Société Mathématique de France
Astérisque 163-164 (1988), p. 9-29.

Computers, Groups and Hyperbolic Geometry

D.B.A. Epstein

This paper consists of notes for a course of lectures given at Rome in the summer of 1986. As such, many details of proofs are missing, and the paper is discursive, rather than following the usual style of mathematical papers, where it is often assumed that the reader is familiar with all preceding contributions to knowledge. Proofs of results have been omitted, and these are provided in [2]. The work to be explained is a very small part of a large programme, mapped out largely by W.P. Thurston. This programme of research is supported by the National Science Foundation, and is known as the Geometry Supercomputer Project. The research has also received generous support from the Science and Engineering Research Council of the United Kingdom. I thank both the N.S.F. and the S.E.R.C. for their support, and I thank Professor Ida Gasparini-Cattaneo for the invitation to speak in Rome and explain some of these ideas, even if only in sketched form. This paper has been improved as a result of careful comments by the referee (though not as much improved as the referee may have hoped).

One of the our objectives is to construct a "catalogue" of 3-dimensional manifolds. Of course, the collection of compact 3-manifolds is infinite, and we have little hope of constructing some kind of effective list. (In contrast, group theorists are able to construct an effective infinite list of simple groups, and a machine can list the positive integers in order.) But we can do something. For example, given a good test of the "complexity" of a 3-manifold, corresponding somehow to our intuitive idea of complexity, we could construct the first few, least complicated, examples. We would also like to produce "typical" or "random" 3-manifolds. As yet the concept of randomness of a 3-manifold has not been formalized in a satisfactory way. The problem of producing a normal definition, which corresponds to our intuition, seems to be a really good problem for research at the current time.

We believe that the development of such a catalogue will produce many good problems requiring solutions. In addition it should provide some good tests for Thurston's Geometrization Conjecture (see G.P. Scott [6] or W.P. Thurston [8]). It should also be useful for testing many other conjectures in 3-manifold theory. The construction of the catalogue will depend on the use of powerful computers and new programs developed for the purpose. One of the most important invariants associated with a 3-manifold is the fundamental group. It is therefore important to be able to work quickly and effectively with such groups, which will normally be given by generators and relations.

A group G with generators $\{x_1,\ldots,x_n\}$ is said to have a *solvable word problem*, if there exists an algorithm which can be applied to any word w in $\{x_1,\ldots,x_n,x_1^{-1},\ldots,x_n^{-1}\}$ giving either the answer "Yes" if $w = 1$ in G or "No" if $w \neq 1$ in G. It is a result of Novikov and Boone that there is a group $G = \{x_1,\ldots,x_n/r_1,\ldots,r_k\}$ which does not have a solvable word problem. This shows that it is not possible to write computer programs which carry out the kind of manipulations we need on general groups.

However general groups are too general to be interesting ! The groups which provide the central interest in mathematics are groups which operate on some nice structure, like a geometric space. In practice, such groups have solvable problems.

I will discuss my joint work with Jim Cannon (Brigham Young University, Provo, Utah), Derek Holt (Warwick), Mike Paterson (Warwick), Bill Thurston (Princeton), in which we produce a new algorithmic approach to certain questions in group theory. We are particularly interested in the *speed* of the algorithms, because we want practical computer programs which will produce answers in a reasonable time.

The theory has already had a practical effect. Figure 1 is a drawing of a tesselation of the hyperbolic plane by triangles with angles $\pi/2$, $\pi/3$ and $\pi/7$. My original program to compute the points and lines for this drawing took 14 hours. The current program, incorporating these new ideas, takes 5 minutes, on the same computer. The reason for this is that in the original program, the only way to check whether a line has already been drawn is to search through a list of existing edges. Apart from being very time-consuming, this procedure is subject to terrible problems from floating point inaccuracies. (Real numbers can not be kept on a computer. We can only keep approximations to such numbers. When computations are carried out, the approximations become increasingly inaccurate.) Using the ideas to be explained here, no such checking is done, because we will know, without checking, whether a particular edge has been drawn or not. In addition to a huge time saving, there is a huge potential saving on space. Since checking is not necessary there is no need to store the edges.

One can compute them and then draw them immediately.

Of course, as research workers, we are not interested in such well-understood tesselations (though they do have an aesthetic interest). But there are many drawings which are invariant under a group and which are of interest for research. Our methods should allow these drawings to be made more quickly.

