



ELSEVIER

Journal of Pure and Applied Algebra 140 (1999) 209–227

JOURNAL OF  
PURE AND  
APPLIED ALGEBRA

www.elsevier.com/locate/jpaa

## Parallel poly-pushdown groups

Gilbert Baumslag<sup>a</sup>, Michael Shapiro<sup>b,\*</sup>, Hamish Short<sup>c</sup>

<sup>a</sup>City College of New York, New York, NY 10031, USA

<sup>b</sup>University of Melbourne, Parkville, Vic 3052, Australia

<sup>c</sup>LATP, Centre de Mathématiques et d'Informatique, Rue Joliot-Curie, Université de Provence,  
F-13453, Marseille cedex 13, France

Communicated by J. Rhodes; received 21 April 1997; received in revised form 22 October 1997

---

### Abstract

We define a class of groups based on parallel computations by pushdown automata. This class generalizes automatic groups. It includes the fundamental groups of all 3-manifolds which obey Thurston's geometrization conjecture. It also includes nilpotent groups of arbitrary class and polynomial degree isoperimetric inequality. It is closed under wreath product, and so contains many groups which are not finitely presented. © 1999 Elsevier Science B.V. All rights reserved.

*AMS Classification:* 20F32

---

### 1. Introduction

The theory of automatic groups has attracted wide attention for the last ten years or so and brought together a large and varied collection of groups. These groups admit a simple description in terms of finite state automata – we refer the reader to Section 2 for details and the various terms that we shall make use of in this introduction.

One of the original motivations for this theory was to provide a means for computing in the fundamental groups of geometric 3-manifolds. The goal was, in part, accomplished by Epstein et al. [3] and Shapiro [8] when they, independently, prove the following theorem: if  $M$  is a 3-manifold that obeys Thurston's geometrization conjecture, then  $\pi_1(M)$  is automatic if and only if  $M$  does not contain a closed Nil or Sol manifold in its connected sum decomposition.

One generalization of the class of automatic groups is the class of asynchronously automatic groups. While this class successfully captures such non-automatic groups

---

\* Corresponding author. Current address: College of Staten Island, City University of New York, New York, NY 10314, USA.

*E-mail address:* rev@groups.sci.cuny.cuny.edu (G. Baumslag)

such as the Baumslag–Solitar groups [2,6] it still does not include the fundamental groups of Nil [6] and Sol [3] manifolds.

Bridson has shown that the fundamental groups of Nil and Sol manifolds have asynchronous combings. That is to say, one can find a normal form in such a group with the so-called asynchronous fellow traveler property. Since the groups in question are not asynchronously automatic, the language of such a normal form is not the language of a finite state automaton. Bridson and Gilman [5] have investigated the computational complexity of these asynchronous combings. They show that they cannot be discovered by means of a pushdown automaton, but they can be discovered by means of a nested stack automaton.

Here we shall take a slightly different approach. Motivated by the notion of computations being carried out in parallel, we allow for the use of as many pushdown automata as we need, in order to describe a variety of different groups. This gives rise to a class of languages that we call *parallel poly-pushdown*, allowing us to generalize automatic groups to *parallel poly-pushdown groups*. Passing from finite state automata to pushdown automata increases the space costs from a bounded amount of memory to a linearly bounded amount of memory. However, this does not increase time costs. Each machine still processes its input in linear time. Thus, from a computational point of view, little is lost.

The resulting class of parallel poly-pushdown groups, which we shall denote here simply by  $\mathcal{P}$ , has a number of properties in common with automatic groups. For example, we shall prove that both the free and the direct product of two groups in  $\mathcal{P}$  is again in  $\mathcal{P}$ . In addition, we shall prove that such parallel poly-pushdown groups are recursively presentable. In fact even more is true

**Theorem 4.2.**  *$\mathcal{P}$ -groups have solvable word problem.*

The similarity between the two classes ends here. One of the most interesting points of departure, which underlines the significance of Theorem 4.2, is a consequence of the following.

**Theorem 5.4.** *The standard wreath product  $U \wr T$  of the parallel poly-pushdown group  $U$  by the parallel poly-pushdown group  $T$  is again a parallel poly-pushdown group.*

Now, the wreath product  $W$  of a finitely presented group  $U$  by a finitely presented group  $T$  is finitely presented if and only if either  $U = 1$  or  $T$  is finite [1]. Since the infinite cyclic group  $C$  belongs to  $\mathcal{P}$ , it follows that the finitely generated but not finitely presented group  $C \wr C$  belongs to  $\mathcal{P}$ , i.e.,  $\mathcal{P}$  is not contained in the class of finitely presented groups, in direct contrast to the class of automatic groups.

Now a finitely generated nilpotent group is automatic if and only if it is virtually abelian [6]. Again we have a pronounced difference between the two classes of groups:

**Theorem 5.2.** *Let  $A$  be a finitely generated central subgroup of the group  $G$ . If  $G/A$  is automatic, then  $G \in \mathcal{P}$ .*

It follows thence that:

**Corollary 5.3.** *Every finitely generated nilpotent group of class at most two is parallel poly-pushdown.*

We have been unable to decide whether every finitely generated nilpotent group is parallel poly-pushdown. However, we have been able to prove

**Theorem 5.6.** *Suppose that  $H$  is a  $\mathcal{P}$  group, and  $\varphi: H \rightarrow \text{Aut } \mathbb{Z}^n$ . Then  $\mathbb{Z}^n \rtimes_{\varphi} H$  is  $\mathcal{P}$ .*

It follows that there are parallel poly-pushdown groups which are nilpotent of arbitrary class. Indeed, every finitely generated torsion-free nilpotent group embeds in a parallel poly-pushdown group, as we see from

**Theorem 6.1.** *Let  $n$  be any positive integer. Then the group of all integral upper uni-triangular matrices of degree  $n$ , is parallel poly-pushdown.*

This implies that every finitely generated torsion-free nilpotent group is a subgroup of a parallel poly-pushdown group, namely the above (nilpotent) group of matrices.

Finally, we remark (Corollary 5.10) that  $\mathcal{P}$  contains the fundamental groups of all 3-manifolds which obey Thurston's geometrization conjecture.

## 2. Preliminaries

Finite state automata and regular languages are now fairly well known to geometric group theorists. We record the basic definitions for completeness.

Given a finite set  $A = \{a_1, \dots, a_k\}$  the free monoid on  $A$  is denoted  $A^*$ . Thus  $A^*$  consists of strings  $w = a_{i_1} \dots a_{i_n}$  where  $a_{i_j} \in A$  and  $n \geq 0$ . The elements of  $A$  are called the *letters* of the *alphabet*  $A$ , and the elements of  $A^*$  are called *words*. The *length* of the word  $w = a_{i_1} \dots a_{i_n}$  is  $n$ , written  $\ell(w) = n$ . If  $n = 0$  then  $w$  is the *empty word* denoted by  $\varepsilon$ . A subset of  $A^*$  is called a *language*.

A finite state automaton is an idealization of a machine which has an input tape and finite amount of internal memory (and hence finitely many states that memory can be in). The automaton reads the input tape and changes its state according to its current state, together with the letter just read from the tape. More formally, a *finite state automaton* over the alphabet  $A$  is a quintuple  $(S, A, \tau, s_0, Y)$  where  $S$  is a finite set of *states*,  $s_0 \in S$  is the *start state*,  $\tau: S \times A \rightarrow S$  is the *transition function* and  $Y \subset S$  is the set of *accept states*. A finite state automaton  $M$  determines a language  $L(M) \subset A^*$  in the following way. For each word  $w = a_{i_1} \dots a_{i_n} \in A^*$ , let  $t_0 = s_0$  and for  $1 \leq j \leq n$  let take  $t_j = \tau(t_{j-1}, a_{i_j})$ . Now let

$$L(M) = \{w = a_{i_1} \dots a_{i_n} \in A^* \mid n \geq 0, t_n \in Y\}.$$

A language is *regular* if it is the language of some finite state automaton.

