



String rewriting for double coset systems

Ronald Brown^{a,1}, Neil Ghani^b, Anne Heyworth^{b,2},
Christopher D. Wensley^{a,*}

^a *School of Informatics, University of Wales Bangor, Bangor, LL57 1UT, United Kingdom*

^b *Department of Computer Science, Leicester University, Leicester, LE1 7RH, United Kingdom*

Received 2 February 2004; accepted 11 October 2005

Available online 18 November 2005

Abstract

In this paper we show how string rewriting methods can be applied to give a new method of computing double cosets. Previous methods for double cosets were enumerative and thus restricted to finite examples. Our rewriting methods do not suffer this restriction and we present some examples of infinite double coset systems which can now easily be solved using our approach. Even when both enumerative and rewriting techniques are present, our rewriting methods will be competitive because they (i) do not require the preliminary calculation of cosets; and (ii) as with single coset problems, there are many examples for which rewriting is more effective than enumeration.

Automata provide the means for identifying expressions for normal forms in infinite situations and we show how they may be constructed in this setting. Further, related results on logged string rewriting for monoid presentations are exploited to show how witnesses for the computations can be provided and how information about the subgroups and the relations between them can be extracted. Finally, we discuss how the double coset problem is a special case of the problem of computing induced actions of categories which demonstrates that our rewriting methods are applicable to a much wider class of problems than just the double coset problem.

© 2005 Elsevier Ltd. All rights reserved.

PACS: 16S15; 19B37; 20F10; 68Q45

Keywords: Double cosets; String rewriting; Knuth–Bendix; Induced actions; Left Kan extensions

* Corresponding address: University of Wales Bangor, School of Informatics, Dean Street, Bangor, Gwynedd, LL57 1UT, United Kingdom. Tel.: +44 0 1248 382495; fax: +44 0 1248 361429.

E-mail addresses: r.brown@bangor.ac.uk (R. Brown), n.ghani@mcs.le.ac.uk (N. Ghani), a.heyworth@mcs.le.ac.uk (A. Heyworth), c.d.wensley@bangor.ac.uk (C.D. Wensley).

¹ INTAS Project 94-436 ext ‘Algebraic K-theory, groups and categories’. Brown was supported for part of this research by a Leverhulme Emeritus Fellowship 2002–2004.

² EPSRC GR/R29604/01 ‘Kan—A Categorical Approach to Computer Algebra’.

1. Introduction

Given a group G and two subgroups H and K , the double cosets are the equivalence classes of the relation \sim where $g \sim g' \Leftrightarrow h g k = g'$ for some $h \in H, k \in K$. The set of double cosets is written $H \backslash G / K$. Combinatorially, the double cosets are the orbits of the left action of H on the right cosets G / K , and also the orbits of the right action of K on the left cosets $H \backslash G$. Double coset techniques give examples of deep and wide applications of group theoretic methods in chemistry and physics (Ruch and Klein, 1983). For example, there are applications through Polya's theory of counting, to considerations of deuterons colliding in scattering theory (Pletsch, 2001). Real semisimple symmetric spaces are often characterised by a pair of commuting involutions of a reductive group and many of their properties are studied in this setting—in this case double cosets are of importance for representation theory of p -adic symmetric K -varieties (Helmink and Brion, 2000). Double coset computation can be seen as a way of constructing finite quotients of HNN-extensions of known groups or as a way of constructing groups given by symmetric presentations (Curtis, 1992).

There are a number of different questions which arise if we want to compute with double cosets, for example: the enumeration of the double cosets; finding a set of representatives for them; deciding questions such as whether a pair of group elements lie within the same double coset or not; and proving either case. There are number of algorithms for computing such double coset problems but these are regarded as incomplete. Indeed, in Section 4.6.8 of the recently published survey book on computational group theory (Holt et al., 2005) we find “*Unfortunately, no really satisfactory algorithm for solving this problem has been found to date*”. In 1981 the first algorithmic methods for computing double cosets were published (Butler, 1981), and applied to groups of order $\leq 10^4$. The approach was a variation on Dimino's algorithm, described in Butler (1991), for computing a list of elements of a small group. In 1982 Laue described a stepwise method, calculating successive double coset representatives and stabilisers, using a series of subgroups $G = H_0 \subseteq H_1 \subseteq \dots \subseteq H_n = H$. This appears to be most successful in the special cases where known structural properties of the groups involved can be used to speed up the computation (Laue, 1982). In Holt et al. (2005), the basic approach for permutation groups is to use orbit methods to compute left or right cosets, and then orbits of these cosets to obtain double cosets. As pointed out in Holt et al. (2005), this may involve the calculation of a large number of cosets in order to determine a small number of double cosets. More recent methods for computing double cosets of finitely presented groups use Todd–Coxeter procedures (Linton, 1991). All of these methods have been implemented in the commonly used programs for computational discrete algebra, GAP (GAP, 2004) and MAGMA (MAGMA, 2005).

The primary alternative to Todd–Coxeter procedures for ordinary coset enumeration and computation of groups given by presentations is string rewriting (Brown and Heyworth, 2000; Sims, 1994). In finite settings the two approaches are comparable: certain problems being more effectively addressed by the enumerative method and others benefiting more from a rewriting approach. However, for cases involving infinitely many elements, rewriting rather than enumeration is the natural choice.

This paper demonstrates how string rewriting can be applied to the problems of computing double cosets, giving a new alternative to the Todd–Coxeter methods and one which can further be applied to infinite groups. In particular, this paper makes the following contributions.

- The introduction of the notion of a double coset rewriting system and associated Knuth–Bendix completion algorithm as a mechanism for attempting to decide whether two elements of a group lie in the same double coset.

- The specification, in Section 4, of a process which takes a finite complete double coset rewriting system and constructs a finite state automaton whose language is a set of unique normal forms for the double cosets.
- The specification of a higher dimensional version of the Knuth–Bendix algorithm and logged double coset rewriting. This gives, for example, presentations for the subgroups defining the double cosets.
- A discussion of the implementation of these algorithms as a deposited package `kan` for GAP.
- In Section 7 we put our algorithm in context by showing how it arises as a special case of rewriting for an induced action of categories, using a category $H \rightleftarrows K$ constructed from the two subgroups.

2. Rewriting for double cosets

If we consider using rewriting to solve double coset problems we have a choice: to develop a specialised type of rewriting for this situation, or to rephrase the problem in a way that allows existing techniques to be applied. The former method has the advantage of specialty—the “double coset rewriting systems” can be examined in isolation as though they were in some way an advance on existing methods rather than a useful application. This is certainly of some value if one is wishing to write a very specialised program, designed to compute only double coset problems and investigate the particular properties of rewriting systems of this type, but it can obscure the simplicity and the best features of the result. Therefore, we choose to adopt the most straightforward method—simulating the required computations by embedding the group G in a particular free monoid and then applying standard procedures (Book and Otto, 1993). We then have to show that the structure we wish to compute coincides with the rewriting model used.

Definition 2.1 (*Presentation of a Double Coset System*). Let G be a group with monoid presentation $\text{mon}\langle X_G, R_G \rangle$ and let $\theta : X_G^* \rightarrow G$ be the natural monoid homomorphism. Let $X_H, X_K \subseteq X_G^*$ be such that $Y_H = \theta(X_H)$ and $Y_K = \theta(X_K)$ are sets of generators for the subgroups H and K respectively. Then we shall say that the quadruple (X_G, R_G, X_H, X_K) is a presentation of the system of double cosets $H \backslash G / K$.