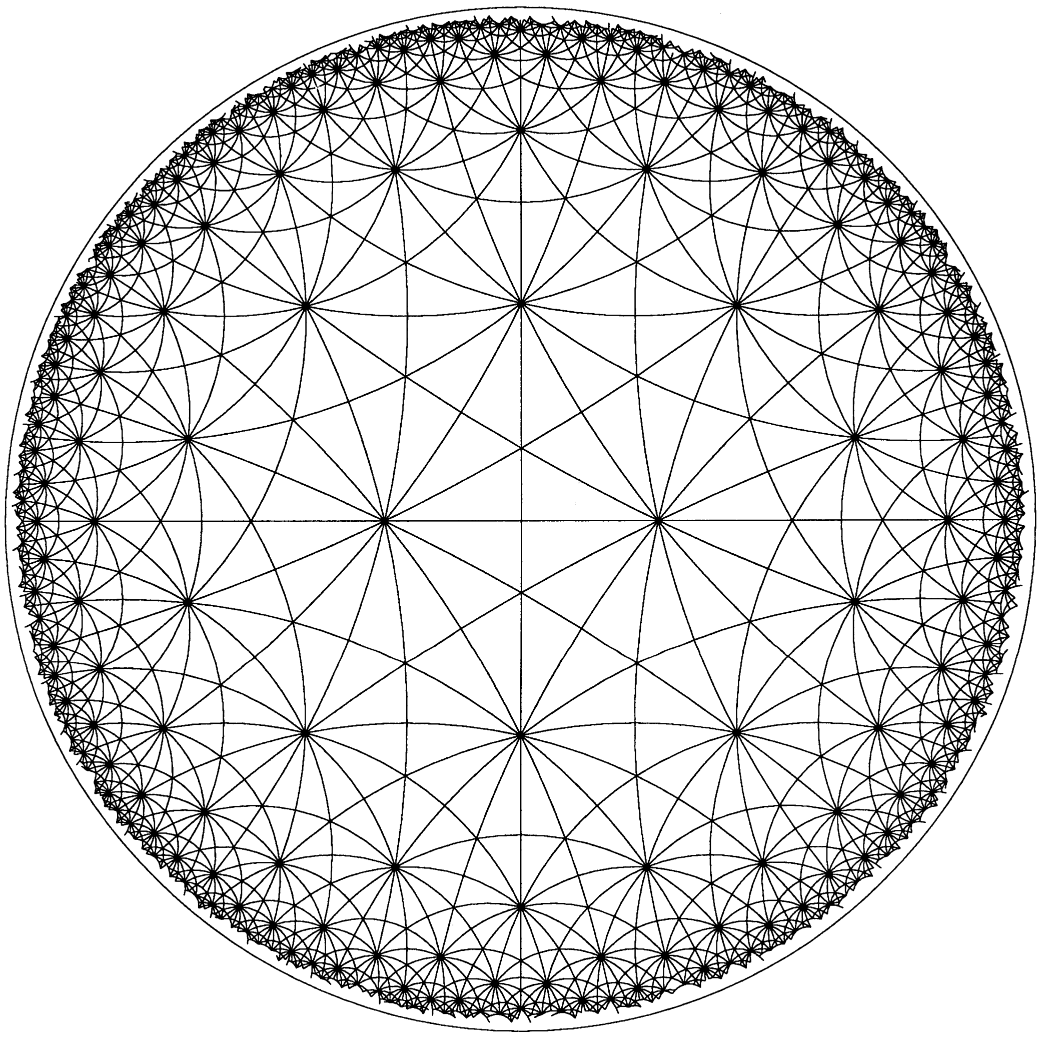


Figure 1.

Tesselation of the hyperbolic plane by the (2,3,7)-group.

14 hours $\rightarrow$ 5 minutes on a Vax 750.

§1. The Cayley Diagram of a group.

Let G be a group with generators $\{x_1,\dots,x_n\}$. We form a graph $\Gamma = \Gamma_G$ with vertices the elements of G and with edges of the form (g,x_i,h) with $g,h \in G$ and $gx_i = h$. Γ is called the *Cayley graph* or *Cayley diagram* of the group and its generators. Clearly Γ is connected. We metrize Γ by assigning each edge the length 1. A *geodesic* in Γ is a shortest path between two points. If $g \in G$, then the *length* of g, written $\ell(g)$, is the smallest value of k such that we can write

$$g = x_{i_1}^{\varepsilon_1} \dots x_{i_k}^{\varepsilon_k} \quad \text{with} \quad 1 \le i_j \le n\,, \varepsilon_j = \pm 1\,.$$

The length of g is equal to the length of a geodesic in Γ from the identity element to g.

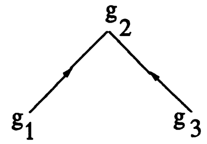
Cannon has proved some beautiful results about a discrete group G of isometries acting on hyperbolic space $\mathbb{H}^n$, with compact quotient. His results continue to be valid for any group F of isometries acting properly discontinuously on a complete Riemannian manifold $\tilde{M}$ of strictly negative curvature with compact quotient.

We choose a point $* \in \tilde{M}$ with trivial stabilizer in G. We have a canonical map $\varphi:\Gamma \to \tilde{M}$ which sends g, a vertex of Γ, to $\varphi(g) = g(*)$ and which sends the edge (g,x_i,gx_i) to the geodesic segment from $g(*)$ to $gx_i(*)$. A basic principle is that Γ (with its left G-action) approximates $\tilde{M}$ (in the sense of Gromov). However nothing of the general theory needs to be known. We use this principle merely to guide us in the formulation and the proofs. Γ gives us an essentially finite combinatorial model for $\tilde{M}$. (The combinatorial structure of Figure 1 is extraordinarily similar to the structure of $\mathbb{H}^2$, even when one pays no attention to angles and lengths of sides.) W.J. Floyd [4] has in fact proved that Γ is a combinatorial model for $\mathbb{H}^2$, by constructing the sphere at infinity in terms of the Cayley graph.

The following is a folk-lore result, possibly due to G.D. Mostow. I first learned it from one of Jim Cannon's papers.

Theorem 1.1. In the above situation, there is a number $L > 0$, such that, for any geodesic w in the Cayley graph Γ, $\varphi(w)$ is contained in the L-neighbourhood of the geodesic segment in $\tilde{M}$ from the initial point to the endpoint of $\varphi(w)$.

Note. All paths in Γ will be assumed to be maps $w:[0,\infty) \to \Gamma$ which are eventually constant, and which are parametrised by arc length on the non-constant part. Thus the path



has length 2, $w(0) = g_1$, $w(1) = g_2$, $w(2) = g_3$ and $w(t) = g_3$ for $t \geq 2$.

The *distance* between two paths is their uniform distance. Because the paths are eventually constant, any two paths are a finite distance apart. A word in $\{x_1,\ldots,x_n,x_1^{-1},\ldots,x_n^{-1}$ defines a path in Γ which starts at the identity element.

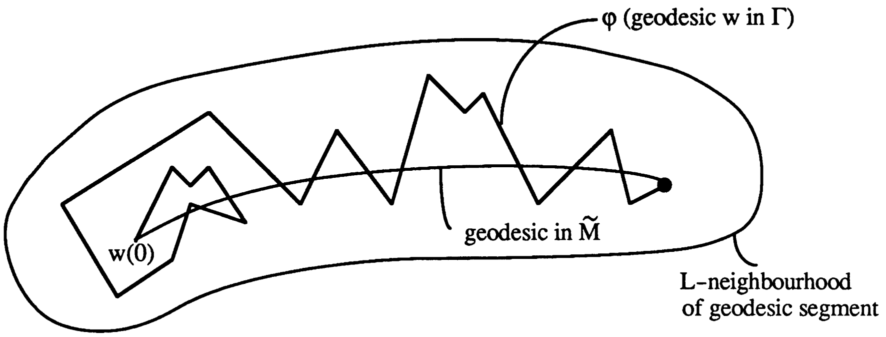


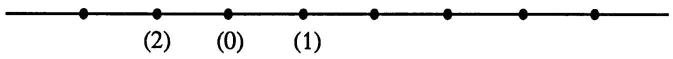
Figure 2.

It is easy to deduce from this:

Corollary 1.2. There is a number $L^1 > 0$ such that any two geodesics (shortest words) w_1, w_2, in Γ from the identity to the same point $g \in G$ satisfy $d_\Gamma(w_1,w_2) < L^1$.