The *concatenation* of two words  $u$  and  $v$  in  $A^*$  is their product  $uv$ . Given two languages  $L \subset A^*$  and  $M \subset A^*$ , the concatenation of these languages is

$$LM = \{uv \mid u \in L, v \in M\}.$$

The Kleene star of a language  $L$  defined to be

$$L^* = \{\varepsilon\} \cup L \cup LL \cup LLL \dots$$

(recall that  $\varepsilon$  denotes the empty word.) The class of regular languages is closed under the set operations of union, intersection and complementation and under the language theoretic operations of concatenation and Kleene star. In fact the class of regular languages is the smallest class of languages which contains all finite languages and is closed under these operations.

Pushdown automata are somewhat less common in geometric group theory. Roughly speaking, a pushdown automaton is a finite state automaton which also has control of a stack. It can write to the top of the stack and erase from the top of the stack. At any point in a computation it chooses its next move based on its current internal state, the top letter of the stack and possibly a letter read from its input tape. More formally, a *deterministic pushdown automaton* is a 7-tuple  $(S, A, B, \tau, s_0, Z_0, Y)$  where

- (1)  $S$  is a finite set of *states*.
- (2)  $A$  is a finite *input alphabet*.
- (3)  $B$  is a finite *stack alphabet*.
- (4) The transition function  $\tau$  is a map from a subset  $\Sigma$  of  $(A \cup \{\varepsilon\}) \times B \times S$  to  $S \times B^*$ .
- (5)  $s_0 \in S$  is the *start state*.
- (6)  $Z_0 \in B$  is the *start symbol*.
- (7)  $Y \subset S$  is the set of *final states*.

Further conditions on the set  $\Sigma$  will be given below.

We now describe how the automaton  $M = (S, A, B, \tau, s_0, Z_0, Y)$  accepts or rejects a word  $w \in A^*$ . At any point in a computation, the current unread content of the input tape is a word  $u \in A^*$ , the current state is an element  $s \in S$  and the current content of the stack is a word  $z \in B^*$ . The *instantaneous description* of this state of affairs is the triple  $(u, z, s)$ . Suppose that  $u = av$  with  $a \in A \cup \varepsilon$  and  $z = yb$  with  $b \in B$ . Then  $M$  makes the transition  $(av, yb, s) \xrightarrow{M} (v, yz, t)$  where  $(t, \alpha) = \tau(a, b, s)$ . That is,  $M$  reads the first letter (or no letter) of its remaining input, replaces the top letter of its stack with a word (possibly the empty word, possibly the letter just erased) and makes a transition to a new internal state. To ensure that the operation of  $M$  is deterministic, i.e., that  $M$  has exactly one possible transition for each instantaneous description, the following requirement is imposed on  $\Sigma$ . Suppose that  $b$  is a stack letter and  $s$  is a state. Then either (i)  $(\varepsilon, b, s) \in \Sigma$ , and  $(a, b, s) \notin \Sigma$  for  $a \in A$ , or (ii)  $(\varepsilon, b, s) \notin \Sigma$ , and  $(a, b, s) \in \Sigma$  for all  $a \in A$ .

There are several ways to define what it means for  $M$  to accept a word. Let  $\xrightarrow{M^*}$  be the reflexive and transitive closure of  $\xrightarrow{M}$ . We say  $M$  *accepts*  $w$  *by empty stack*

if  $(w, Z_0, s_0) \xrightarrow{M^*} (\varepsilon, \varepsilon, s)$ . We say  $M$  accepts  $w$  by final state if  $(w, Z_0, s_0) \xrightarrow{M^*} (\varepsilon, z, s)$  with  $s \in Y$ . These two notions are equivalent in the following sense.

**Proposition 2.1.** *Let  $L \subset A^*$ . Then there is a deterministic pushdown automaton  $M$  such that  $L$  is the set of words which  $M$  accepts by empty stack if and only there is a deterministic pushdown automaton  $M'$  so that  $L$  is the set of words which  $M'$  accepts by final state.*

If  $L$  is a language determined by a deterministic pushdown automaton in either of these ways, then  $L$  is said to be *the language of a deterministic pushdown automaton* (or  $L$  is a deterministic context-free language). In view of Proposition 2.1, the automata constructed in this article will use either acceptance by empty stack or acceptance by final state, as seems most convenient. We shall usually not provide an explicit construction of the states, transition function, etc. However, we hope that our descriptions will be sufficiently clear in each case for the dedicated reader to be able to construct explicitly the corresponding automaton as a straightforward (if tedious) exercise.

The membership problem for deterministic pushdown languages can be solved in an efficient manner in the following sense:

**Proposition 2.2.** *Let  $L \subset A^*$  be the language accepted by a deterministic pushdown automaton.*

*There is an algorithm to decide whether or not words  $w \in A^*$  lie in  $L$  taking time proportional to the length of the word  $w$  considered.*

When  $L$  is the language of a general non-deterministic pushdown automaton, then one can prove that there exists such an algorithm taking time proportional to the cube of the length of  $w$ . For proofs of these results, and a general treatment of pushdown automata (context-free grammars etc.), see for instance [7] (especially chapters 5, 6 and 10). The languages studied in this article are the intersections of finitely many deterministic pushdown languages, so the same efficiency result holds.

One simple use of a pushdown stack is as a counter. We can use the stack to keep track of an element of  $\mathbb{Z}$  in the following manner. The number  $n \geq 0$  might be represented by placing  $n$  copies of some symbol (say  $+1$ ) on the stack. The number  $-n \leq 0$  might be represented by placing  $n$  copies of some symbol (say  $-1$ ) on the stack. If we choose some bound  $C$  in advance then we can build a pushdown automaton  $M$  which is capable of adding any number  $m$ ,  $-C \leq m \leq C$  to the current contents of the stack. If  $m$  and the stack contents  $n$  have the same sign,  $M$  pushes  $|m|$  copies of the appropriate symbol onto the stack. If  $m$  and the stack contents  $n$  differ in sign,  $M$  begins by popping from the stack. It continues until it has either done this  $|m|$  times or has exhausted the stack. If it exhausts the stack before popping  $|m|$  times, it continues by pushing the appropriate number of symbols of the opposite sign onto the stack. Since  $|m|$  is bounded, all of this can be programmed into the states of  $M$  to be performed upon consuming a single letter of input.

Similarly,  $M$  can be designed so that if the number  $n$  is contained in the stack and  $M$  encounters a string  $x^m$  as input, then  $M$  can consume the string  $x^m$  and add  $m$  to the stack.

We make the following definition:

**Definition.** A language  $L \subset A^*$  is said to be *parallel poly pushdown* ( $\mathcal{P}$ ) if there are finitely many languages  $L_i, i = 1, \dots, k$  of deterministic pushdown automata, such that  $L = \bigcap_{i=1}^k L_i$ .

Notice that the class of deterministic pushdown languages is not closed under the operation of intersection, as is shown by the example considered in [7], Section 6.2:

$$\{a^i b^i c^i \mid i > 0\} = \{a^i b^i c^j \mid i, j > 0\} \cap \{a^j b^i c^i \mid i, j > 0\}.$$

Our aim is to use the class of  $\mathcal{P}$  languages to describe normal forms in certain groups. To describe multiplication in the groups concerned, following the ideas used in the theory of automatic groups, we describe a method for deciding whether or not two given words in normal form correspond to group elements which differ by multiplication on the right by a generator. This leads to the notion of an asynchronous two-tape parallel poly-pushdown language.

Given an alphabet  $A$ , let  $A^\#$  and  $A^\flat$  be disjoint sets of the same cardinality as  $A$ , with isomorphisms  $u: A^* \rightarrow A^{\#\#}$  and  $v: A^* \rightarrow A^{\flat\flat}$  denoted by  $u \mapsto u^\#$  and  $v \mapsto v^\flat$ . Given a pair of words  $(u, v) \in A^* \times A^*$ , a *shuffle*,  $\sigma(u, v)$  is defined to be a word  $u_1^\# v_1^\flat \dots u_k^\# v_k^\flat \in (A^\# \amalg A^\flat)^*$  so that  $u = u_1 \dots u_k, v = v_1 \dots v_k$  and  $u_i \neq \varepsilon$  if  $i > 1$ , and  $v_j \neq \varepsilon$  if  $j < k$ . In spite of the notation  $\sigma$  is not a function.