If R generates an equivalence relation or congruence on a free monoid S then the class of $s \in S$ is denoted $[s]_R$. Similarly, we write $H \backslash G / K = \{[g]_{\sim} \mid g \in G\}$, where \sim is the relation defined at the beginning of Section 1. If $\text{mon}\langle X_G, R_G \rangle$ is a monoid presentation for G then the free monoid T_+ in which we compute is generated by X_G together with two extra (tag) variables H and K . A string HwK represents the double coset $[\theta w]_{\sim}$, and we require a congruence $\overset{*}{\leftrightarrow}_R$ such that $HwK \overset{*}{\leftrightarrow}_R Hw'K$ if and only if $\theta w' \sim \theta w$. Clearly T_+ contains many elements that are not of the form HwK , but these do not arise in the computations performed when completing the rewriting system which determines the double cosets. This is the key observation which allows us to use standard methods.

Theorem 1 (*Double Coset Rewriting*). Let (X_G, R_G, X_H, X_K) be the data for the double coset system $H \backslash G / K$, let H and K be symbols, and let T be the subset of terms of the form HwK of the free monoid $T_+ = (\{H, K\} \cup X_G)^*$ where $w \in X_G^*$. Define

$$R = R_G \cup \{(Hh, H) : h \in X_H\} \cup \{(kK, K) : k \in X_K\}.$$

Let \rightarrow_R be the reduction relation generated by R on the free monoid T_+ , and let $\overset{*}{\leftrightarrow}_R$ be the reflexive, symmetric, transitive closure of \rightarrow_R , which is the congruence generated by R . Then there is a bijection of sets

$$\frac{T}{\overset{*}{\leftrightarrow}_R} \cong \frac{G}{\sim}.$$

Proof. We first show that there is a well-defined map

$$\phi : \frac{T}{\overset{*}{\leftrightarrow}_R} \rightarrow \frac{G}{\sim} \quad \text{where } \phi([HwK]_R) = [\theta w]_{\sim}.$$

If $[HwK]_R = [Hw'K]_R$, there exists a sequence $w = w_1, w_2, \dots, w_n = w'$ in X_G^* such that for $i = 1, \dots, n-1$ either (w_i, w_{i+1}) or (w_{i+1}, w_i) has one of the following forms:

- (i) (ulv, urv) for some $(l, r) \in R_G, u, v \in X_G^*$,
- (ii) (hv, v) for some $h \in X_H, v \in X_G^*$,
- (iii) (uk, u) for some $k \in X_K, u \in X_G^*$.

Since θ is a monoid homomorphism, in the first case $\theta(w_i) = \theta(w_{i+1})$, in the second case $\theta(hv) \sim \theta(v)$ and in the third case $\theta(uk) \sim \theta(u)$. Thus in all cases $[\theta(w_i)]_{\sim} = [\theta(w_{i+1})]_{\sim}$ as required.

Secondly, let $\tau : G \rightarrow X_G^*$ be a section of θ so that $(\theta \circ \tau)g = g$ and $(\tau \circ \theta)g \overset{*}{\leftrightarrow}_{R_G} g$ for all $g \in G$. Further, $\tau(g_1g_2) \overset{*}{\leftrightarrow}_{R_G} \tau(g_1)\tau(g_2)$ since θ maps both $\tau(g_1)\tau(g_2)$ and $\tau(g_1g_2)$ to g_1g_2 . Define

$$\phi' : \frac{G}{\sim} \rightarrow \frac{T}{\overset{*}{\leftrightarrow}_R} \quad \text{where } \phi'([g]_{\sim}) = [H(\tau g)K]_R.$$

To verify that ϕ' is well-defined, suppose $[g]_{\sim} = [g']_{\sim}$ for some $g, g' \in G$. Then, by the definition of \sim , we have $h \in X_H$ and $k \in X_K$ such that $g' = (\theta h)g(\theta k)$, so that

$$H(\tau g')K \overset{*}{\leftrightarrow}_R H(\tau \theta h)(\tau g)(\tau \theta k)K \overset{*}{\leftrightarrow}_{R_G} Hh(\tau g)kK \overset{*}{\rightarrow}_R H(\tau g)K.$$

Finally, we observe that $\phi(\phi'([g]_{\sim})) = \phi([H(\tau g)K]_R) = [\theta(\tau g)]_{\sim} = [g]_{\sim}$, and that $\phi'(\phi([HwK]_R)) = \phi([\theta w]_{\sim}) = [H(\tau(\theta w))K]_R = [HwK]_R$ since $R_G \subseteq R$. Thus ϕ is a bijection with inverse ϕ' . \square

Note the use of the tags H and K . They provide a particularly simple way of deleting elements of X_H from a word providing they occur at the far left of the word and similarly for deleting elements of X_K providing they occur at the far right of the word. Given the double coset presentation (X_G, R_G, X_H, X_K) as in the theorem above, we may refer to R as a *double coset rewriting system* for $H \setminus G / K$. Of course R may not be complete (confluent and noetherian) and so the natural next step would be to use Knuth–Bendix completion to try and obtain an equivalent, but complete, rewriting system. As justified above, we choose to perform this completion by considering rewriting over the free monoid T_+ rather than the set T . However, we must then be sure that if $\rightarrow_{R'}$ is complete on T_+ , and its closure $\overset{*}{\leftrightarrow}_{R'}$ coincides with $\overset{*}{\leftrightarrow}_R$, then the restriction of $\rightarrow_{R'}$ to T is also complete. Fortunately, this is obviously true as \rightarrow_R is closed on the subset T , that is, if $w_1 \rightarrow_R w_2$ then $w_1 \in T$ if and only if $w_2 \in T$. To see this, note that \leftrightarrow_R never removes or adds tags. Thus, we can immediately state the following corollaries.

Corollary 2. *If the double coset rewriting system R can be completed on T_+ then we have a solution to the problem of deciding whether two elements g, g' of G lie within the same double coset.*

Corollary 3. *If the double coset rewriting system R can be completed on T_+ then we can find a unique normal form for each double coset.*

Remark 4 (Implementations). Besides the fact that the string rewriting methods we have presented enable us to tackle problems involving infinite groups, they also allow us to immediately use existing string rewriting programs such as those in GAP and in KBMAG (Holt, 1996) to compute double cosets.

An alternative approach is to complete a rewriting system for G and then construct a double coset rewriting system on the elements of G . Such a system obeys more subtle laws than a standard rewriting system on a free monoid. For example, if $g > g'$ in the termination order used by the completion process, we may not deduce that gg^{-1} is greater than $g'g^{-1}$. Consequently, we believe the approach we have chosen is cleaner than if we were to have worked with multiple rewriting systems at different levels to describe the one structure.

While it is straightforward to try to use a standard Knuth–Bendix completion program to calculate the completion of a double coset rewriting relation on the free monoid T_+ , it will necessarily be less efficient than a specialised Knuth–Bendix program which, for example, restricts itself to the non-free monoid T . For example there will be tests for more overlaps between words wK and zK than can possibly arise: we know (but the program does not) that the only way in which an overlap can occur is when $z = uw$ or $w = uz$, since the tag symbol K will not occur within the strings w or z . A specialised program would also allow different types of ordering, treating symbols H, K in a way different from those in X_G , yielding results that could not be obtained with a standard ordering. If one wishes to do many calculations of this type, it would be worth refining the system to recognise tags and deal with tagged rules sensibly, and to separate rules into subsystems which are completed separately. Such an approach would not be designed specifically for double cosets and could have many other applications (Brown and Heyworth, 2000).

3. Completion of rewriting systems for double cosets

As we have seen, if the double coset rewriting system R is complete, we can solve problems such as whether two elements of the group belong to the same double coset. Usually, R is incomplete, and so we attempt to convert it to an equivalent complete system. We can apply the Knuth–Bendix completion procedure (Knuth and Bendix, 1970) to R on T_+ in the standard way, as detailed below. If R completes on T_+ , we are required to prove that the restriction of \rightarrow_R to T is preserved throughout the algorithm.

Algorithm 1 (Completion). K1 (**Input**) Start with a set of pairs $R = R_G \cup \{(Hh, H) : h \in X_H\} \cup \{(kK, K) : k \in X_K\} \subset T_+$ and a compatible well-ordering on T_+ .