This important property will be basic for our algorithmic study.

Other important ideas introduced by Cannon are the *cone-type* and the *N-type* of a vertex in Γ. The *cone from a vertex* x (denoted $[x,\infty)$) consists of all points y of Γ such that there is a geodesic from y to the identity passing through x. We say x and y have the same *cone-type* if the left multiplication by yx^{-1} takes $[x,\infty)$ to $[y,\infty)$. For example in $\mathbb{Z}$ with generator 1



there are three cone types. The cone type (0) of 0 is equal to Γ. The cone type (1) of 1 is isomorphic to all points to the right of 0 and the cone type (2) of -1 is isomorphic to all points to the left of 0.

In $\mathbb{Z} \times \mathbb{Z}$, with the obvious generators

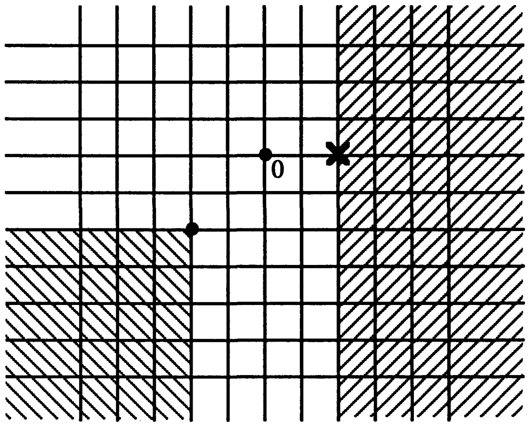


Figure 3.

there are 9 cone types. Two of these are shaded in Figure 3, namely the cone types of (2,0) and of (-2,-2).

In order to define the *N-type* of a vertex x of Γ, we take a neighbourhood of Γ of radius N around x, and label each vertex in the neighbourhood with its distance from the identity. We subtract from each of these labels the integer $d_\Gamma(\mathrm{id},x)$, so that now each element of the neighbourhood has a label between -N and N. Finally we translate the neighbourhood to B_N, the ball around the identity with radius N. Each point p of B_N is thus labelled with an integer $i_{(p)}(-N \le i_{(p)} \le N)$. The *N-type* of x is this function $i : B_N \to [-N,N]$.

Theorem 1.3. With $\tilde{M}$, G and Γ as described above, there exists an N > 0 such that if x and y have the same N-type, then they have the same cone type.

The hypotheses of this theorem do not apply to $\mathbb{Z} \times \mathbb{Z}$ (because the plane is not negatively curved), but the conclusions do apply. Some typical 1-types are shown in Figure 4.

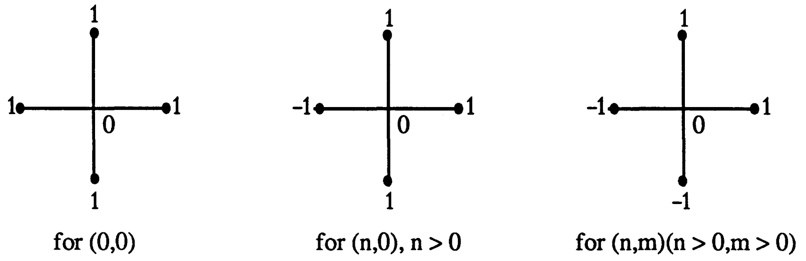


Figure 4.

Corollary 1.4. With $\tilde{M}$, G, Γ as above, there are only a finite number of cone types.

To construct a geodesic starting at the identity in Γ, we proceed step by step. Suppose a geodesic has been constructed from the identity to x. In order to continue the geodesic, it is necessary and sufficient to know the cone type of x. If the geodesic is continued by the single generator g to y,

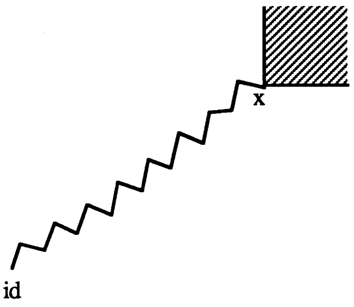


Figure 5.

then the cone type of y consists of all geodesics γ such that $g\gamma$ is in the cone type of x. Thus we arrive at an important idea

1.5	A geodesic in Γ from the identity (i.e. a sequence of shortest words in G) is constructed from a finite set (the finite set of cone types) by applying a finite set of rules (for each cone type, of x say, determine all y within the cone at a distance 1 from x and determine the cone type of each such y).

The essential element of this statement can be expressed in the language of Computer Science. "The set of shortest words of G over the alphabet $\{x_1,\ldots,x_n,x_1^{-1},\ldots,x_n^{-1}\}$ forms a *regular language*". We emphasize that this statement is only true for special groups. These groups include groups of symmetries of hyperbolic tesselations, where the tiles are compact.

The next step is to explain what a regular language is.

§2. Regular language and finite state automata.

This section is standard material in a beginning Computer Science course.

Let I be any finite set of symbols. (I is assumed not to contain the six symbols (,), |, $*$, \$, ε.) I will be called an *alphabet*. A *language* over I is any subset of I^*, the free non-abelian semigroup with identity, generated by I. We denote by ε the identitv element. The elements of I^* are words over I and ε is the empty word.

Let L_1, L_2, L be languages over I. We have the following operations on languages:

union	$L_1 \cup L_2$
concatenation	$L_1L_2 = \{w: \exists w_1 \in L_1, w_2 \in L_2 \quad \text{with} \quad w = w_1w_2\}$

We define $L^0 = \{\varepsilon\}$, a language with exactly one element. We define $L^1 = L$, and, inductively, $L^k = L^{k-1}L$ $(k > 0)$. The *Kleene closure* of L is defined to be $L^* = \bigcup_{k\geq 0} L^k$.

We now define the concept of a *regular expression* over the alphabet I, and, at the same time, we will say how a regular expression defines a language. Such a language is called *regular*. The *regular language defined by a regular expression* will be defined recursively (see below). The set of regular languages is the smallest subset of the set of all languages defined over I, containing the following languages:

the empty set,

the language $\{\varepsilon\}$ which has exactly one element,

the languages $\{x\}$ $(x \in I)$, each with exactly one element,

and such that the subset of languages is closed under union, concatenation and Kleene closure.