Let  $\$$  be a symbol which is not in the alphabet  $A$ . An *asynchronous two-tape pushdown automaton* over the alphabet  $A$  is a deterministic pushdown automaton whose input alphabet is  $(A \cup \{\#\})^\# \amalg (A \cup \{\#\})^\flat$ , and whose states are partitioned into two disjoint sets  $S_\#$  and  $S_\flat$ . Given an asynchronous two-tape pushdown automaton  $T$  over the alphabet  $A$ , and a pair of words  $(u, v) \in A^* \times A^*$ , we say that  $(u, v)$  is *accepted* by  $T$  if there is a shuffle  $\sigma = \sigma(u\$, v\$)$  so that  $\sigma$  is accepted by  $T$  and when reading  $\sigma$ ,  $T$  reads only  $(A \cup \{\#\})^\#$  letters while in  $S_\#$  states and only  $(A \cup \{\#\})^\flat$  letters while in  $S_\flat$  states. We call such a  $\sigma$  an *acceptance* of  $(u, v)$ . The deterministic nature of the automaton  $T$  has the following consequence for the efficiency of  $T$ :

**Lemma 2.3.** *For each pair  $(u, v) \in A^* \times A^*$ , there is at most one shuffle  $\sigma = \sigma(u\$, v\$)$  which is accepted by  $T$ . Moreover, there is an algorithm taking time proportional to the sum of the lengths of the words  $u, v$ , which determines whether or not such a shuffle exists.*

We say that the *language* of  $T$  is the set of pairs accepted by  $T$ . Strictly speaking  $T$  cannot read an  $A^\#$  letter while in a  $S_\flat$  state, nor can it read an  $A^\flat$  letter while in a  $S_\#$ . Speaking colloquially, we will say that  $T$  “goes to a fail state” in such a situation.

We say that a subset of  $A^* \times A^*$  is an *asynchronous two-tape parallel poly-pushdown language* (AT $\mathcal{P}$ ) if it is the intersection of the languages of finitely many asynchronous two-tape pushdown automata. We emphasize that these machines are taken to be deterministic. It follows immediately from Lemma 2.3 that membership of such a language can be determined by an algorithm taking time proportional to the sum of the lengths of the pair of words involved.

It seems likely that if we pass to nondeterministic machines the analogous algorithms could take exponential time.

### 3. Elementary properties of $\mathcal{P}$ languages

**Proposition 3.1.** *Suppose that  $L$  and  $M$  are  $\mathcal{P}$ , and that  $R$  is regular. Then the following are  $\mathcal{P}$ :*

(1)  $L \cap M$ .

(2)  $L \cup R$ .

(3)  $L - R$ .

(4) *If  $M$  and  $L$  are  $\mathcal{P}$  languages over disjoint alphabets, and  $\varepsilon \notin L \cup M$ , then  $LM$  and  $(LM)^*$  are  $\mathcal{P}$ .*

*If  $M$  and  $L$  are  $\mathcal{P}$  languages over disjoint alphabets, and  $\varepsilon \in L \cap M$ , then*

$$M(L - \{\varepsilon\}M - \{\varepsilon\})^*L$$

*is  $\mathcal{P}$ .*

**Proof.** (1) We have  $L = \bigcap_{i=1}^m L_i$ ,  $M = \bigcap_{i=1}^n M_i$ , so  $L \cap M = (\bigcap_{i=1}^m L_i) \cap (\bigcap_{i=1}^n M_i)$ .

(2) We have  $L \cup R = \bigcap_{i=1}^m (L_i \cup R)$ . But the union of a deterministic pushdown language with a regular language is itself a deterministic pushdown language (see [7]).

(3)  $L - R = L \cap (A^* - R)$ . But  $A^* - R$  is regular, hence deterministic pushdown, so we are done.

(4) We have  $L = \bigcap_{i=1}^m L_i$ ,  $M = \bigcap_{i=1}^n M_i$ , where  $L_i$  and  $M_i$  are the languages of deterministic pushdown automata  $P_i$  and  $Q_i$  (we suppose that these automata accept by final state). Notice that  $LM = \bigcap_{i,j} L_i M_j$ . Thus, it suffices to construct a deterministic pushdown automaton  $S_{ij}$  which accepts the language  $L_i M_j$ .  $S_{ij}$  starts in the start state of  $P_i$  and continues with the operation of  $P_i$  until it encounters a generator from the alphabet of  $M$ . If it is not in an accept state of  $P_i$ , it goes to a fail state and reads the rest of the tape. If it is in an accept state of  $P_i$ , it empties the stack, and goes into the start state of  $Q_j$ . The automaton  $S_{ij}$  accepts if and only if it ends in an accept state of  $Q_j$ .

The same construction generalizes to  $(LM)^*$ , as the conditions given on the languages ensure that  $((\bigcap_{i=1}^m L_i)(\bigcap_{j=1}^n M_j))^* = \bigcap (L_i M_j)^*$ .

For the final more complicated case, we suppose that there is no  $\varepsilon$  transition from the initial states of the machines used to define  $L$  and  $M$ . We introduce a new start state, which, if the first letter seen lies in the alphabet of  $M$ , then the automaton

reacts is if it had been in the start state of the machine for  $M$ , otherwise it reacts as if it had been in the start state of the machine for  $L$ .  $\square$

The interested reader can consult Hopcroft and Ullman’s book [7] for other properties of (deterministic) pushdown automata and their languages.

#### 4. $\mathcal{P}$ groups

Let  $G$  be a group, and let  $A$  be a finite monoid generating set, i.e., a finite set equipped with a map from  $A \rightarrow G$ , such that the induced monoid homomorphism  $A^* \rightarrow G$  is surjective. The homomorphism  $A^* \rightarrow G$  is here denoted by  $w \mapsto \bar{w}$ . A language  $L \subset A^*$  is a *parallel poly-pushdown structure* for  $G$  if

- (1) the map  $L \mapsto \bar{L} = G$  is a bijection,
- (2)  $L$  is  $\mathcal{P}$ ,
- (3) for each  $a \in A$ ,  $\{(w, w') \in L \times L \mid \bar{w}' = \bar{w}a\}$  is  $\text{AT}\mathcal{P}$ .

We say  $G$  is *parallel poly-pushdown* ( $\mathcal{P}$ ) if  $G$  has a parallel poly-pushdown structure.

**Remarks.** (1) Concerning condition 3, notice that it is easy to check whether or not a given pair of words lies in  $L \times L$ . Thus, when checking this condition, we will always assume we are given a pair in  $L \times L$  and concentrate on checking equality.

(2) If  $G$  is a  $\mathcal{P}$  group, and  $B$  is any finite monoid generating set for  $G$ , we do not know whether or not there is a  $\mathcal{P}$  structure for  $G$  with this generating set.

(3) Clearly, the synchronous and asynchronous automatic groups of [6] are  $\mathcal{P}$  (with respect to all finite generating sets).

(4) Without loss of generality, we can take  $A$  to be closed under inverses. We need only check that if  $\{(w, w') \in L \times L \mid \bar{w}' = \bar{w}a\}$  is  $\text{AT}\mathcal{P}$ , then so is  $\{(w, w') \in L \times L \mid \bar{w}' = \overline{wa^{-1}}\}$ . But this is exactly the same set of pairs taken in the opposite order, so it suffices to unplug the two input tapes and plug them back into each other’s sockets!