K2 (**Search**) Find all overlaps: pairs of rules $(l_1, r_1), (l_2, r_2)$ which may be applied to the same word and which coincide on some subword. There are essentially two cases, $ul_1v = l_2$ or $ul_1 = l_2v$ for some $u, v \in T$. Add each pair (ur_1v, r_2) or (ur_1, r_2v) to a list of critical pairs.

K3 (**Resolve**) Attempt to resolve each critical pair by reducing both terms with respect to the rules in R . If the reduced terms are equal then the pair has resolved, otherwise the reduced pair is orientated according to the well-ordering and added to a set of new rules and to R .

Table 1
Five types of overlap in G

Type	Overlap case	New rule to add	Picture
Prefix	$l_1 v = l_2$	$(r_1 v, r_2)$	
Suffix	$u l_1 = l_2$	$(u r_1, r_2)$	
Internal subword	$u l_1 v = l_2$	$(u r_1 v, r_2)$	
Left offset	$l_1 v = u l_2$	$(r_1 v, u r_2)$	
Right offset	$u l_1 = l_2 v$	$(u r_1, r_2 v)$	

K4 (Loop) If no new rules were added then go to the next step. Otherwise, repeat the last two steps with the new set of rules, checking pairs that arise between the new rules and between the new rules and the old rules, but not pairs just between old rules (as these have already been checked).

K5 (Output) Return the resulting R^C , a complete rewriting system on T_+ with respect to the given well-ordering.

We now prove that if the input for the algorithm is a double coset rewriting system of the form specified earlier, then the algorithm will attempt to produce an equivalent complete system.

Theorem 5 (Completeness of Double Coset Rewriting Systems). *Let the input for the above algorithm be a double coset rewriting system R as given in Theorem 1, such that the algorithm terminates, giving output R^C . Then the restriction of \rightarrow_{R^C} to T is a complete rewriting system equivalent to the restriction of \rightarrow_R to T .*

Proof. We prove the result directly, by showing that no step in the completion procedure alters the restriction of $\overset{*}{\leftrightarrow}_R$ to T . The argument holds because of the form of the input and the way in which new pairs are generated.

It is convenient (at each stage of the algorithm) to partition a rewrite system R into four subsets R_H, R_K, R_G and R_{HK} , depending on whether the rule involves H, K , neither or both. The subset R_{HK} is initially empty. Formally:

$$\begin{aligned}
 R_G &:= \{(l, r) \in R \mid l, r \in X_G^*\}, \\
 R_H &:= \{(Hl, Hr) \in R \mid l, r \in X_G^*\}, \\
 R_K &:= \{(lK, rK) \in R \mid l, r \in X_G^*\}, \\
 R_{HK} &:= \{(HlK, HrK) \in R \mid l, r \in X_G^*\}.
 \end{aligned}$$

We observe that in step K3 of the algorithm a new rule is generated from an overlap of the left hand sides of two existing rules (followed by their subsequent reductions).

Overlaps of rules (l_1, r_1) and (l_2, r_2) in R_G may be separated into five types. In Table 1, l_1 is either a prefix of l_2 ; a suffix of l_2 ; an internal subword of l_2 ; or the overlap is offset to the left or the right. Neither of u, v may equal the empty word id .

Using the same terminology for the other types of rule, we see that the overlaps which may occur are of the types shown in Table 2. In each case we observe that the new pair added to R will not change the definition of $\overset{*}{\leftrightarrow}_R$ when it is restricted to T since the new pair is an element of $\overset{*}{\leftrightarrow}_R$ and also an element of $T \times T$.

The reduction of the new rules with respect to the existing rules also gives a pair which is already an element of $\overset{*}{\leftrightarrow}_R$ when it is restricted to T . This is because replacement of a substring l_i in a word in T_+ by r_i will preserve the positions of the tags. \square

Table 2
Overlap types for all pairs of rules

Overlap words	Prefix	Suffix	Internal	Left offset	Right offset
l_1, l_2	✓	✓	✓	✓	✓
l_1, Hl_2		✓	✓		✓
l_1, l_2K	✓		✓	✓	
Hl_1, Hl_2	✓				
Hl_1, l_2K				✓	
l_1K, l_2K		✓			
l_1, Hl_2K			✓		
Hl_1, Hl_2K	✓				
l_1K, Hl_2K		✓			

4. Automata for double coset rewriting systems

When the number of double cosets is infinite, it is not possible to list all their normal forms, so a regular expression giving an impression of the shape or pattern of these forms may be useful. In simpler string rewriting systems, for example with left or right cosets, we can build up a catalogue of the normal forms. In finite cases where we have a length non-increasing order this is extremely effective, and even in infinite cases it can serve to show up patterns. The cataloguing procedure relies heavily on the fact that, if a term is reducible and generators are appended onto a chosen end, then the term remains reducible.

In the case of double cosets and doubly tagged strings we cannot use these methods. If a term HLK is reducible, then one of $l \rightarrow r$ or $Hl \rightarrow Hr$ or $lK \rightarrow rK$ or $HLK \rightarrow HrK$ is true, but we cannot assume that either $HlxK$ or $HxlK$ are reducible. For example, if $Hl \rightarrow Hr$, then we are unable to deduce that $Hxl \rightarrow Hxr$. Since we cannot use cataloguing, we turn to the alternative: in string rewriting we use automata when we wish to find an expression for the set of normal forms and cataloguing is insufficient. Here, we describe techniques for constructing automata whose languages are the sets of normal forms for our double cosets. First we recall the construction of an automaton for accepting normal forms in G .

We will use the following notation. For any set of rules R we set $l(R) = \{l \mid (l, r) \in R\}$, the set of *left-hand sides* of these rules. Then $pl(R)$ is the set of all *prefixes* of the rules and $ppl(R)$ is the set of all *proper prefixes*:

$$pl(R) = \{u \mid (uv, r) \in R, u \neq \text{id}\}, \quad ppl(R) = \{u \mid (uv, r) \in R, u, v \neq \text{id}\}.$$

Similarly, $sl(R)$ and $psl(R)$ denote the sets of *suffixes* and *proper suffixes*. A non-deterministic automaton N , with state set S ; alphabet Σ ; initial state $s_0 \in S$; transition function $\delta : S \times \Sigma \rightarrow 2^S$; and accepting states $A \subseteq S$, is written $N = (S, \Sigma, s_0, \delta, A)$. A deterministic automaton has $\delta : S \times \Sigma \rightarrow S$.

Definition 4.1 (*Word Acceptor for G*). The *word acceptor* of a group $G = \text{mon}\langle X_G, R_G \rangle$ with a finite complete rewrite system R_G^C is constructed as follows. First form a non-deterministic automaton

$$N_G = (ppl(R_G^C) \cup \{\text{id}, \text{sink}\}, X_G, \text{id}, \delta_G, \{\text{sink}\})$$

whose states consist of all proper prefixes p from R_G^C ; the identity word as initial state; and a sink state sink which is the only accepting state.

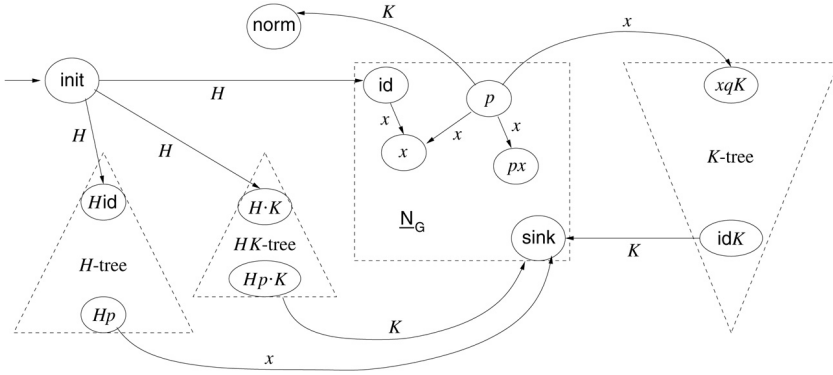


Fig. 1. Sketch of a double coset automaton.

