups. J. Symbolic Computation **12**, 397–414.

Gersten, S.M., Short, H.B. (1991). Rational Subgroups Of Biautomatic Groups. Ann. Math. **134**, 125–158.

Holt, D.F. (1996). The Warwick automatic groups software. In *Geometrical and Computational Perspectives on Infinite Groups*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 25, ed. Gilbert Baumslag et. al., 69–82.

Hurt, D.F. (1996). *The Use of Knuth-Bendix Methods and Automatic Coset Systems for Solving the Generalized Word Problem and Finding Subgroup Presentations*. PhD Thesis, University of Warwick.

Johnson, D.L. (1990). *Presentations of Groups*. London Math. Soc. Student Texts **15**, Cambridge University Press.

McShane, G., Parker, J., Redfern, I. (1994). Drawing Limit Sets of Kleinian Groups Using Finite State Automata. *Experimental Mathematics* **3**, 153–172.

Redfern, I.D. (1993). *Automatic Coset Systems*. PhD Thesis, University of Warwick.

Sims, Charles C. (1994). *Computation with Finitely Presented Groups*. Encyclopedia of mathematics **48**, Cambridge University Press.