**Proposition 4.1.** *If  $G$  is  $\mathcal{P}$  then  $G$  has a  $\mathcal{P}$  structure in which the identity is represented by the empty word.*

**Proof.** Suppose that  $L'$  is a  $\mathcal{P}$  structure for  $G$  and that  $w \in L'$  is the representative for the identity. We claim  $L = L' - \{w\} \cup \{\varepsilon\}$  is also a  $\mathcal{P}$  structure for  $G$ . Clearly,  $L$  is  $\mathcal{P}$ , by Proposition 3.1. Now suppose  $L'_a = \{(w, w') \in L' \times L' \mid \bar{w}' = \bar{w}a\}$  and that for each  $a \in A$ , the representatives of  $\bar{w}a$  and  $\bar{a}^{-1}$  are  $u_a$  and  $v_a$ . Then using  $L_a$  to represent  $\{(w, w') \in L \times L \mid \bar{w}' = \bar{w}a\}$ , we have that  $L_a = L'_a - \{(w', u_a), (v_a, w')\} \cup \{(\varepsilon, u_a), (v_a, \varepsilon)\}$ . As before, it is easy to see that this is  $\text{AT}\mathcal{P}$ .  $\square$

**Theorem 4.2.** *Suppose  $G$  is  $\mathcal{P}$ . Then  $G$  has a solvable word problem.*

**Proof.** The proof of this is general nonsense. Solvability of the word problem is independent of generating set. We choose a generating set  $A$  so that  $L \subset A^*$  is a  $\mathcal{P}$  structure. We have now organized things so that

(1)  $L$  is a recursively enumerable language which surjects to  $G$ . (In fact, it is a recursive language which bijects to  $G$ .)

(2) For each generator  $a$  the subset of  $L \times L$  which denote  $a$ -edges is recursive.

(3)  $L_1$ , the set of normal form words for the identity, is recursive. (In fact, it is the language containing only the empty word.)

We are given the word  $w = a_1 \dots a_k$  and asked to determine if  $\bar{w} = 1$ . We let  $w_0$  be a normal form word for the identity. We assume that  $j \leq k$  and that we have found  $w_{j-1}$ , a normal form word for  $\overline{a_1 \dots a_{j-1}}$ . If  $j < k$ , we enumerate the normal form words of  $L$  and for each word  $v \in L$  of this enumeration, we test whether or not  $(w_{j-1}, v)$  denotes an  $a_j$ -edge. We will eventually find such a  $v$ , and when we do, we take it as  $w_j$ . If  $j = k$ , we test whether  $w_j \in L_1$ . As  $\overline{w_k} = \bar{w}$  this determines whether or not  $w$  evaluates to the identity.  $\square$

The efficiency of this algorithm depends on the efficiency of the enumeration of (1) and the decision procedures of (2) and (3). For  $\mathcal{P}$  structures, as with automatic structures, deciding (1) or (2) can be done in a length of time proportional to the length of the proposed word or words. Likewise the decision entailed in (3) takes place in one step. In the case of an automatic structure, producing  $w_j$  from  $w_{j-1}$  is highly efficient. It can be done in linear time, and this gives rise to a quadratic time algorithm for solving the word problem in an automatic group. This final estimate depends on the fact that the length of a normal form word is linear in the length of the element it represents.

For an asynchronously automatic group, this algorithm can rise to exponential time for the simple reason that the length of  $w_j$  may be exponential in  $j$ . Since  $\mathcal{P}$  groups include the asynchronously automatic groups, this algorithm can be at least that bad for  $\mathcal{P}$  groups. At present we do not know how to bound the length of a  $\mathcal{P}$  normal form word for a group element of length  $k$ . This would shed some light on the efficiency of this algorithm for  $\mathcal{P}$  groups.

It follows from Theorem 4.2 that  $\mathcal{P}$  groups are recursively presented. We will see below (Corollary 5.5) that they are not necessarily finitely presented.

## 5. Closure properties of $\mathcal{P}$ groups

**Theorem 5.1.** *The set of  $\mathcal{P}$  groups is closed under direct product and free product.*

**Proof.** We suppose that  $G$  and  $G'$  are  $\mathcal{P}$  groups and that  $L \subset A^*$  and  $L' \subset A'^*$  are  $\mathcal{P}$  structures for  $G$  and  $G'$ , respectively, with disjoint finite alphabets  $A, A'$ .

We claim that  $M = LL' \subset (A \cup A')^*$  is a  $\mathcal{P}$  structure for the direct product  $G \times G'$ . Map  $A$  and  $A'$  to the images of  $\bar{A}$  and  $\bar{A}'$  under their natural inclusions into  $G \times G'$ . In Proposition 3.1 it is shown that  $M$  is  $\mathcal{P}$ , and the induced map  $M \rightarrow G \times G'$  is clearly bijective. It remains to show that, for each  $a \in A \cup A'$ ,  $\{(w, w') \in M \times M \mid \overline{w} = \overline{w'a}\}$  is  $AT\mathcal{P}$ .

Suppose  $a \in A'$ . Since  $L'$  is a  $\mathcal{P}$  structure for  $G'$  there is a collection of asynchronous two-tape pushdown automata which taken together define  $L'' = \{(w, w') \in L' \times L' \mid \overline{w'} = \overline{w}a\}$ . Modify each of these machines to look initially for the diagonal in  $A^* \times A^*$ , followed by the representatives of the elements of  $L''$  when the first letter in  $A'$  appears. A similar argument applies when  $a \in A$ .

Now consider the free product. We can assume that  $L$  and  $L'$  contain the empty word as their respective representatives for 1. The language  $N = L(L'^-L^-) * L'$  is  $\mathcal{P}$ , by Proposition 3.1, and it clearly maps bijectively to  $G * G'$ . As usual, we must check that for each  $a \in A \cup A'$ ,  $\{(w, w') \in N \times N \mid \overline{w'} = \overline{w}a\}$  is  $\text{AT}\mathcal{P}$ . Suppose that  $a \in A$ . We can assume  $\bar{a} \neq 1$  for otherwise there is nothing to prove. Let  $M_1, \dots, M_k$  be the asynchronous two-tape pushdown machines that determine the corresponding language for  $G$ . We describe machines  $M_0, M'_1, \dots, M'_k$  that determine  $\{(w, w') \in N \times N \mid \overline{w'} = \overline{w}a\}$ .

Now  $w$  and  $w'$  must have the form  $w = u_1v_1 \dots u_lv_l$  and  $w' = u'_1v'_1 \dots u'_jv'_j$ . Further, if  $\overline{w'} = \overline{w}a$ , we have  $|l - j| \leq 1$ . We use an asynchronous two-tape finite state automaton  $M_0$  to check that, except for the last  $G$  factor, we have  $u_i = u'_i$  and  $v_i = v'_i$ , and that for the last  $G$  factor  $u_i \neq u'_i$ .

Each  $M'_i$  operates as follows. It starts by reading the two initial  $L$  portions of  $w$  and  $w'$  emulating the action of  $M_i$ . Upon encountering a change of generating set, it acts in the same way  $M_i$  would react to  $\$$ . If  $M_i$  would accept the initial pair, it remembers this and clears its stack. It then checks to see if it has read the final  $L$  factors, and accepts if it has done so. If it has not, it reads through the next  $L'$  portion of  $w$  and  $w'$ . Upon encountering a change of generating set, it repeats its emulation of  $M_i$ . Clearly,  $M_0$  and  $M'_i$  both accept the pair  $(w, w')$  if and only if these two differ only in their last  $L$  factors, these last factors would have been accepted by  $M_i$  and the remaining  $L'$  factors are trivial. Hence,  $M_0, M'_1, \dots, M'_k$  perform as required. An analogous construction works for  $a' \in A'$ .  $\square$

**Theorem 5.2.** *Suppose  $K$  is a finitely generated abelian group and*

$$1 \rightarrow K \rightarrow G \xrightarrow{p} H \rightarrow 1$$

*is a central extension of an automatic group  $H$ . Then  $G$  is  $\mathcal{P}$ .*

**Proof.** Recall that we can identify  $G$  with the set  $K \times H$  endowed with the multiplication  $(k, h)(k', h') = (k + k' + \rho(h, h'), hh')$  where  $\rho$  is the 2-cocycle that determines the extension. We write  $K = \mathbb{Z}^n \times F$  where  $F$  is finite. We take  $A_K = \{x_1^{\pm 1}, \dots, x_n^{\pm 1}\} \cup A_F$  to be a generating set for  $K$ , where  $\{\bar{x}_i\}$  is a basis for  $\mathbb{Z}^n$  and  $A_F$  has the same number of elements as  $F$ . We assume  $A'_H$  is a generating set for  $H$  and take  $A_H$  to be a set of lifts of these generators chosen by the inclusion  $H \mapsto \{0\} \times H$ . Since  $H$  is assumed to be automatic, there is a regular language  $L'_H \subset A'^*_H$  which bijects to  $H$  and has the “fellow traveler property” (see below). We let  $L_H$  be the corresponding sublanguage of  $A^*_H$ . We take  $L_K = \{x_1^{m_1} \dots x_n^{m_n} f \mid m_i \in \mathbb{Z}, f \in A_f\}$ . We claim  $L = L_H L_K \subset (A_H \cup A_K)^*$  is a  $\mathcal{P}$  structure for  $G$ .