The transition function is given by

$$\delta_G(\text{sink}, x) = \{\text{sink}\} \quad \text{for all } x \in X_G,$$

$$\delta_G(p, x) = \begin{cases} \{\text{sink}\} & \text{if } px = ul \text{ for some } l \in l(R_G), \text{ else} \\ \{p_i \in \text{ppl}(R_G) \mid px = u_i p_i \text{ for some } u_i \in X_G^*\}. \end{cases}$$

Standard results of automata theory (Cohen, 1991; Lawson, 2004) allow us to determinize \underline{N}_G (using the accessible subset construction); take the complement (accepting states become non-accepting, and conversely); and minimize, giving a deterministic automaton \underline{A}_G which accepts only the normal forms of elements of G .

Theorem 6 (Word Acceptor for $H \setminus G / K$). *Let R^C be a finite, complete, double coset rewriting system for subgroups H generated by $Y_H = \theta X_H$ and K generated by $Y_K = \theta X_K$ of the group G which is given by the monoid presentation $\text{mon}\langle X_G, R_G \rangle$. Let T and T_+ be defined as in Theorem 1. Then there is a regular expression representing a regular language L over T_+ such that $L = \{[g]_{\sim} : g \in G\}$.*

Proof. We define a non-deterministic automaton \underline{N} with input alphabet $\Sigma = X_G \cup \{H, K\}$ which accepts exactly the set $\text{irr}_{R^C}(T)$ of irreducible elements of T with respect to \rightarrow_{R^C} .

As before, we partition the rules in R^C into $R_G^C \cup R_H^C \cup R_K^C \cup R_{HK}^C$, where R_G^C is the complete rewrite system for G . The automaton has four main components:

- the non-deterministic form \underline{N}_G of the word acceptor for G , with states S_G , as in Definition 4.1;
- an H -tree, whose states are $S_H = \{Hid\} \cup \text{ppl}(R_H^C)$;
- a K -tree, whose states are $S_K = \{\text{id}K\} \cup \text{psl}(R_K^C)$;
- an HK -tree, whose states are $S_{HK} = \{Hid \cdot K\} \cup (\text{ppl}(R_{HK}^C) \cdot K)$.

There are two additional states, an initial state init and a normal form state norm which is the only non-accepting state. An informal sketch showing how these components and states are connected by transitions is shown in Fig. 1.

The formal definition of the non-deterministic automaton \underline{N} is

$$\underline{N} = (S, \Sigma = X_G \cup \{H, K\}, \text{init}, \delta, S \setminus \{\text{norm}\}), \quad \text{where}$$

$$S = \{\text{init}, \text{norm}\} \cup S_G \cup S_H \cup S_K \cup S_{HK}.$$

The transition function δ is defined in Table 3, where $s \in S, x \in X_G, a \in \Sigma, p \in \text{ppl}(R_G^C), Hp \in \text{ppl}(R_H^C), qK \in \text{psl}(R_K^C)$ and $Hp \cdot K \in \text{ppl}(R_{HK}^C) \cdot K$.

Table 3
Transition function δ for double cosets automaton

Location	Transition
from init	$\delta(\text{init}, H) = \{\text{id}, \text{Hid}, \text{Hid} \cdot K\}$ $\delta(\text{init}, a) = \{\text{sink}\}$ when $a \neq H$
by H	$\delta(s, H) = \{\text{sink}\}$ when $s \neq \text{init}$
by K	$\delta(p, K) = \{\text{norm}\}$ $\delta(\text{id}K, K) = \{\text{sink}\}$ $\delta(Hp \cdot K, K) = \{\text{sink}\}$ if $HpK \in l(R_{HK})$
H -tree	$\delta(Hp, x) = \begin{cases} \{Hpx\} & \text{if } Hpx \in \text{ppl}(R_H) \\ \{\text{sink}\} & \text{if } Hpx \in l(R_H) \end{cases}$
K -tree	$\delta(xqK, x) = \{qK\}$ if $xqK \in \text{psl}(R_K)$
HK -tree	$\delta(Hp \cdot K, x) = \{Hpx \cdot K\}$ if $Hpx \in \text{ppl}(R_{HK})$
in \underline{N}_G	$\delta(p, x) = \begin{cases} \{\text{sink}\} & \text{if } px = ul \text{ for some } l \in l(R_G), \text{ else} \\ \{p_i \in \text{ppl}(R_G) \mid px = u_i p_i\} \cup \{xqK \mid xqK \in l(R_K)\} \end{cases}$
otherwise	$\delta(s, a) = \emptyset$

The extended state transition function $\delta^* : S \times \Sigma^* \rightarrow 2^S$ is such that, for $t \in T$, the intersection of $\delta^*(\text{init}, t)$ with $S \setminus \{\text{norm}\}$ is non-empty if and only if t is a word in Σ^* which is not in T or is reducible.

Just as we converted \underline{N}_G to \underline{A}_G , we make \underline{N} deterministic; take its complement; and minimize. The language L recognised by the resulting automaton \underline{A} is $T_+ - (T_+ - \text{irr}_{R^C}(T)) = \text{irr}_{R^C}(T)$. Hence, by Kleene’s Theorem, L is regular. Since R^C is a complete rewriting system on T , there exists a unique irreducible word in each class of T with respect to $\xrightarrow{*}_R$. Therefore the set $\text{irr}_{R^C}(T)$ is bijective with $T / \xleftrightarrow{*}_R = L$.

The automaton \underline{A} gives rise to a system of right linear language equations with a unique solution, which is a regular expression for the language L accepted by the automaton. The regular expression can be obtained by applying Arden’s Theorem to solve the language equations. \square

Thus an automaton \underline{A} can be constructed from the complete double coset rewriting system and a regular expression for the set of double cosets L is obtained from solving the language equations of the determinized and minimized complement of \underline{A} . Section 6 includes some examples of these automata.

5. Logged double coset rewriting

It is often useful to label the original rewrite rules and record how they are used during Knuth–Bendix completion, and then during the reduction of words. One instance is when we require precise proof of a particular equivalence in terms of the original data.

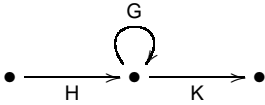
Suppose that $\alpha_s : s \rightarrow t$ is a rewrite rule. Then we know that the rewrite $usv \rightarrow utv$ can be performed, and a reasonable label for this is $u\alpha_s v$. Similarly if $\alpha_t : t \rightarrow q$ then we may perform α_s followed by α_t , rewriting $s \rightarrow q$, which we choose to label $\alpha_s \bullet \alpha_t$. Thus any sequence of rewrites may be recorded by a combination of labels of the form:

$$u_1 \alpha_{s_1} v_1 \bullet u_2 \alpha_{s_2} v_2 \bullet \dots \bullet u_n \alpha_{s_n} v_n.$$

The algebra of recorded rewrites is formalised by observing the sesquicategorical or 2-categorical structure. Briefly, a 2-category consists of 0-cells (objects \bullet_i), 1-cells (arrows between objects $(w_{ij} : \bullet_i \rightarrow \bullet_j)$) and 2-cells (arrows between arrows $(\alpha : w_{ij} \rightrightarrows w'_{ij})$), with a category structure on the 1-cells (arrow composition) and two compatible (by the interchange law)

category structures (horizontal and vertical composition) on the 2-cells, which preserve sources and targets (Mac Lane, 1998).

In particular *logged double coset rewriting*, is formalised in terms of a 2-category whose 0-cells (vertices) and 1-cells (paths along arrows) are illustrated in the following graph.