Let I contain n symbols. We form the set J, consisting of n+4 symbols, namely the symbols of I and, in addition, (,), |, $*$. The symbol | is pronounced "or". The symbol $*$ is pronounced "star". Every regular expression (except for the nullset) is an element of J^*, but not every element of J^* is a regular expression.

1) The empty word of J^* is a regular expression, defining the empty language.
2) $\varepsilon \in J^*$ is a regular expression, defining the language $\{\varepsilon\}$ with exactly one element.
3) Let $x \in I$. Then x is a regular expression, defining the language $\{x\} \subseteq I^*$ with exactly one element.
4) If r and s are regular expressions, defining languages R and S, then
   (r|s) defines $R \cup S$
   (rs) defines RS
   $(r*)$ defines R^*.

Brackets are, by convention, often suppressed, just as in arithmetic. This is possible because union is associative, concatenation is associative, and we regard $*$ as having higher precedence than | and juxtaposition, and juxtaposition as having higher precedence than |. Thus we can write ab$*$ | a, instead of the more precise form : ((a(b$*$)) | a).

Lemma 2.1. Given a regular language R, there is a regular expression r defining R, such that

$$r = r_1 \mid r_2 \mid \dots \mid r_k$$

where, for $1 \le i \le k$, r_i does not contain | .

Proof. We use induction on the length of r as a word in J*.

If $r = (s_1 \mid s_2)$, we are done. If $r = (s_1 s_2)$, with $s_1 = r_1 \mid \dots \mid r_k$ and $s_2 = t_1 \mid \dots \mid t_n$, then

$$s_1 s_2 = r_1 t_1 \mid r_1 t_2 \dots \mid r_k t_n \quad \text{(kn terms)}.$$

(We are writing an equality between words of J*. Technically this is incorrect. What we mean is that the languages defined are equal.) If $r = s*$ and $s = r_1 \mid \dots \mid r_k$, then

$$r = ((r_1*)(r_2*)\dots(r_k*))* .$$

Regular expressions are used in compilers and in any good quality word processor. One reason for this is that associated algorithms are extremely fast. An example of their use is the substitute command in the Unix program ED. In ED, the command s/r/w/ substitutes the element $w \in I*$ for each occurrence of a word in R, the regular language defined by r. Thus, if we want to replace each occurrence in a file of more than one consecutive blank (denoted ƀ) by a single blank, we give the command

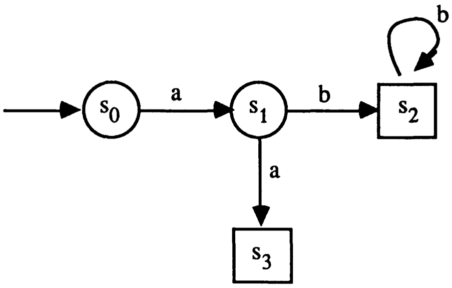
s/ƀƀƀ*/ƀ/ .

The regular expression ƀƀƀ* defines the language {ƀƀ, ƀƀƀ, ƀƀƀƀ, ...}. ED does this job surprisingly quickly, and the speed is achieved by building a finite state automaton.

A *deterministic finite state automaton* (DFA) is a machine which has a finite number of possible states. The machine reads in letters one at a time from a certain input word, and changes state according to its current state and the letter read. At the end of the input the machine either accepts the input or rejects it, according to its state.

More formally, a DFA consists of an alphabet I, a finite set S of states, an "initial" state s_0, a subset $A \subseteq S$ of "accepting" states, and a "transition function" $S \times I \to S$.

An example is given in Figure 6 :



which recognizes the language aa|abb*

Figure 6.

Here I = {a,b,c} and A consists of the two states $\{s_2,s_3\}$. It is conventional to denote accepting states by squares. Note that certain arrows have not been marked in the figure. This is because "dead-end states" have been omitted, as is conventional. A *dead-end state* is a state from which it is impossible to get to an accepting state, no matter what the input. The initial state is indicated by an unlabelled arrow.

The full state set in Figure 6 requires another state s_4, which is a dead-end state. The diagram is completed to

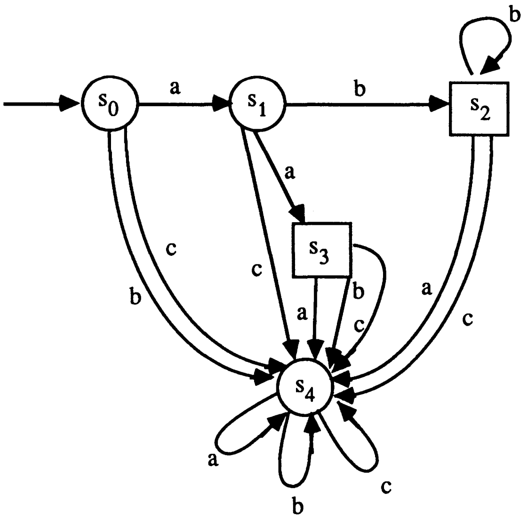


Figure 7.

Given a DFA M, we define L(M) to be the language defined by M. That is, L(M) is the set of words accepted by M. (A word is *accepted* by M if, starting from s_0, and reading from left to right, the state at the end-of-input is an accepting state.)

There is a generalization of the idea of a DFA, namely a *non-deterministic finite state automaton* NFA. Figure 8 is a drawing of an NFA. Recall that ε denotes the identity in I^*. The notation here is supposed to remind one of this, although, formally, ε is just another symbol, not

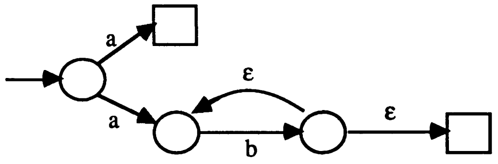


Figure 8.

already in I. The NFA of Figure 8 accepts the language a | abb*. Formally an NFA consists of a finite set S of states, a non-empty set B of initial states $B \subseteq S$, a possibly empty set A of accepting states and a transition function.

$$\theta: S \times (I \cup \{\varepsilon\}) \to 2^S$$

where 2^S is the set of all subsets of S. The connection with Figure 8 is that, given a state $s \in S$ and given $x \in I$ or $x = \varepsilon$, the subset $\theta(s,x)$ of S is the collection of all states $t \in S$ such that there is an arrow labelled x from s to t.