Clearly, the natural map  $L \rightarrow G$  is a bijection and  $L$  is regular. It is easy to see that for each  $a \in A_K$   $\{(w, w') \in L \times L \mid \overline{w'} = \overline{wa}\}$  is the intersection of  $L \times L$  with the language of a synchronous two-tape finite state automaton, and thus  $\text{AT}\mathcal{P}$ . We now show that for each  $a \in A_H$ ,  $\{(w, w') \in L \times L \mid \overline{w'} = \overline{wa}\}$  is  $\text{AT}\mathcal{P}$ . So suppose we have  $w = ux_1^{m_1} \dots x_n^{m_n} f$ ,  $w' = u'x_1^{m'_1} \dots x_n^{m'_n} f'$ , and  $a \in A_H$ . We will use  $p_i$  and  $p_f$  to represent the maps  $K \rightarrow \langle \overline{x_i} \rangle$  and  $K \rightarrow F$ . We will have

$$\begin{aligned} \overline{w'} &= \overline{wa} && \Leftrightarrow \\ \overline{u'x_1^{m'_1} \dots x_n^{m'_n} f'} &= \overline{ux_1^{m_1} \dots x_n^{m_n} fa} && \Leftrightarrow \\ \overline{x_1^{m'_1} \dots x_n^{m'_n} f'} &= \overline{uau'^{-1}x_1^{m_1} \dots x_n^{m_n} f}. \end{aligned}$$

These will happen if and only if:

- (1)  $p(\overline{w'}) = p(\overline{wa})$ ,
- (2) for each  $i$ ,  $m'_i = m_i + p_i(\overline{uau'^{-1}})$ , and
- (3)  $f' = f + p_f(\overline{uau'^{-1}})$ .

The first condition is easily checked by the comparator two-tape finite state automaton for  $H$ . We assume the first condition is satisfied. To check the second condition, for each  $i$  we build a two-tape pushdown automaton  $M_i$  which reads  $(u, u')$  and leaves  $p_i(\overline{uau'^{-1}})$  on its stack.

Recall that if condition 1 is satisfied,  $L'_H$  has the fellow traveler property, i.e., there is a constant  $C$  so that for each  $j$ , there is  $t_j \in H$  with  $\ell(t_j) \leq C$ , so that  $p(\overline{u'(j)}) = p(\overline{u(j)})t_j$ . (Here  $u(j)$  and  $u'(j)$  denote the initial segments of length  $j$  of  $u$  and  $u'$ .)

We start with the stack empty and  $t_0 = 1$ . We assume inductively that  $M_i$  has read the first  $j$  letters of  $u$  and  $u'$ , that it has the value  $p_i(p(\overline{u(j)})t_j p(\overline{u'(j)})^{-1})$  on the stack and knows the value of  $t_j$ . (Since  $\ell(t_j) \leq C$ , this latter requires only a bounded amount of memory and can be carried in  $M_i$ 's internal memory.)  $M_i$  reads  $a_{j+1}$  and  $a'_{j+1}$  from  $u$  and  $u'$  respectively. If one of  $u$  or  $u'$  has been exhausted, but the other has not, then one of these letters can be taken to be the empty word. Then  $t_{j+1} = p(\overline{a_{j+1}})^{-1}t_j p(\overline{a'_{j+1}})$ . Since there are only finitely many values in this formula, this computation can be done in  $M_i$ 's internal memory. Now observe that

$$\begin{aligned} p_i(p(\overline{u(j+1)})t_{j+1} p(\overline{u'(j+1)})^{-1}) &= p_i(p(\overline{u(j)})t_j p(\overline{u'(j)})^{-1}) \\ &\quad + p_i(p(t_{j+1}, p(\overline{a'_{j+1}})^{-1})t_j^{-1} p(\overline{a_{j+1}})). \end{aligned}$$

The first term on the right is the contents of the stack before  $M_i$  reads  $a_{j+1}$  and  $a'_{j+1}$  and the second term on the right is determined by the finitely many possible values of  $a_{j+1}$ ,  $a'_{j+1}$ ,  $t_j$  and  $t_{j+1}$ . Thus, in order to compute  $p_i(p(\overline{u(j+1)})t_{j+1} p(\overline{u'(j+1)})^{-1})$ ,  $M_i$  need only add  $p_i(p(t_{j+1}, p(\overline{a'_{j+1}})^{-1})t_j^{-1} p(\overline{a_{j+1}}))$ . This can be done since only finitely many such values occur. If condition 1 is satisfied, then when  $M_i$  is finished reading  $u$  and  $u'$ , it's final  $t$ -value will be  $p(a)$  and the stack will contain  $p_i(\overline{uau'^{-1}})$  as required.

It is now easy to check condition 2 using  $\{M_i\}$ . For each  $i$  we build a machine  $M'_i$  which first uses  $M_i$  to read  $u$  and  $u'$ .  $M'_i$  reads  $w$  and  $w'$  until it gets to the  $x_i^{\pm 1}$  portion. It then pops the contents of the stack, canceling it letter by letter against the  $x_i^{\pm 1}$  letters of  $w$  or  $w'$  as appropriate. After it has emptied the stack, it reads the  $x_i^{\pm 1}$  letters of  $w$  and  $w'$  one at a time and accepts if and only if these agree in sign and it exhausts these simultaneously.

Condition 3 can be checked in the same way, except that here we do not require a stack to keep track of  $p_f$  since only finitely many values occur.  $\square$

**Corollary 5.3.** *Every finitely generated nilpotent group of nilpotency class 2 is  $\mathcal{P}$ .*

Notice that Theorem contrasts with the fact that the class of (asynchronously) automatic groups contains no nilpotent groups of class greater than 1 (see [6]).

We now show how to construct some non-finitely presented  $\mathcal{P}$  groups.

**Theorem 5.4.** *The class of  $\mathcal{P}$  groups is closed under wreath product.*

**Proof.** Let  $G$  and  $H$  be  $\mathcal{P}$  groups. The wreath product  $G \wr H$  can be identified with the semi-direct product  $G^{[H]} \rtimes H$ . Let  $L_H \subset A_H^*$  and  $L_G \subset A_G^*$  be languages of  $\mathcal{P}$  structures for these groups. Fixing an ordering on the finite set  $A_H$  induces a “short lex” ordering  $\prec$  on  $A_H^*$ , i.e.,  $u \prec v$  if  $u$  is shorter than  $v$  or if  $u$  and  $v$  have the same length, and  $u$  is lexically prior to  $v$ .

To keep track of  $H$ -conjugates of elements of  $G$ , we introduce formal inverses of the elements of  $L_H$ . That is, we introduce a disjoint copy  $A_H^{-1}$  of formal inverses of the elements of  $A_H$ , and the formal inverse map  $A_H^* \rightarrow (A_H^{-1})^* : u = a_1 \dots a_n \mapsto u^{-1} = a_n^{-1} \dots a_1^{-1}$ . We use the bold face exponent  $^{-1}$  to distinguish the process of taking formal inverses in  $A_H^{-1}$  from the process of taking inverses in  $A_H$ . We will also need a second copy of  $A_H$  which we will denote by  $A_H^\circ$ . The isomorphism between  $A_H$  and  $A_H^\circ$  induces an isomorphism of  $A_H^*$  and  $(A_H^\circ)^*$  denoted by  $u \mapsto u^\circ$ .