The generating 2-cells are

$$\begin{aligned} \alpha_{h_i} &: Hh_i \rightarrow H, & \text{for each } h_i \in X_H, \\ \alpha_{l_i} &: l_i \rightarrow r_i, & \text{for each } (l_i, r_i) \in R_G, \\ \text{and } \alpha_{k_i} &: k_i K \rightarrow K, & \text{for each } k_i \in X_K. \end{aligned}$$

The vertical composition of 2-cells, written $\alpha \bullet \gamma$, is the composition of the two rewrites when the source of the second coincides with the target of the first. We may also “whisker” the 2-cells with suitable 1-cells. For example, the 2-cell α_{l_i} may be whiskered by Hu on the left and v on the right to obtain a 2-cell $(Hu\alpha_{l_i}v : Hul_i v \rightarrow Hur_i v)$. Finally, we require the interchange law, so that it does not matter in which order we combine a pair of 2-cells which rewrite non-overlapping parts of a string. This defines the horizontal composition,

$$\alpha \circ \beta := \alpha \text{ src}(\beta) \bullet \text{tgt}(\alpha)\beta = \text{src}(\alpha)\beta \bullet \alpha \text{ tgt}(\beta),$$

and corresponds to the fact that if rewrite rules do not overlap on a string then it does not matter which one we apply first. Note that whiskering is equivalent to composing with identity 2-cells, $u\alpha v = 1_u \circ \alpha \circ 1_v$ where $(1_u : u \Rightarrow u)$ for $u \in \Sigma$, and that $\alpha v \circ \beta = \alpha \circ v\beta$.

Logged rewriting for monoid presentations is explained in detail in Heyworth and Johnson (2005) and we shall only recall the key ideas here.

The essential difference between the logged version and the standard Knuth–Bendix algorithm is the level of detail it records. If we have an overlap which introduces a new rule, we require an expression for the new rule in terms of the original labels. Often we will be in the situation where w reduces to w_1 by one sequence β_1 of 2-cells and to w_2 by another sequence β_2 . Assuming w_2 is the larger string, we add in the rule $w_2 \rightarrow w_1$, and note that this relation can be achieved by “un-reducing” w_2 to w and then reducing w to w_1 . This “un-reducing” is more formally known as an *inverse derivation* and gives the vertical composition of 2-cells a groupoid structure. In this situation we add the 2-cell $\beta_2^{-1} \bullet \beta_1$ at the same time as the rule $w_2 \rightarrow w_1$.

Assume that, using these methods, we obtain a complete, logged, rewrite system for the double cosets, which means that we have a 2-cell associated with each of the rules. Suppose now that we have two group elements g_1 and g_2 , represented by strings w_1 and w_2 in the free monoid, so that $\theta(w_1) = g_1$ and $\theta(w_2) = g_2$. We can determine whether or not g_1 and g_2 lie within the same double coset by rewriting Hw_1K and Hw_2K . If they both reduce to the same string Hzk , then we can examine the logs of the reductions to find $h_1, \dots, h_m \in X_H$ and $k_1, \dots, k_n \in X_K$ such that

$$\theta(h_1^{\varepsilon_1}) \cdots \theta(h_n^{\varepsilon_n}) g_1 \theta(k_1^{\varepsilon_{n+1}}) \cdots \theta(k_m^{\varepsilon_{n+m}}) = g_2. \tag{1}$$

The following section includes examples of this computation.

In the case of left or right cosets, the logs of the complete rewriting system may be used to derive a presentation for the subgroup itself. In other situations the logs and

particularly the logs of circular rewrites (endorewrites) have more interesting interpretations and applications (Heyworth and Wensley, 2003). In the double coset case we can make the following observations. We use $E(w)$ to denote the set of endorewrites of the string w , the set of 2-cells associated to rewrites of w back to itself.

- (i) The sets $E(Hw)$ are bijective with each other for all w in X^* . These give information about the group H in the form of a presentation (Ghani and Heyworth, 2003).
- (ii) Similarly, the sets $E(wK)$ give a presentation of K .
- (iii) The sets $E(w)$ are all bijective, and these give generators for the module of identities among relations for the group G (Heyworth and Johnson, 2005; Heyworth and Wensley, 2003).
- (iv) The sets $E(HwK)$ are not all bijective in general. However, in the case that $\theta(w_1) = \theta(w_2)$, there is a bijection between the sets $E(Hw_1K)$ and $E(Hw_2K)$. In general each endorewrite of this type gives us information regarding the relationship between the subgroups H and K within G . Generators for the subgroup $H \cap K$ can certainly be obtained in this way.

6. Examples

The examples given below were calculated using a prototype package `kan`, available from Heyworth and Wensley (2005). This is a collection of GAP functions which are designed to tackle rewriting problems by translating them to a categorical framework; using a generalised Knuth–Bendix type algorithm to solve the problem; and then translating back into the format appropriate for the structure in question. The double coset functions in `kan` make use of functions from the GAP package `automata` (Delgado et al., 2005). The main purpose of these examples is to demonstrate the methods we have presented and the fact that they can be applied to a wider class of problems than could previously be computed, rather than to be technically impressive.

Example 7 (*A Finite Double Coset Rewriting System*). Let G be the free group on generators $\{a, b\}$ and let $H = \langle a^6 \rangle$, $K = \langle a^4 \rangle$. (Varying the powers of a gives a family of examples of this type.) The double coset HK contains $a^2 = a^{\text{gcd}(6,4)}$, and it is clear that one set of double coset representatives is

$$\{HK, HaK, Ha^i ba^j K, Ha^i buba^j K \mid 0 \leq i \leq 5, 0 \leq j \leq 3, u \in \{a, b\}^*\}.$$

The initial set of rules is

$$\{(Aa, \text{id}), (aA, \text{id}), (Bb, \text{id}), (bB, \text{id}), (Ha^6, H), (a^4 K, K)\}.$$

After completion, the last two rules are replaced by

$$\{(Ha^4, HA^2), (HA^3, Ha^3), (a^3 K, AK), (A^2 K, a^2 K), (Ha^2 K, HK), (HAK, HaK)\}.$$

Note that two HK -rules appear, reflecting the fact that $H \vee K = \langle a^2 \rangle$. The non-deterministic automaton \underline{N} has 22 states,

$$\{\text{init, norm, sink}\} \cup \{\text{id, } a, A, b, B\} \cup \{H, Ha, Ha^2, Ha^3, HA, HA^2\} \\ \cup \{K, aK, a^2 K, AK\} \cup \{H \cdot K, Ha \cdot K, Ha^2 \cdot K, HA \cdot K\}.$$

Determinizing \underline{N} gives an automaton with 24 states which, after complementation and minimization, reduces to a deterministic automaton with 15 states and transitions shown in Table 4 (where $\mathbb{N}, \mathbb{S}, \mathbb{I}$ correspond to `norm, sink, init`).

Table 4
Minimal double coset automaton for Example 7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	\mathbb{N}	\mathbb{S}	B	\mathbb{I}	a	id	a^3	b	BA	a^2	ba^3	ba^2	ba	A	BA^2
H	2	2	2	6	2	2	2	2	2	2	2	2	2	2	2
K	2	2	1	2	1	1	2	1	1	2	2	1	1	2	2
a	2	2	13	2	10	5	2	13	2	7	11	11	12	2	2
A	2	2	9	2	2	14	2	9	15	2	2	2	2	7	15
b	2	2	2	2	8	8	8	8	8	8	8	8	8	8	8
B	2	2	3	2	3	3	3	2	3	3	3	3	3	3	3

Table 5
Logged rewrite rules for the trefoil monoid

Rule	Label
(Yy, id)	α_3
(yY, id)	α_4
(x^3, y^2)	α_5
(y^2x, xy^2)	$\alpha_6 = (\alpha_5^{-1}x) \bullet (x\alpha_5)$
(X, x^2Y^2)	$\alpha_7 = (X\alpha_4^{-1}) \bullet (Xy\alpha_4^{-1}Y) \bullet (X\alpha_5^{-1}Y^2) \bullet (\alpha_1x^2Y^2)$
(Yx, yxY^2)	$\alpha_8 = (Yx\alpha_4^{-1}) \bullet (Yxy\alpha_4^{-1}Y) \bullet (Yx\alpha_5^{-1}Y^2) \bullet (Y\alpha_5xY^2) \bullet (\alpha_3yxY^2)$

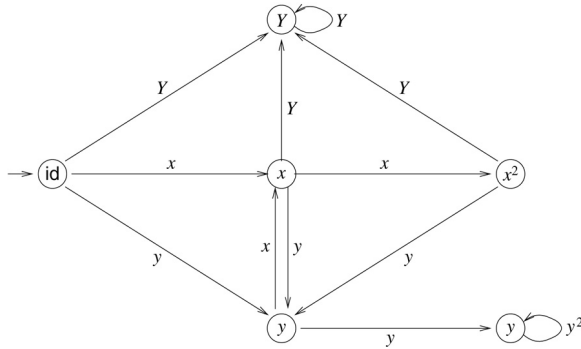


Fig. 2. Sketch of the word acceptor $\underline{A}_{\mathcal{T}}$ for \mathcal{T} .