By a *path of arrows* in the NFA we mean a finite sequence

$$s_{i(0)}, x_{i(1)}, s_{i(1)}, \ldots, x_{i(r)}, s_{i(r)},$$

where, for $1 \leq j \leq r$, $x_{i(j)} \in I$ or $x_{i(j)} = \varepsilon$ and $s_{i(j)} \in \theta(s_{i(j-1)}, x_{i(j)})$. The path is said to *start* at $s_{i(0)}$ and *end* at $s_{i(r)}$. We say the path *factorizes* $w \in I^*$ if $w = x_{i(1)} \ldots x_{i(r)}$ when each $x_{i(j)}$ labelled ε is interpreted as the identity element in I^*. A word $w \in I^*$ is *accepted* by the NFA if there is a path of arrows factorizing w, which starts at some initial state and ends at some accepting state. Note that, since I^* is a free semigroup, if we are given w, then $x_{i(1)},\ldots,x_{i(r)}$ are determined, except for those $x_{i(j)}$ which are equal to ε.

Once again given an NFA M, we define L(M) to be the set of words accepted by M.

A fundamental result of computer science is the following theorem.

Theorem 2.2

a) Let R be a regular language. Then there is an NFA, M_1, such that $L(M_1) = R$.

b) Given an NFA M_1, there exists a DFA M_2, such that $L(M_2) = L(M_1)$.

c) Given a DFA, M_2, the language $L(M_2)$ is regular.

Proof. a) is proved by induction on the length of a regular expression. The induction starts with R equal to the empty set, or to $\{x\}$, with $x \in I$ or $x = \varepsilon$. The construction of a corresponding NFA is left to the reader.

Suppose we have regular expressions r and s corresponding to M_r and M_s. i) Then r | s corresponds to the disjoint union of the two NFA's. We take the union of their state sets, the union of their initial states, then union of their accepting states and the obvious transition function.

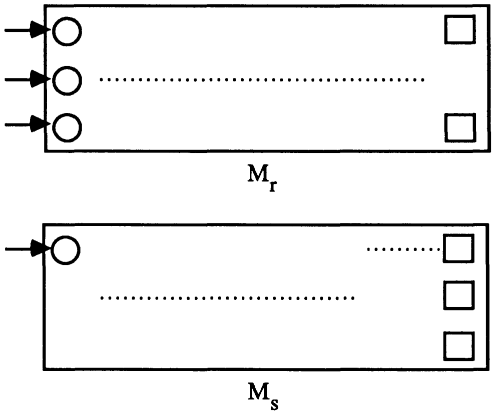


Figure 9.

ii) rs corresponds to M_r followed by M_s

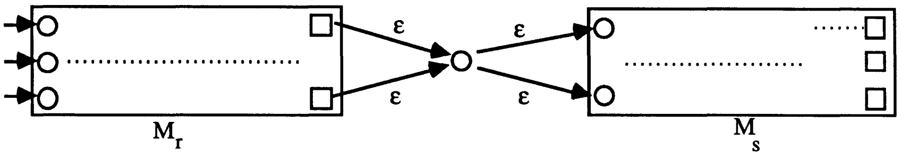


Figure 10.

The new set of initial states is equal to the set of initial states for M_r. The new set of accepting states is equal to the set of accepting states for M_s.

iii) r∗ corresponds to the NFA

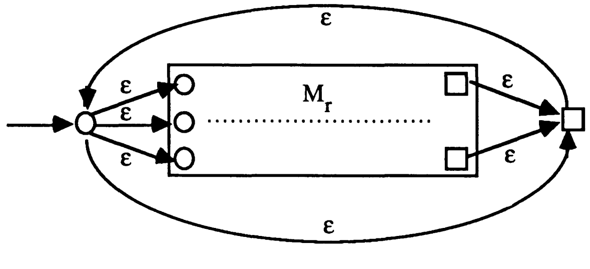


Figure 11.

There is now one initial state and one final state.

b) Let S be the state set for M_1. We define the state set for M_2 to be 2^S, the set of all subsets of S. If $B \subseteq S$ is the set of initial states for M_1, then B is the single initial state of M_2. If A is the set of accepting states for M_1, then any subset of S containing an element of A is an accepting state for M_2.

We now give the definition of the transition function θ_2 for M_2. Given $x \in I$ and a subset $T \subseteq S$ (i.e. a state of M_2), we must define $\theta_2(T,x)$ as a subset of S. For each $t \in T$ and for each $x \in I$, find all $t' \in S$ such that there is a path of arrows from t to t' factorizing x. (Any factorization of x consists of a finite number of ε's followed by x followed by a finite number of ε's.) Let $\theta_2(T,x)$ be the set of all such t'. (Note that $\theta_2(T,x)$ can be found algorithmically. Since cycles of ε's can be omitted, we may assume that the lengths of the sub-paths of ε-arrows are less than the number of states of S. Thus θ_2 can be found by a finite process.)

The fact that $L(M_1) = L(M_2)$ is clear.

To prove c), let S be the set of states of M_2. Let K be a subset of S and let $s,t \in S$. Consider the set of paths of arrows in M_2 of the form

$$s, x_{i(1)}, s_{i(1)}, \dots , x_{i(r)}, t$$

with $s_{i(1)}, \dots , s_{i(r)} \in K$. For each such path, we have a word $x_{i(1)} \dots x_{i(r)} \in I^*$. We prove by induction on the number of elements $|K|$ in K that the language $L_{s,t,K}$ is regular.

If $|K| = 0$, then $L_{s,t,K}$ is a finite subset of I, and this is a regular language. If $|K| > 0$, let $K = J \cup \{u\}$. Then

$$L_{s,t,K} = L_{s,t,J} \cup L_{s,u,J} (L_{u,u,J})^* L_{u,t,J}.$$

By the induction hypothesis, the right hand side is regular. This proves the induction.

c) now follows by setting s equal to the initial state of M_2, and taking the union of $L_{s,t,S}$ as t runs over elements of A.

§3. First Order Predicate Calculus.

Given a proposition p, whose truth depends on some parameter $w \in I^*$, we define the language L(p) to be the set of w such that p(w) is true. A typical example arises in practical computing. When the user enters a word in $I = \{0,1,\dots,9,.\}$ (I has eleven elements, the last element in the list of elements being a dot), does the word represent a real number? Here we take p(w) to mean

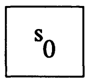
"w is a floating point number, given by the conventional English representation".