We now take

$$L = \{w_1 \dots w_k u_0^\circ \mid \begin{aligned} w_i &= u_i v_i u_i^{-1} \text{ for } i = 1, \dots, k, \\ u_i &\in L_H \text{ for } i = 0, \dots, k, \\ v_i &\in L_G \text{ for } i = 1, \dots, k, \\ u_i &\prec u_{i+1} \text{ for } i = 1, \dots, k - 1 \}. \end{aligned}$$

Because the natural maps from  $L_H$  and  $L_G$  to  $H$  and  $G$  are bijections, it is easy to see that the natural map from  $L$  to  $G \wr H$  is also a bijection. Notice that since the  $\{u_i\}$  are in strictly increasing  $\prec$  order,  $\bar{u}_i \neq 1$  for  $i = 2, \dots, k$ .

We now check that  $L$  is a  $\mathcal{P}$  language. We use one machine to check that each  $u_i$  and  $u_i^{-1}$  are formal inverses of each other. This is done by successively pushing each  $L_H$  subword onto the stack and popping it off letter by letter and checking it against the next  $L_H^{-1}$  subword encountered. This machine accepts only if each of these comparisons is successful (except of course for the  $u_0$  subword).

A second machine checks that for  $i = 1, \dots, k - 1$ , we have  $u_i \prec u_{i+1}$ . This machine acts by pushing each  $u_i$  subword encountered onto the stack. If the first machine accepts the word, then each  $u_i^{-1}$  subword is the formal inverse of the previous  $u_i$  subword. Since pushing this onto the stack reverses order, one may now pop the contents of the stack, comparing them letter by letter with the next  $u_i^{-1}$  subword to determine if they are in  $\prec$  order. This machine accepts if each of these comparisons is successful.

Finally, a collection of  $L_H$  machines successively check each  $L_H$  (and  $L_H^\circ$ ) subword and accepts only if each of these leads to an acceptance, while a collection of  $L_G$  machines check each  $L_G$  subword. All of these machines accept if and only if  $w \in L$ .

We now wish to check that for each  $a$ ,  $\{(w, w') \in L \times L \mid \overline{w'} = \overline{w}a\}$  is  $AT\mathcal{P}$ . If  $\overline{a}$  is in  $\overline{A_H^{\pm 1}}$ , this is easy to do. We simply check that  $w$  and  $w'$  are identical up to the final  $u_i$  subword, and then use the multiplication machines for  $L_H$  to compare  $u_o^\circ$  with  $u_o'^\circ$ . (When multiplying by  $a^{-1} \in A_H^{-1}$ , notice that  $\overline{u'} = \overline{u}a^{-1} \Leftrightarrow \overline{u} = \overline{u'}a$ .) So we suppose that  $\overline{a} \in \overline{A_G}$  and  $w = w_1 \dots w_k u_o^\circ \in L$ , with  $w_i = u_i v_i u_i^{-1}$ , and with the  $u_i$  in increasing order. We can assume that  $\overline{a} \neq 1$ . Now if  $\overline{w'} = \overline{w}a$ , there are three possibilities for  $w$ :

(1) there is no  $i$ ,  $1 \leq i \leq k$  so that  $u_i = u_0$ ; in this case,  $w' = w_1 \dots w_i u_0 v_a u_0^{-1} w_{i+1} \dots w_k u_o^\circ$ , where  $v_a$  is the  $L_G$  word for  $\overline{a}$  and  $i$  is the largest value such that  $u_i \prec u_0$ ;

(2) there is an  $i$ ,  $1 \leq i \leq k$  so that  $u_0 = u_i$  and  $\overline{v_i} = (\overline{a})^{-1}$ ; in this case,  $w' = w_1 \dots w_{i-1} w_{i+1} \dots w_k u_o^\circ$ ;

(3) there is an  $i$ ,  $1 \leq i \leq k$  so that  $u_0 = u_i$  and  $\overline{v_i} \neq (\overline{a})^{-1}$ . In this case  $w' = w_1 \dots w_{i-1} u_i v'_i u_i^{-1} w_{i+1} \dots w_k u_o^\circ$ , where  $\overline{v'_i} = \overline{v_i} \overline{a}$ .

In particular,

(★)  $w'$  differs from  $w$  by either differing at  $v_i$  with  $u_i = u_0$  or by insertion of  $u_0 v_a u_0^{-1}$  or by deletion of  $u_0 v_{a^{-1}} u_0^{-1}$  ( $\overline{v_a} = \overline{a}$ ,  $\overline{v_{a^{-1}}} = (\overline{a})^{-1}$ ).

We first build a machine  $M_0$  that determines if  $w$  and  $w'$  differ in exactly one of these ways. This machine starts reading each of  $w$  and  $w'$ . The disjoint sets of generators indicate whether a letter being read lies in a  $u_i$ , a  $v_i$ , a  $u_i^{-1}$  or  $u_o^\circ$  subword. The machine  $M_0$  reads each of  $w$  and  $w'$  one letter at a time until it discovers a discrepancy. If that first discrepancy is in the last  $u_i$  subword, it enters a fail state. If the first discrepancy is in a  $v_i$  subword, it finishes reading that subword from each side and accepts if the remainder of the two words is identical. If  $M_0$  discovers the first discrepancy in a  $u_i$  subword (other than the last one), say,  $u_i \neq u'_i$ , it continues to read the  $u_i$  subwords and determines which word is  $\prec$  earlier. If  $u_i \prec u'_i$ , then  $M_0$  checks if  $v_i$  was the word in  $L_G$  which represents  $(\overline{a})^{-1}$ . If not, it rejects the pair. This information is preserved in state of the machine while the subword  $v_i$  was read. If  $\overline{v_i} = (\overline{a})^{-1}$ , it then continues reading, pushing  $u_i^{-1}$  onto the stack. The subword  $u_{i+1}$  is then read, and after this,  $M_0$  continues reading each subword, checking for equality. When it arrives at the final  $u_i$  subword, it pops the stack checking to see if  $u_i = u_0$  and accepts if both these things happen. If, on the other hand  $u'_i \prec u_i$ ,  $M_0$  checks that  $v'_i$  is the word in  $L_G$  representing the element  $\overline{a}$ . If so, it pushes  $u_i^{-1}$  onto the stack and proceeds as before to check that  $u'_i = u_0$  and that the remainder of the words are identical. Finally, if  $M_0$  does not

encounter a discrepancy, it rejects the pair. Thus if  $M_0$  is given a pair  $(w, w') \in L \times L$ , it determines whether or not the pair satisfies  $(\star)$ , above.

It is now easy to check if  $\overline{w'} = \overline{w}a$ . For each machine  $M_j$  used to check a multiplication in  $L_G$ , we build a machine  $M'_j$ .  $M'_j$  checks each  $u_i$  against the corresponding  $u'_i$  looking for inequality. It checks each  $v_i$  against the corresponding  $v'_i$  in the manner of  $M_j$ . It continues both of these tasks until it succeeds at one of them. Notice that if the pair is in  $L \times L$  and is accepted by  $M_0$ ,  $M'_j$  can only succeed at one of these tasks, and for only one value of  $i$ . Thus all of these machines together determine  $\{(w, w') \in L \times L \mid \overline{w'} = \overline{w}a\}$  as required.  $\square$

In view of the fact that there are wreath products of  $\mathcal{P}$  groups which are not finitely presentable (for instance, if  $C$  denotes the infinite cyclic group,  $C \wr C$  is not finitely presented), we have:

**Corollary 5.5.** *There are  $\mathcal{P}$  groups which are not finitely presented.*

We now turn to the consideration of semi-direct products of abelian groups by  $\mathcal{P}$  groups.

**Theorem 5.6.** *Suppose that  $H$  is a  $\mathcal{P}$  group, and  $\varphi : H \rightarrow \text{Aut } \mathbb{Z}^n$ . Then  $\mathbb{Z}^n \rtimes_{\varphi} H$  is  $\mathcal{P}$ .*

To obtain this result, we use the following lemma:

**Lemma 5.7.** *Suppose  $A \in \text{Aut } \mathbb{Z}^n$ . Then*

$$L_A = \{(x_1^{m_1} \dots x_n^{m_n}, x_1^{m'_1} \dots x_n^{m'_n}) \mid (m'_1, \dots, m'_n) = A(m_1, \dots, m_n)\}$$

is  $\text{AT}\mathcal{P}$ .