Example 8 (The Trefoil Group). This is an example in which the group has a finite rewriting system, but the double coset system is infinite. Starting with an initial monoid presentation with rules

$$[\alpha_1 = (Xx, \text{id}), \alpha_2 = (xX, \text{id}), \alpha_3 = (Yy, \text{id}), \alpha_4 = (yY, \text{id}), \alpha_5 = (x^3, y^2)],$$

the fundamental group $\mathcal{T} = \langle x, y \mid x^3 = y^2 \rangle$ of the trefoil knot has a complete rewriting system with six logged rules shown in Table 5.

The ordering used here is the wreath product order with $X > x > Y > y$. A group version of these logged rules is given in Heyworth and Wensley (2003).

The non-deterministic automaton $\underline{N}_{\mathcal{T}}$ has seven states, and there are 12 states in the determinized automaton, reducing to seven states on minimization. The automaton $\underline{A}_{\mathcal{T}}$ is pictured in Fig. 2. (For clarity, the sink state and transitions to it have been excluded. All states are accepting, so double circles have been omitted.)

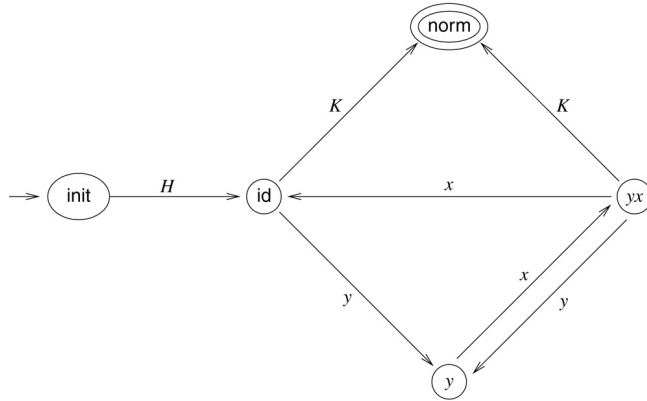


Fig. 3. Double coset automaton for the trefoil group.

A regular expression for the language accepted by \underline{A}_T is

$$(1 + y)x(yx + xyx)^*(1 + x)(y^* + Y^+) + (y^* + Y^+).$$

We consider subgroups $H = \langle x \rangle$ and $K = \langle y \rangle$. This is an example which apparently cannot be computed using algorithms previously available. The double coset rewriting system initially requires the additional rules

$$\{\beta_1 = (HX, H), \beta_2 = (Hx, H), \beta_3 = (YK, K), \beta_4 = (yK, K)\}.$$

The `kan` package includes a *limited* version of the Knuth–Bendix functions which stop the calculation after a specified number λ of rules have been added to the system. Subsets of rules which involve either H , or K , or both may then be extracted. Adopting the wreath product order with $K > H > X > x > Y > y$ we find that β_1 reduces, and there are no additional K -rules or HK -rules. As λ increases we obtain an increasing number of rules from the following infinite set:

$$\{(Hwy^2, Hw), (HwY, Hwy) | w \text{ is any word in } \{yx, yx^2\}^*\}.$$

Note that the words yx and yx^2 label directed cycles in Fig. 2. It is straightforward to verify that, if we add all these rules, the system is complete, despite the fact that it is infinite: the H acting as a tag on the left restricts the possible overlaps severely and only a few cases need be checked.

Also as λ increases, the left-hand sides of the rules may be used to form the three trees in the double coset automaton of Fig. 1. Applying the construction of Theorem 6, we obtain a sequence of minimized automata $\{\underline{A}_\lambda\}$ which rapidly converges to the automaton depicted in Fig. 3. Indeed, with $\lambda = 10$, we obtain a sufficient set of three H -rules and two K -rules, namely $\{\beta_2, \beta_3, \beta_4\}$ together with

Rule	Label
$\beta_5 = (Hy^2, H)$	$H\alpha_5^{-1} \bullet \beta_2 x^2 \bullet \beta_2 x \bullet \beta_2$
$\beta_6 = (HY, Hy)$	$\beta_5^{-1} Y \bullet Hy\alpha_4$

The normal forms can be read straight off the automaton:

$$\{HwK | w \text{ is any word in } \{yx, yx^2\}^*\}.$$

For an example of logged reduction, consider the double coset HYK , where Y is a normal form for \underline{A}_T . Applying $H\beta_3$ we obtain immediately that $HYK \rightarrow HK$. Alternatively, we may apply

$$\beta_6K \bullet H\beta_4 = \beta_2^{-1}YK \bullet \beta_2^{-1}xYK \bullet \beta_2^{-1}x^2YK \bullet H\alpha_5K \bullet Hy\alpha_4K \bullet H\beta_4,$$

which gives successive rewrites

$$HYK \rightarrow HxYK \rightarrow Hx^2YK \rightarrow Hx^3YK \rightarrow Hy^2YK \rightarrow HyK \rightarrow HK.$$

Applying Eq. (1) to these two reductions (where $w_1 = Y$ and $w_2 = \text{id}$) gives

$$\begin{aligned} \theta(w_1)\theta(Y)^{-1} &= y^{-1}y = 1_G = \theta(w_2), \\ (\theta(x))^3\theta(w_1)\theta(y)^{-1} &= \theta(x^3Y^2) = 1_G = \theta(w_2), \end{aligned}$$

and so we have obtained an endorewrite $H\beta_3^{-1} \bullet \beta_6K \bullet H\beta_4 \in E(HK)$.

Example 9 (*A Group with an Infinite Rewriting System*). When the group G has an infinite rewriting system the double coset rewriting system will also be infinite. In this case it may be possible to use the package **KBMAG** to compute a word acceptor for G . In the **kan** package the finite state automaton provided by **KBMAG** is converted to a deterministic automaton in the format used by the **automata** package, and then included as the \underline{N}_G part of the double coset automaton shown in Fig. 1. It is still necessary to find appropriate sets of rules R_H , R_K and R_{HK} and, since R_G is infinite, the limited Knuth–Bendix functions should again be used. An interactive use of the package is required: experimenting with different limits gives partial results from which we may be able to deduce exact answers.

In the following example we take G to be the group with generators $\{a, b\}$ and relators $[a^3, b^3, (ab)^3]$. The normal forms of the monoid elements are strings alternating in a or A with b or B . Not all such strings are irreducible, for example $bab \rightarrow ABA$ and $abaB \rightarrow BAB$. The automatic structure computed by **KBMAG** has a word acceptor with 17 states.