Thus p(.1.1.2..45) is false, while p(11.245) is true. In this particular example L(p) is regular.

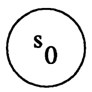
Exercise. Construct the corresponding DFA.

Any finite state automaton over I can be regarded as a proposition over I. M(w) denotes the statement that M accepts w. We call a proposition p depending on a parameter $w \in I^*$ *regular* if L(p) is a regular language, i.e. if there is a DFA M such that L(M) = L(p).

We claim that regular propositions satisfy first order predicate calculus. In order to make sense of this we need to clarify a few points. First we consider DFA's with $I = \varnothing$. Then there are no arrows, and so, without changing the language accepted, we may assume that consists of exactly one state, the initial state s_0. We obtain two possible DFA's, depending on whether the set of accepting states is empty or not.



TRUE



FALSE

The only possible input word is ε, which the DFA either rejects or accepts.

If I_1 and I_2 are two alphabets, then we wish to consider propositions depending on two variables (w_1,w_2) with $w_1 \in I_1{}^*$ and $w_2 \in I_2{}^*$. From the point of view of finite state automata, we wish to consider automata which accept or reject pairs of words. One's first inclination is to consider automata over $I_1 \times I_2$. But $(I_1 \times I_2)^*$ can be identified with the subset of $I_1{}^* \times I_2{}^*$ consisiting of pairs of words of equal length, so this would mean that one could only consider words w_1 and w_2 of the same length. This is not good enough for the applications we have in mind.

Instead we consider the alphabet $(I_1 \cup \{\$\}) \times (I_2 \cup \{\$\})$. The symbol \$ is conventionally used to denote end-of-input (i.e. end-of-word). In this new alphabet the automaton can read words like (abb,ccdde). If one presents the automaton with this pair of words, it will read in successively (a,c),(b,c),(b,d),(\$,d),(\$,e), and then stop.

We have to make sure that the automaton can only read in letters like(a,\$) or (\$,d) at the end. For this reason we define a *finite state automaton over a pair of alphabets* (I_1, I_2) to be an automaton with alphabet $(I_1 \cup \{\$\}) \times (I_2 \cup \{\$\})$ and with state set $S = S' \cup S_1 \cup S_2$ (disjoint union). Each arrow of the form (a,\$) must end in S_1 and each arrow starting in S_1 has the form (a,\$) for some $a \in I_1$. Similarly for arrows of the form (b,\$) with $b \in I_2$. There is no arrow labelled (\$,\$). This description only makes sense by virtue of omitting certain dead-end states (we are busy constructing a DFA).

Note that if $I_1 = \varnothing$ then a finite state automaton over (I_1, I_2) is virtually the same thing as a finite state automaton over I_2. (There is a slight difference formally but the languages accepted will be the same.)

Theorem 3.1. (see [3]).

a) Let p be a regular proposition over I (so that p(w) is TRUE or FALSE for each $w \in I^*$). Then $\sim$p is also a regular proposition.

b) If p_1 and p_2 are regular propositions, then so are

$$p_1 \,\&\, p_2 \,(\equiv p_1 \wedge p_2) \quad \text{and} \quad p_1 \mid p_2 \,(\equiv p_1 \vee p_2)\,.$$

c) If p is a regular proposition over (I_1, I_2) (so that $p(w_1,w_2)$ is TRUE or FALSE for each

$w_1 \in I_1{}^*$ and $w_2 \in I_2{}^*$) then

$$(\forall w_1)(p(w_1,w_2)) \quad \text{and} \quad (\exists w_1)(p(w_1,w_2))$$

are regular propositions over I_2.

Proof. a) Let M be a DFA over I representing p. Then $\sim p$ is represented by the DFA with the same set S of states and the same initial state, but with the set of accepting states equal to $S \backslash A$, where A is the set of accepting states for M.

b) Let M_i be a DFA representing p_i ($i = 1,2$). Let S_i be the state set for M_i with initial state s_i, and set of accepting states A_i. The alphabet in each case is I. We form a new DFA with state set $S_1 \times S_2$ and initial state (s_1,s_2) and alphabet I. The DFA representing $p_1 \& p_2$ is obtained by taking $A_1 \times A_2$ as the set of accepting states. The DFA representing $p_1 \mid p_2$ is obtained by taking $A_1 \times S_2 \cup S_1 \times A_2$ as the set of accepting states.

c) We have seen in a) that $\sim p$ is regular if and only if p is regular. We need only show that if p is regular, then $(\exists w_1)(p(w_1,w_2))$ is regular. This is because

$$(\forall w_1)(p(w_1,w_2)) \quad \text{is equivalent to} \quad \sim(\exists w_1)(\sim p(w_1,w_2))\,.$$

Let M correspond to p. The NFA M′ corresponding to $(\exists w_1)(p(w_1,w_2))$ is constructed as follows. We take the same set of states for M′ as for M, the same (set of) initial state(s) and the same set of accepting states. The set of arrows for M′ is also equal to the set of arrows for M, but the labels are different. A label of the form (x_1,x_2) in M with $x_1 \in I_1 \cup \{\$\}$ and $x_2 \in I_2$ is changed to the label x_2 in M′. A label of the form $(x_1,\$)$ in M with $x_1 \in I_1$ is changed to the label ε in M′.

It is easy to see that M′ corresponds to the existence statement.

It is an extremely important aspect of our theory that various types of mathematical structure can be defined using first order predicate calculus, by listing axioms. Using the preceding theorem, one can then construct a machine which will test automatically whether or not the axioms are satisfied.

My current programming work consists of producing programs which will perform the manipulations on finite state automata which correspond to the theoretical results described above. In some cases, corresponding to the writing of compilers, a very great deal is known about how to do this efficiently. In other cases some research is needed to find better algorithms than the obvious ones. The emphasis is on the *speed* of the manipulations. The description I have given already shows that there are algorithms which work. A particular problem to avoid is the exponential blow up ($S \mapsto 2^S$) in the passage from an NFA to a DFA (Theorem 2.2b)). There are examples where exponential blow up is unavoidable, but this can often be avoided in practice, using tricks from compiler writing.

§4. Automatic groups.