**Proof.** We start by observing that for each  $i$ ,  $1 \leq i \leq n$ , we can build a pushdown automaton which reads  $x_1^{m_1} \dots x_n^{m_n}$  and ends with the  $i$ th coordinate of  $A(m_1, \dots, m_n)$  on its stack. It does this by adding an  $(A)_{ij}$  to the stack for each  $x_j$  that it encounters. Now, to recognize the  $i$ th coordinate of the language  $L_A$ , we continue by reading  $(x^{m'_1}, \dots, x^{m'_n})$  and subtracting 1 for each  $x_i$  or adding 1 for each  $x_i^{-1}$  we encounter. The automaton then accepts by empty stack. The collection of automata, one for each  $i$ , then determines the language  $L_A$ .  $\square$

**Proof of Theorem 5.6.** Let  $L_H \subset A_H^*$  be the language of a  $\mathcal{P}$  structure for  $H$ , and take the language

$$L = L_H \{x_1^{m_1} \dots x_n^{m_n}\}.$$

Right multiplication by a generator  $x_i^{\pm 1}$  is easily realized, and in view of the lemma, it is also easy to check right multiplication by a generator  $h \in A_H$  using  $A = A_{\varphi(h)}$ .  $\square$

**Corollary 5.8.** *The class of  $\mathcal{P}$  groups contains groups of isoperimetric inequality of every polynomial degree.*

**Proof.** Bridson and Gersten [4] have characterized the isoperimetric inequalities of groups  $\mathbb{Z}^n \rtimes_A \mathbb{Z}$ , where  $A$  is a nilpotent matrix. Such a group has isoperimetric inequality  $n^{d+1}$  where  $d$  is the size of the longest block of the Jordan canonical form of  $A$ .  $\square$

The fundamental groups of Nil and Solvgeometry manifolds are almost of the above form, that is, each contains a finite index subgroup of the form  $\mathbb{Z}^2 \rtimes_A \mathbb{Z}$ . Unfortunately, we have no general result allowing us to pass to finite index supergroups, so we prove the following.

**Lemma 5.9.** *If  $G$  contains a finite index subgroup which is a semi-direct product of the form  $\mathbb{Z}^n \rtimes_A \mathbb{Z}$ , then  $G$  is  $\mathcal{P}$ .*

**Proof.** We take generators  $x_1^{\pm 1}, \dots, x_n^{\pm 1}$  for  $\mathbb{Z}^n$ ,  $z^{\pm 1}$  for  $\mathbb{Z}$  and a finite set  $T$  giving a transversal for  $\mathbb{Z}^n \rtimes_A \mathbb{Z}$  in  $G$ . We take as our language  $L = \{z^p x_1^{m_1} \dots x_n^{m_n}\}T$ . We must check that we can detect right multiplication by a generator. So suppose we have the pair  $w = z^r x_1^{m_1} \dots x_n^{m_n} t$  and  $w' = z^{r'} x_1^{m'_1} \dots x_n^{m'_n} t'$ , and we wish to check if  $\overline{w} = \overline{w'g}$ . To do this we will use  $n$  automata, one for each of the letters  $x_i$ . Let  $g$  be a generator. There are only finitely many possibilities for  $tg$ , and each of these can be written in the form  $us$  with  $u = z^a x_1^{b_1} \dots x_n^{b_n} \in \mathbb{Z}^n \rtimes_A \mathbb{Z}$  and  $s \in T$ . We start by reading the  $z$  portions of  $w$  and  $w'$ . If  $|r - r'|$  exceeds the largest  $z$  exponent in any  $u$ , then  $\overline{w} \neq \overline{w'g}$  and we reject the pair. If  $|r' - r|$  does not exceed this value, we remember  $r' - r$ . This requires a bounded amount of memory (i.e., this information is stored on one of a finite number of states). We now proceed to read the remainder of  $w$ , pushing  $x_i^{m_i}$  onto the stack in the  $i$ th automaton. When we encounter  $t$ , we then compute  $u$  and  $s$ . There are only finitely many possibilities for these. We can now check if  $r' - r = a$ . If not, we reject the pair. If  $r' - r = a$ , we continue. We read the  $\mathbb{Z}^n$  portion of  $w'$ , applying the transformation  $A^a$  as in Lemma 5.6 popping and pushing the contents of each stack accordingly. When this is done, we accept if and only if for each  $i$ , the stack of the  $i$ th machine contains  $b_i$  and  $t' = s$ .  $\square$

**Corollary 5.10.** *Suppose  $M$  is a 3-manifold which obeys the Thurston geometrization conjecture. Then  $\pi_1(M)$  is  $\mathcal{P}$ .*

**Proof.** If  $M$  is such a manifold,  $\pi_1(M)$  is the free product of an automatic group with finitely many fundamental groups of closed Sol or Nil geometry manifolds. Each of these Sol or Nil geometry groups contains a finite index subgroup of the form  $\mathbb{Z}^2 \rtimes_A \mathbb{Z}$ , where  $A$  is either nilpotent or Anosov. Since automatic groups are  $\mathcal{P}$  and  $\mathcal{P}$  groups are closed under free product, the result now follows.  $\square$

### 6. Nilpotent groups

We here study the group  $U(n)$  whose elements are the  $n \times n$  upper triangular integral matrices with 1's on the diagonal. Our interest in this group comes from the fact that if  $G$  is a finitely generated torsion free nilpotent group, then for sufficiently large  $n$ ,  $G$  embeds in  $U(n)$  [1].

**Theorem 6.1.** *For each  $n$ ,  $U(n)$  is  $\mathcal{P}$ . In particular  $\mathcal{P}$  contains nilpotent groups of every class and every finitely generated torsion free nilpotent group embeds in a  $\mathcal{P}$  group.*

Before proving the theorem, we recall some basic facts about  $U(n)$ . Each element of  $U(n)$  has the shape

$$\begin{pmatrix} 1 & * & * & \dots & * & * \\ 0 & 1 & * & \dots & * & * \\ \vdots & & \ddots & & & \vdots \\ 0 & 0 & 0 & \dots & 1 & * \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

We will take  $H(n)$  to be those matrices with the shape

$$\begin{pmatrix} 1 & * & * & \dots & * & * & * \\ 0 & 1 & 0 & \dots & 0 & 0 & * \\ \vdots & & \ddots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & * \\ 0 & 0 & 0 & \dots & 0 & 1 & * \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

That is,  $H(n)$  consists of those matrices of  $U(n)$  for which all nonzero entries are either on the diagonal, the first row, or the last column. Let  $T(n) \subset H(n)$  be those matrices in which all nonzero entries are restricted to the diagonal and the top row. Similarly, let  $R(n) \subset H(n)$  be those matrices in which all nonzero entries are restricted to the diagonal and the extreme right column. Let  $e_{ij}$  be the matrix which has as single nonzero entry, a 1 in the  $ij$  position.

**Lemma 6.2.** (1)  $T(n)$  is a free abelian group of rank  $n - 1$  generated by  $x_j = 1 + e_{1j}$ ,  $2 \leq j \leq n$ .  
 (2)  $R(n)$  is a free abelian group of rank  $n - 1$  generated by  $y_i = 1 + e_{in}$ ,  $1 \leq i \leq n - 1$ .

(3)  $H(n) = T(n)R(n)$ . Further  $H(n)$  has the presentation

$$\langle x_2, \dots, x_{n-1}, y_2, \dots, y_{n-1}, z \mid [x_i, x_j] = 1, [y_i, y_j] = 1, [x_i, y_j] = 1, i \neq j \\ [x_i, y_i] = z, z \text{ is central} \rangle.$$

In particular,  $L_{H(n)} = \{x_2^{p_2} \dots x_{n-1}^{p_{n-1}} y_2^{q_2} \dots y_{n-1}^{q_{n-2}} z^r\}$  is the language of a  $\mathcal{P}$  structure for  $H(n)$ .