If we take H to be generated by $[ab]$ and K to be generated by $[ba]$, we find that all three additional sets of rules are infinite:

$$\begin{aligned} R_H &= \{(Hab, H), (HaB, Hb), (H(bA)^nB, H(Ab)^na), n \geq 0\}, \\ R_K &= \{(baK, K), (BaK, bK), (B(Ab)^nK, a(bA)^nK), n \geq 0\}, \\ R_{HK} &= \{(Hb(Ab)^nK, H(Ab)^nAK), n \geq 0\}. \end{aligned}$$

The sequence $\{\underline{A}_\lambda\}$ of minimized automata exhibits an increasing number of states, reaching 48 at $\lambda = 200$. On inspection, we find that these automata have a common set of states (the right-hand half in Fig. 4), and two chains which gradually increase in length. Taking $\underline{A} = \lim_{\lambda \rightarrow \infty} \underline{A}_\lambda$, these chains shrink to two-state loops, and we obtain the 19-state automaton shown in Fig. 4, where the **norm** and **sink** states have been omitted, and states shown with a double circle are those having a K -transition to **norm**.

The language recognised by \underline{A} is

$$H(a + (\text{id} + A)(bA)^* + AB(aB)^*A + b(aB)^+A(bA)^* + A(bA)^*(Ba)^*b)K.$$

7. Induced actions and left Kan extensions

The algorithms we have presented arose as part of a programme of applying categorical constructions in computer algebra so as to allow increased flexibility and a wider range of applications. It is worth recalling (Mac Lane, 1998) in noting that our goal is *not the reduction of*

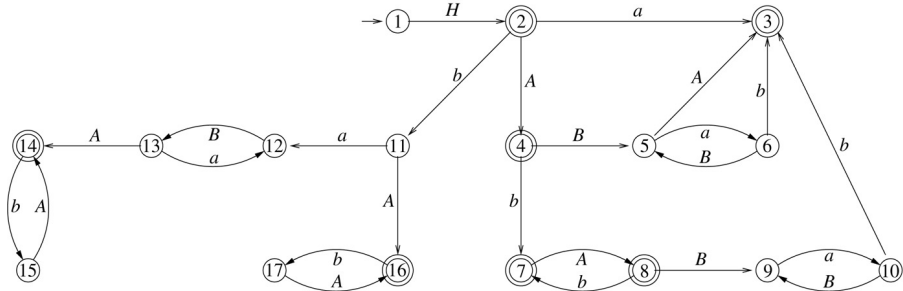


Fig. 4. Automaton \underline{A} for Example 9.

the familiar to the unfamiliar, but the extension of the familiar to cover many more cases. In this section, we shall demonstrate this by showing how the double coset problem is an instance of the much more general construction of induced actions of monoids and categories. Our aim also is to advertise this construction, which has many uses apart from those given here (for examples see Brown and Heyworth (2000)).

Let $F : M \rightarrow N$ be a morphism of monoids, and let M, N act on sets X, Y respectively. These are right actions, and we use the notation x^m, y^n . An F -morphism $\varepsilon : X \rightarrow Y$ is a function of sets such that $\varepsilon(x^m) = (\varepsilon x)^{F(m)}$ for all $m \in M, x \in X$. In the case $F = 1_N$ we call ε an N -morphism. The F -morphism ε is said to be universal if, for any action of N on a set Z and any F -morphism $\phi : X \rightarrow Z$, there is a unique N -morphism $\psi : Y \rightarrow Z$ such that $\psi \circ \varepsilon = \phi$. When ε is universal we say that the action of N on Y is induced from X by F , and we write $Y = F_*(X)$.

It is easy to construct this Y from the action $X \times M \rightarrow X$ and the morphism F . We let $Y' = X \times N$ with N -action $(x, n)^{n'} := (x, nn')$, and define Y to be Y' factored by the equivalence relation \approx generated by

$$(x^m, n) \approx (x, F(m)n), \quad n \in N, \quad m \in M, \quad x \in X, \tag{2}$$

with $\varepsilon : X \rightarrow Y$ mapping x to the equivalence class of $(x, 1_N)$.

A common example is where M is a subgroup of a group N , F is the inclusion morphism, and X is a singleton. Then $F_*(X)$ can easily be identified with $M \backslash N$, the set of left cosets of M in N , and the usual N -action. Note also that we immediately have an extension of the problem of determining cosets to that of extending an action of the subgroup M on a set X to an action of N on a new set $F_*(X)$.

This notion of induced action is easily extended to the case where M, N are small categories, $F : M \rightarrow N$ is a functor, and the action of M is given by a functor $X : M \rightarrow \mathbf{Sets}$ (see below). In Heyworth (1999) and Brown and Heyworth (2000) string rewriting for a monoid presentation of a monoid N was generalised to string rewriting for an induced action of categories, given a presentation of the data for this. Our string rewriting procedure for double cosets is a special case of this general form of string rewriting for induced actions of categories.

Note that induced actions are also well known in category theory as left Kan extensions, and have many applications under that terminology. In that setting, it is often convenient to describe a choice of left Kan extensions for all actions as giving a functor of functor categories

$$F_* : (\mathbf{Sets})^M \rightarrow (\mathbf{Sets})^N$$

which is left adjoint to the standard functor

$$F^* : (\mathbf{Sets})^N \rightarrow (\mathbf{Sets})^M$$

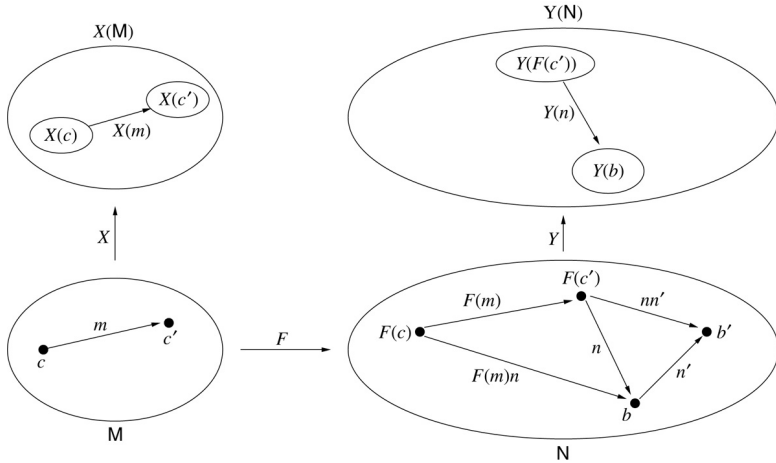


Fig. 5. Action induced from X by F .

given by composition with F . An implication of this is that F_* preserves colimits of actions, but we do not pursue this theme.

The construction of the functor $Y = F_*(X) : \mathbf{N} \rightarrow \mathbf{Sets}$ in the category case is an easy generalisation of that for monoids, but with appropriate attention to the objects of the categories concerned. For $b \in \text{Ob}(\mathbf{N})$ let $Y'(b)$ be the disjoint union of the sets $X(c) \times \mathbf{N}(F(c), b)$ for all $c \in \text{Ob}(\mathbf{M})$. On $Y'(b)$ we impose the equivalence relation generated by Eq. (2) to give $Y(b)$ as the quotient. Note that now $x \in X(c)$, $x^m \in X(c')$, $m \in \mathbf{M}(c, c')$ and $n \in \mathbf{N}(c', b)$, as in Fig. 5. The action of \mathbf{N} is induced by $(x, n)^{n'} = (x, nn')$ as before. This construction is known in category theory as that of a coend.

We now apply this construction to the double coset problem. Let Γ be a set with commuting right \mathbf{H} - and \mathbf{K} -actions, so that $(\gamma^h)^k = (\gamma^k)^h$ for all $\gamma \in \Gamma, h \in \mathbf{H}, k \in \mathbf{K}$. Intuitively we prefer to think of the \mathbf{H} -action as a left action, defining ${}^{h^{-1}}\gamma := \gamma^h$. We define a category $\mathbf{M} = \mathbf{H} \bowtie \mathbf{K}$ with objects $\{1, 2\}$ and the following elements:

$$\mathbf{M}(1, 1) = \mathbf{H}, \quad \mathbf{M}(2, 2) = \mathbf{K}, \quad \mathbf{M}(1, 2) = \mathbf{H} \times \mathbf{K}, \quad \mathbf{M}(2, 1) = \emptyset.$$

Composition in \mathbf{M} is given by the usual multiplication in \mathbf{H}, \mathbf{K} and by

$$h_2(h_1, k_1)k_2 = (h_2h_1, k_1k_2), \quad \text{so that } (h, k) = h(1, 1)k.$$

For the \mathbf{M} -action X we take $X(1) = \Gamma \times \{1\}$ and $X(2) = \Gamma \times \{2\}$ to be copies of Γ . For the morphisms we define

$$X(h)(\gamma, 1) = ({}^{h^{-1}}\gamma, 1), \quad X(k)(\gamma, 2) = (\gamma^k, 2), \quad X(h, k)(\gamma, 1) = ({}^{h^{-1}}\gamma^k, 2),$$

so $X(1, 1)$ is the isomorphism between copies of Γ mapping $(\gamma, 1)$ to $(\gamma, 2)$.