At last we are ready to explain the main point of these lectures. let G be a group with generators $\{x_1,\ldots,x_n\}$ (typically a group of hyperbolic isometries), and let I be the alphabet $\{x_1,\ldots,x_n,x_1^{-1},\ldots,x_n^{-1}\}$. Let $\pi:I^* \to G$ be the obvious surjective homomorphism. We say that G is an *automatic group* (with respect to the particular choice of generators) if there are (n+2) DFA's W, $M_0,M_1,\ldots,M_n$ with the following properties:

4.1.1) W has input alphabet I and

$$\{w:W(w)\} = \{w:w \text{ is accepted by } W\} \subseteq I^*$$

is mapped *onto* G by Π.

4.1.2 $M_0,M_1,\ldots,M_n$ are defined over (I, I).

4.1.3 $M_0(w_1,w_2) \Leftrightarrow (W(w_1) \ \& \ W(w_2) \ \& \ \pi w_1 = \pi w_2)$. We say "$M_0$ recognises equality".

4.1.4 $M_i(w_1,w_2) \Leftrightarrow (W(w_1) \ \& \ W(w_2) \ \& \ \pi w_2 = \pi(w_1 x_i))$ We say "M_i recognises multiplication by x_i" for $1 \le i \le n$.

Clearly M_0 is the same kind of DFA as M_i, and it is often convenient to discuss M_0 under the same heading as M_i by writing $x_0 = \varepsilon$. In practice M_0, M_1, ..., M_n have the same state set and the same initial state, differing only in the set of accepting states. But this is not part of the definition.

The data above, (G, $\{x_1,\ldots,x_n\}$, W, M_0, ..., M_n), is called an *automatic structure* on G.

The first result is the following.

Theorem 4.2. If G is automatic with respect to one set of generators, it is automatic with respect to any other (finite) set of generators.

Examples 4.3

1) Let G be the fundamental group of a compact hyperbolic manifold (or orbifold). The DFA W can be constructed using 1.5. The DFA's M_0, M_1, ..., M_n are constructed using Theorem 1.1. The language corresponding to W consists of all words in the generators which are shortest words (geodesics in the Cayley graph Γ from the identity).

2) A finite group is automatic. The state set for W is G itself. The state set for M_0, ..., M_n is G × G. The transition functions are given by right multiplication. Details are left to the reader.

3) The infinite cyclic group with generator x. The regular language corresponding to W is $x* \mid \bar{x}*$. (We write $\bar{x}$ for x^{-1}.) The equality recognizer is $(x,x)*|(\bar{x},\bar{x})*$ (equality of words in I*, in this particular case). $M_1 = M_x$ is given by $(x,x)* (\$,x) \mid (\bar{x},\bar{x})*(\bar{x},\$)$.

4) The free abelian group on 2-generators {x,y}.

$W : (x* \mid \bar{x}*)(y* \mid \bar{y}*)$ so that xyx is forbidden

M_0 : equality of words as members of I*.

$M_x : ((x,x)*(\$,x) \mid (\bar{x},\bar{x})*(\bar{x},\$))\ ((y,y))* \mid (\bar{y},\bar{y})*)$

M_y : similar to M_x.

Examples 3) and 4) indicate that an automatic structure has a close relationship with the problem of giving each element $g \in G$ a *normal form* with respect to the generators. In $\mathbb{Z}$ the normal form is x^n. In $\mathbb{Z} \times \mathbb{Z}$, the normal form is $x^n y^m$.

4.4 In a general automatic group, there are many words w, accepted by W, in the inverse image of a fixed $g \in G$. There may even be infinitely many. However, we can construct a normal form as follows.

Let $p_1(w_1, w_2)$ be the proposition "length (w_1) < length (w_2)". Let $p_2(w_1,w_2)$ be the proposition "length(w_1) = length (w_2) and w_1 occurs before w_2 in lexicographic ordering (dictionary ordering)". In order that p_2 make sense, we have to fix a linear ordering (which is arbitrary) on I. It is easy to see that both p_1 and p_2 are regular. The proposition "$w_1 = w_2$ (in I*)" is also regular. It follows from Theorem 3.1 that the proposition with parameter $w \in I^*$

$$W(w) \quad \& \quad (\forall w')(\sim M_0(w,w') \mid p_1(w,w') \mid p_2(w,w') \mid w = w')$$

is regular. Let W′ be the corresponding automaton. In words, w is recognised by W′, if and only if, for each word w′ of I* which is recognised by W and represents the same element of G, we have that w is shorter than w′, or that w has the same length and occurs before w′ in dictionary order, or that w is identical with w′. We can now change M_i to $M_i'(i=0,1,...,n)$ giving a new automatic structure on G, and now *each element of* G *corresponds to a unique word of* I* *recognized by* W′.

This gives a normal form for elements of G in terms of the generators of G. In practice the normal forms derived by the computer are often not of a form which would seem natural to a human being.

Theorem 4.5. Let $G = \{x_1,...,x_n/r_1,...,r_m\}$ be a finitely presented automatic group. Then an automatic structure
$(W, M_0, ..., M_n)$ can be found algorithmically from the generators and relations.

Theorem 4.6. There is no algorithm which can determine for all presentations $G = \{x_1,...,x_n/r_1,...,r_m\}$, whether or not G is automatic.

Thus, if one is given a group G by generators and relations, and one sets to work the algorithm of Theorem 4.5, then either the algorithm terminates giving an automatic structure for G, or else the algorithm does not terminate, and, at any finite time, we do not know, in general, if it will terminate in the future, or continue indefinitely. Of course, if the group happens to be hyperbolic, we know that the algorithm will terminate.

It is unfortunate that, given a group which we know is automatic, the only general algorithm to produce W is enormously long, impossibly long. The algorithm is as follows. The DFA's are effectively enumerable. That is, they can be listed. We just look at one example after another on the list, until we finally arrive at the correct answer. In practice, we try to construct W by looking for an automatic structure of a particular form, and this makes the job manageable in many cases. Sometimes we have special knowledge of the properties of the group (for example, we might know that it is the fundamental group of the complement of a knot in the 3-sphere), and this extra knowledge directs our search for an automatic structure.

The construction of $M_0, M_1, \ldots, M_n$ also imposes considerable problems in practice. For many examples extremely large amounts of storage may be necessary. We will have to learn by experience.

In special cases, like the one of particular interest to us, where shortest words define the regular language recognised by W, the structure can often be determined extremely quickly, in practice. For example it took only a few minutes for a VAX 750 to generate the DFA below for the (2,3,7) group, whose tesselation is shown in Figure 1. This is the group of isometries of the hyperbolic plane, generated by reflections in the sides of a hyperbolic triangle with angles $\pi/2$, $\pi/3$ and $\pi/7$. The DFA tabulated is the word acceptor. We do not yet have programs producing the other DFA's in the definition of an automatic structure.