(4)  $U(n)$  is generated by  $\{1 + e_{ij} \mid j > i\}$ . There is a split short exact sequence

$$1 \rightarrow H(n) \rightarrow U(n) \rightarrow U(n - 2) \rightarrow 1.$$

The splitting is given by the inclusion of  $U(n - 2)$  into  $U(n)$  as  $1 \oplus U(n - 2) \oplus 1$ , the subset of  $U(n)$  for which the off-diagonal entries in the first row and last column are all zero. The action of the generator  $1 + e_{ij}$ ,  $1 < i < j < n$  on  $H(n)$  carries  $x_2^{p_2} \dots x_{n-1}^{p_{n-1}} y_2^{q_2} \dots y_{n-1}^{q_{n-1}} z^r$  to  $x_2^{p'_2} \dots x_{n-1}^{p'_{n-1}} y_2^{q'_2} \dots y_{n-1}^{q'_{n-1}} z^r$  where for  $m \neq j$ ,  $p'_m = p_m$ ,  $p'_j = p_i + p_j$ , for  $m \neq i$ ,  $q'_m = q'_m$ ,  $q'_i = q_i - q_j$ .

**Proof.** First observe that

$$e_{ij}e_{kl} = \begin{cases} 0 & \text{if } j \neq k, \\ e_{il} & \text{if } j = k, \end{cases}$$

so that for  $i < j$ ,  $(1 + e_{ij})^{-1} = 1 - e_{ij}$ . Statements (1) and (2) now follow easily. In this way it is also easy to see that  $x_2, \dots, x_{n-1}, x_n = z = y_1, y_2, \dots, y_{n-1}$  fulfill the relations of the presentation of (3). One checks that  $H(n)$  is in fact a subgroup of  $U(n)$ . From the relations of the presentation it is easy to see that each product of the elements  $x_2, \dots, x_{n-1}, x_n = z = y_1, y_2, \dots, y_{n-1}$  can be put into the form

$$x_2^{p_2} \dots x_{n-1}^{p_{n-1}} y_2^{q_2} \dots y_{n-1}^{q_{n-2}} z^r.$$

A computation shows that this element corresponds to the matrix

$$\begin{pmatrix} 1 & p_2 & p_3 & \dots & p_{n-1} & r + \sum_{i=2}^{n-1} p_i q_i \\ 0 & 1 & 0 & \dots & 0 & q_2 \\ \vdots & & \ddots & & & \vdots \\ 0 & 0 & 0 & \dots & 1 & q_{n-1} \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

Thus  $L_{H(n)}$  bijects to  $H(n)$ . In particular, it follows that the elements  $x_2, \dots, x_{n-1}, x_n = z = y_1, y_2, \dots, y_{n-1}$  generate  $H(n)$ . Notice that the words of  $L_{H(n)}$  are already in the form  $T(n)R(n)$ , so  $H(n) = T(n)R(n)$  as claimed.

Let  $\mathcal{H}(n)$  be the group determined by the presentation. Since the generators of  $H(n)$  obey the relations of the presentation, the obvious map from  $\mathcal{H}(n)$  to  $H(n)$  is a surjective homomorphism. To see that this is an isomorphism, let  $g$  be an element of

the kernel. Using the relations of the presentation, we can write  $g$  in the normal form we have used in  $H(n)$ . The fact that  $g$  maps to the identity matrix in  $H(n)$  forces  $p_2 = \dots = p_{n-1} = q_2 = \dots = q_{n-1} = 0$  and  $r = 0$ , so in fact  $g = 1$  in  $\mathcal{H}(n)$  as required.

We must check that  $L_{H(n)}$  gives a  $\mathcal{P}$  structure. It is a regular language such that the natural map to  $H(n)$  is bijective. We now show how to check right multiplication in the appropriate way. Right multiplying by  $z$  or  $y_i$  only increases  $r$  or  $q_i$  by 1, and this can be checked by a finite state automaton. It remains to check right multiplication by  $x_i$ . This increases  $p_i$  by one and changes  $r$  by  $q_i$ . This can be checked using a single stack. We leave the details to the reader.

To prove (4), first notice that

$$\begin{pmatrix} 1 & a_{12} & a_{13} & \dots & a_{1n-2} & a_{1n-1} & a_{1n} \\ 0 & 1 & a_{23} & \dots & a_{2n-2} & a_{2n-1} & a_{2n} \\ \vdots & & \ddots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1 & a_{n-2n-1} & a_{n-2n} \\ 0 & 0 & 0 & \dots & 0 & 1 & a_{n-1n} \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix} \\ = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & a_{23} & \dots & a_{2n-2} & a_{2n-1} & 0 \\ \vdots & & \ddots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1 & a_{n-2n-1} & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix} \\ \times \begin{pmatrix} 1 & a_{12} & a_{13} & \dots & a_{1n-2} & a_{1n-1} & a_{1n} \\ 0 & 1 & 0 & \dots & 0 & 0 & a_{2n} \\ \vdots & & \ddots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & a_{n-2n} \\ 0 & 0 & 0 & \dots & 0 & 1 & a_{n-1n} \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

We now include  $U(n - 2)$  into  $U(n)$  as  $1 \oplus U(n - 2) \oplus 1$ . Then the above equation can be read as saying that  $U(n) = U(n - 2)H(n)$ . Now  $U(1), U(2)$ , and  $H(n)$  (for all  $n$ ) are generated by elements of the form  $1 + e_{ij}, i < j$ . Inductively, we conclude that  $U(n)$  is generated by  $\{1 + e_{ij} \mid i < j\}$ . It is now easy to check that  $H(n)$  is a normal subgroup, since we need only check conjugation by generators,  $1 + e_{ij}, i < j$ . This will

verify that the action on  $H(n)$  is as claimed. It remains only to check that the quotient of  $U(n)$  by  $H(n)$  is as claimed. We take the map of  $U(n)$  to  $U(n-2)$  to be the “forgetful” map which strips each matrix of its first and last rows and columns. A computation shows that this is a homomorphism and the kernel is clearly  $H(n)$ .  $\square$

**Proof of Theorem 6.1.** We use the short exact sequence to perform an induction argument. Since this induction takes us from  $U(n-2)$  to  $U(n)$ , it requires two basis steps,  $U(1)$  and  $U(2)$ . These groups are respectively the trivial group and  $\mathbb{Z}$ , both of which are  $\mathcal{P}$ , so the basis step is complete.

We now assume that  $L(n-2)$  is a regular language over an alphabet consisting of letters for  $\{1 \pm e_{ij} \mid 1 < i < j < n-2\}$  which is  $\mathcal{P}$  structure for  $U(n-2)$ . We take  $L(n) = L(n-2)L_{H(n)}$ . By the short exact sequence and our choice of generators for  $H(n)$ , this is a language over the desired generating set and is in one-to-one correspondence with  $U(n)$ . We must check that this is a  $\mathcal{P}$  structure. Right multiplication by an element of  $H(n)$  is easily checked, as we have seen in (3) of the Lemma 6.2. On the other hand, we can check right multiplication by an element of  $U(n-2)$  since this requires that we check right multiplication in  $U(n-2)$  (which we can do by induction) and check the action of a generator on  $L_{H(n)}$ , which we can do by Lemma 5.7.  $\square$

## References

- [1] G. Baumslag, Lecture Notes on Nilpotent Groups, Regional Conf. Series of A.M.S., vol. 2, 1971.
- [2] G. Baumslag, S.M. Gersten, M. Shapiro, H. Short, Automatic groups and amalgams, J. Pure Appl. Algebra 76 (1991) 229–316.
- [3] N. Brady, The geometry of asynchronous structures on groups, Ph.D. Thesis, University of California at Berkeley, 1992.
- [4] M.R. Bridson, S.M. Gersten, The optimal isoperimetric inequality for torus bundles over the circle, Quart. J. Math. Oxford Ser. (2) 47 (185) (1996) 1–23.
- [5] M.R. Bridson, R. Gilman, Formal language theory and the geometry of 3-manifolds, Comm. Math. Helv. 71 (4) (1996) 525–555.
- [6] D.B.A. Epstein, with J.W. Cannon, D.F. Holt, S.V.F. Levy, M.S. Paterson, W.P. Thurston, Word Processing in Groups, Jones and Barlett, Boston, 1992.
- [7] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, Reading, MA, 1979.
- [8] M. Shapiro, unpublished.