Proposition 10. *If \mathbf{N} is the trivial category with one object 0 and 1_0 the only morphism, X is the \mathbf{M} -action given above, and F is the unique functor $\mathbf{M} \rightarrow \mathbf{N}$, then $F_*(X)$ may be identified with the set of orbits $\mathbf{H} \backslash \Gamma / \mathbf{K}$.*

Proof. An N -action Y is just a set $Y(0)$ and the identity function $Y(1_0)$ on $Y(0)$. The construction above gives $Y'(0) = (T \times \{1\} \times \{1_0\}) \sqcup (T \times \{2\} \times \{1_0\})$. Applying the equivalence rule (2) with $m = h \in H$, $m = (1, 1)$ and $m = k \in K$, we obtain

$$(h^{-1}\gamma, 1, 1_0) \approx (\gamma, 1, 1_0), \quad (\gamma, 2, 1_0) \approx (\gamma, 1, 1_0), \quad (\gamma^k, 2, 1_0) \approx (\gamma, 2, 1_0),$$

identifying the two copies of T and modelling the H, K actions on T . \square

When $T = G$ and the actions are $g^h := h^{-1}g$, $g^k := gk$, then $F_*(X)$ may be identified with the set of double cosets $H \backslash G / K$.

To summarise, as with cosets and other problems in computational algebra, double cosets are an instance of the general problem of computing Kan extensions or induced actions of categories. By developing string rewriting for computing such Kan extensions, we therefore have a generic algorithm applicable to all of these problems.

8. Conclusions

One of the nice outcomes of our results is that existing powerful string rewriting software can immediately be applied to double coset problems, provided that we have a presentation for the group and generating sets for the subgroups. Of course the algorithm may not terminate, but it will in all cases where there are a finite number of cosets Hg and gK , (where the existing enumerative methods could be used), and also in some cases where there are an infinite number of double cosets and enumerative methods are likely to fail.

We have developed procedures for logged string rewriting, which via a 2-categorical structure, records all computations made from the original presentations. This enables us to compute not only whether two elements lie within the same double coset but also, using logged rewriting, to produce a proof of this in the case that they do. We expect to release the **GAP** functions to do both determining and proving via logged string rewriting as a share package.

In this paper we apply our results only to group theory, but as we showed above, they hold much more generally. It might be interesting to see whether there are analogues to the double coset problem in other structures such as monoids or algebras—structures where we also already have Kan extension rewriting methods available to us.

Acknowledgements

The authors would like to thank Steve Linton for helpful discussions, especially regarding the **GAP** implementations of the methods presented in this paper; and Tim Porter for help with applications of category theory. Thanks are also due to the referee for several helpful suggestions. Figures were typeset using **XFig**.

References

- Book, R., Otto, F., 1993. String-rewriting Systems. Springer-Verlag, New York.
- Brown, R., Heyworth, A., 2000. Using rewriting systems to compute left Kan extensions and induced actions of categories. *J. Symbolic Comput.* 29 (1), 5–31.
- Butler, G., 1981. Double cosets and searching small groups. In: Proceedings of the 4th ACM Symposium on Symbolic and Algebraic Computation. Snowbird, UT, pp. 182–187.
- Butler, G., 1991. Dimino's Algorithm. In: Lecture Notes in Comput. Sci., vol. 559. pp. 13–23.
- Cohen, D.I.A., 1991. Introduction to Computer Theory, revised edn. John Wiley and Sons Inc., New York.

- Curtis, R.T., 1992. Symmetric presentations. I. Introduction, with particular reference to the Mathieu groups M_{12} and M_{24} . In: Groups, Combinatorics & Geometry (Durham, 1990). In: London Math. Soc. Lecture Note Ser., vol. 165. Cambridge Univ. Press, Cambridge, pp. 380–396.
- Delgado, M., Linton, S., Morais, J.E., 2005. GAP package Automata. <http://cmup.fc.up.pt/cmup/mdelgado/automata/>.
- GAP, 2004. GAP — Groups, Algorithms, and Programming, version 4.4. The GAP Group. <http://www.gap-system.org>.
- Ghani, N., Heyworth, A., 2003. A rewriting alternative to Reidenmeister-Schreier. In: Rewriting Techniques and Applications. In: Lecture Notes in Comput. Sci., vol. 2706. Springer, Berlin, pp. 452–466.
- Helmkink, A.J., Brion, M., 2000. On orbit closures of symmetric subgroups in flag varieties. *Canad. J. Math.* 52 (2), 265–292.
- Heyworth, A., 1999. Applications of rewriting systems and Groebner bases to computing Kan extensions and identities among relations. Ph.D. Thesis. University of Wales, Bangor.
- Heyworth, A., Johnson, M., 2005. Logged rewriting for monoid presentations. Tech. Rep. Univ. Leicester. <http://arxiv.org/abs/math.CO/0507344>.
- Heyworth, A., Wensley, C.D., 2003. Logged rewriting and identities among relators. In: Groups St. Andrews 2001 in Oxford. vol. I. In: London Math. Soc. Lecture Note Ser., vol. 304. Cambridge Univ. Press, Cambridge, pp. 256–276.
- Heyworth, A., Wensley, C.D., 2005. Kan, version 0.91. <http://www.math.bangor.ac.uk/chda/kan/>.
- Holt, D.F., 1996. The Warwick automatic groups software. In: Geometric and computational perspectives on infinite groups (Minneapolis, MN and New Brunswick, NJ, 1994). In: DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 25. Amer. Math. Soc., Providence, RI, pp. 69–82.
- Holt, D.F., Eick, B., O'Brien, E.A., 2005. Handbook of computational group theory. In: Discrete Mathematics and its Applications (Boca Raton). Chapman and Hall/CRC, Boca Raton, FL.
- Knuth, D.E., Bendix, P.B., 1970. Simple Word Problems in Universal Algebras. In: Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967). Pergamon, Oxford, pp. 263–297.
- Laue, R., 1982. Computing double coset representatives for the generation of solvable groups. In: EUROCAM. In: Lecture Notes in Comput. Sci., vol. 144. Springer, pp. 65–70.
- Lawson, M.V., 2004. Finite Automata. Chapman and Hall/CRC, Boca Raton, FL.
- Linton, S.A., 1991. Double coset enumeration. *J. Symbolic Comput.* 12 (4–5), 415–426. Computational Group Theory, Part 2.
- Mac Lane, S., 1998. Categories for the Working Mathematician, 2nd edn. In: Graduate Texts in Mathematics, vol. 5. Springer-Verlag, New York.
- MAGMA, 2005. MAGMA, version V2.11-15. Computational Algebra Group, University of Sydney. <http://magma.maths.usyd.edu.au/magma/>.
- Pletsch, B., 2001. Investigating Young group double cosets with computer algebra. Tech. Rep. Technical Vocational Institute, Albuquerque Abstract. <http://www.jssac.org/Conference/ACA/general/pletsch.pdf>.
- Ruch, E., Klein, D.J., 1983. Double cosets in chemistry and physics. *Theoret. Chim. Acta.* 63, 447–472.
- Sims, C.C., 1994. Computation with Finitely Presented Groups. Cambridge University Press.