Word acceptor DFA for (2,3,7)-group.

9 generators.

The entry in row i and column j shows the image under the transition function of the jth state and the ith generator.

initial state: 1

accepting states: all states except 0.

states:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
	0	2	0	0	0	0	12	0	0	10	0	0	0	0	0
	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	2	0	0	0	0	2	12	0	2	0	10	0	0	12
	0	3	0	0	0	0	13	3	0	12	0	3	0	0	13
	0	4	0	0	0	0	4	4	0	4	0	4	0	0	4
	0	5	0	0	0	0	5	5	0	5	0	5	0	0	5
	0	6	9	10	10	8	0	0	0	0	0	0	14	0	0
	0	7	0	7	11	0	0	0	0	0	0	0	0	14	0
	0	8	8	0	0	0	8	0	0	0	8	0	8	0	0

(The reader may be puzzled by the fact that there are 9 generators. The reason is that the element x of order 7 has been replaced by generators x, x^2, x^4. It is often more efficient, given a generator x of high order, to replace it with x, x^2, x^4, x^8,..., thus enabling us to express x^n as a product of log n generators instead of as a product of n generators.)

Given a group $G = \{x_1,...,x_n/r_1,...,r_m\}$, one starts to search for the automatic structure $(W,M_0,...,M_n)$. *How does one know when to stop?* How does one recognise success? Basically the answer is given by Theorem 3.1, and the fact that axioms for a group can be written in first order predicate calculus. First we need the following:

Proposition 4.7. . If G is an automatic group, there is an automatic structure such that W satisfies the prefix property i.e. if $W(w.x_i)$ or $W(w.\bar{x}_i)$ with $w \in I^*$, then W(w).

Proof. We first eliminate from W all dead-end states. Then we change every state of W to an accepting state. We also modify the M_i in a corresponding manner.

While searching for W, M_0, ...,M_n (using techniques which I am unable to explain here due to lack of space and time) we periodically stop and ask the computer to verify the following axioms (which are all in regular propositions, and can therefore be verified automatically).

Proposition 4.8.

(1) $(\forall x \in I, w \in I^*)(W(w.x) \Rightarrow W(w))$ (Proposition 4.7).

(2) $(\forall w)(M_0(w,w) \Leftrightarrow W(w))$ each element is equal in G to itself.

(3) $(\forall w_1,w_2,w_3)(M_0(w_1,w_2)\ \&\ M_0(w_2,w_3) \Rightarrow M_0(w_1,w_3))$ transitivity of equality in G.

(4) $(\forall w_1,w_2)(M_0(w_1,w_2) = M_0(w_2,w_1))$.

For simplicity we omit the quantifiers in the axioms which follow.

(5) $(M_0(w_1,w_2)\ \&\ M_i(w_1,w_3)) \Rightarrow M_i(w_2,w_3)$

$(M_i(w_1,w_3)\ \&\ M_i(w_2,w_3)) \Rightarrow M_0(w_1,w_2)$.

(Multiplication by a generator is well-defined.)

(6) $(W(w.x_i)$ and $W(w)) \Rightarrow M_i(w,w.x_i)$.

(7) Recall that $G = \{x_1,...,x_n/r_1,..,r_m\}$. For simplicity we suppose that the inverse X of any generator x is also a generator, and that amongst the relators appear relators of the form xX, which express this relationship. Each relator r_j can be written in the form $r_j = x_{i_1},...,x_{i_k}$. Then we have a condition for each j $(1 \leq j \leq m)$. The condition is :

$(\forall w_0,w_1,...,w_k)(N_1(w_0,w_1)$ and $N_2(w_1,w_2)...$ and $N_k(w_{k-1},w_k) \Rightarrow M_0(w_0,w_k))$,

where $N_r = M_{i_r}$. In words : the relation r_j is satisfied.

The idea is, that if these conditions are satisfied, then the search for the automatic structure is successful, and the program can report the results and exit. (This statement requires a formal proof which is given in [2]). Unfortunately, things are a little bit more complicated than this. There are more conditions to be fulfilled, of a similar nature to those already listed, but with more complicated statements. Readers who would like more detail are referred to [2], available in preprint form by writing to the author.

§5. Other results.

A number of theorems have been proved about regular groups. But the reader should be warned that even very nice groups, with a well-known solvable word problem, even nice fundamental groups of nice compact 3-manifolds, are not always regular.

Theorem. Let G be a torsion free, regular, nilpotent group. Then G is abelian.

The proof is too complicated to be given here, but it starts with the fact that nilpotent groups have polynomial growth, thus restricting the kind of regular expressions which can be used.

Another important result, in view of our basic aim of understanding and cataloguing 3-manifolds, is that the fundamental group of a link complement with a hyperbolic structure is automatic. This is useful, since most links do have hyperbolic complement (though, as already stated, we do not yet know how to define the word *most*).

One of the most important consequences of our theory is the following theorem.

Theorem. Given an automatic structure on a group, one can produce an algorithm which will reduce a word of length n to normal form in time $O(n^2)$. In particular, the word problem is soluble for an automatic group.

Note that the algorithm is fast, and this will be important in applications.

Bibliography.

[1] J.W. Cannon, *The Combinatorial Structure of Cocompact Discrete Hyperbolic Groups,* Geometriae Dedicata **16** (1984) pp.123-148.

[2] J.W. Cannon, D.F. Holt, D.B.A.Epstein, M.S. Paterson & W.P. Thurston, *Word processing and group theory,* (to appear, preprint available from D.B.A.Epstein).

[3] A. Church, *Logic, arithmetic and automata,* Proceedings of the International Congress of Mathematicians (1962).

[4] W.J. Floyd, *Group completion and limit sets of Kleinian groups,* Invent. Math, **57** (1980), pp.205-218.

[5] J.E. Hopcroft & J.D. Ullman, *Introduction to Automata Theory, Languages and Computation,* Addison-Wesley 1979.

[6] G.P. Scott, Bull. of London Math. Soc. **15** (1983), pp.401-487.

[7] V.J. Rayward-Smith, *A first course in formal language theory,* Blackwells, 1983.

[8] W.P. Thurston, B.A.M.S. **6** (1982), pp.357-381.

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