Ackermannian Integer Compression and the Word Problem for Hydra Groups*

Will Dison[1], Eduard Einstein[2], and Timothy R. Riley[3]

1 Bank of England
Threadneedle Street, London, EC2R 8AH, United Kingdom
william.dison@gmail.com

2 Department of Mathematics, Cornell University
310 Malott Hall, Ithaca, NY 14853, USA
ee256@cornell.edu

3 Department of Mathematics, Cornell University
310 Malott Hall, Ithaca, NY 14853, USA
tim.riley@math.cornell.edu

Abstract

For a finitely presented group, the word problem asks for an algorithm which declares whether or not words on the generators represent the identity. The Dehn function is a complexity measure of a direct attack on the word problem by applying the defining relations. Dison and Riley showed that a "hydra phenomenon" gives rise to novel groups with extremely fast growing (Ackermannian) Dehn functions. Here we show that nevertheless, there are efficient (polynomial time) solutions to the word problems of these groups. Our main innovation is a means of computing efficiently with enormous integers which are represented in compressed forms by strings of Ackermann functions.

1998 ACM Subject Classification F.2.2 Geometrical problems and computations, E.4 Data compaction and compression

Keywords and phrases Ackermann functions, hydra, word problem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.30

1 Ackermann functions, compressed integers, and our first theorem

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. For $i \in \mathbb{N}$, the Ackermann functions $A_i : \mathbb{N} \to \mathbb{N}$ are a family of recursively defined increasingly fast-growing functions:

(i) $A_0(n) = n + 1$ for all $n \in \mathbb{Z}$,
(ii) $A_1(n) = 2n$ for all $n \in \mathbb{Z}$,
(iii) $A_i(0) = 1$ for all $i \geq 2$, and
(iv) $A_{i+1}(n + 1) = A_i A_{i+1}(n)$ for all $n \geq 0$ and all $i \geq 1$.

The following table, showing some values of $A_i(n)$, can be constructed by first inserting the $i = 0, 1$ rows and then $n = 0$ column, and then filling in the subsequent rows left-to-right according to the recurrence relation.

* We gratefully acknowledge partial support from NSF grant DMS-1101651 (TR) and Simons Collaboration Grant 318301 (TR), and the hospitality of the Mathematical Institute, Oxford (EE & TR), and the Institute for Advanced Study, Princeton (TR).

41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016).
Editors: Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier; Article No. 30; pp. 30:1–30:14
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

	0	1	2	3	4	$\cdots$	n	$\cdots$
A_0	1	2	3	4	5	$\cdots$	$n+1$	$\cdots$
A_1	0	2	4	6	8	$\cdots$	$2n$	$\cdots$
A_2	1	2	4	8	16	$\cdots$	2^n	$\cdots$
A_3	1	2	4	16	65536	$\cdots$	$\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}n$	$\cdots$
A_4	1	2	4	65536	$\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}65536$	$\cdots$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$			

Due to the increasing nesting of the recursion, the A_i represent the successive graduations in a hierarchy of all primitive recursive functions due to Grzegorczyk (see e.g. [30]).

The functions A_i are all strictly increasing and hence injective, so have partial inverses:

(I) $A_0^{-1} : \mathbb{Z} \to \mathbb{Z}$ mapping $n \mapsto n-1$,
(II) $A_1^{-1} : 2\mathbb{Z} \to \mathbb{Z}$ mapping $n \mapsto n/2$, and
(III) $A_i^{-1} : \text{Img } A_i \to \mathbb{N}$ for all $i > 1$.

Starting with zero and successively applying a few Ackermann functions and their inverses can produce an enormous integer. For example,

$$A_3A_0A_1^2A_0(0) \;=\; A_3A_0A_1^2(1) \;=\; A_3A_0A_1(2) \;=\; A_3A_0(4) \;=\; A_3(5) \;=\; 2^{65536}$$

because

$$A_3(5) \;=\; A_2^5A_3(0) \;=\; A_2^5(1) \;=\; 2^{2^{2^{2^{2}}}} \;=\; 2^{65536}.$$

Thus Ackermann functions give highly compact representations for some very large numbers.

More precisely, here is how a string w of Ackermann functions may represent an integer $w(0)$. For $x_1, \ldots, x_n \in \{A_0^{\pm 1}, \ldots, A_k^{\pm 1}\}$, we say the word $w = x_n x_{n-1} \cdots x_1$ is *valid* if $x_m x_{m-1} \cdots x_1(0)$ is defined for all $0 \leq m \leq n$. That is, if we evaluate $w(0)$ by starting with 0 and proceeding through w from right to left applying successive x_i, we never encounter the problem that we are trying to apply x_i to an integer outside its domain.

For example, $w := A_2^{-1}A_1A_1A_0$ is valid, and $w(0) = \log_2(2 \cdot 2 \cdot (0+1)) = 2$. But $A_2A_0^{-1}$ and $A_1A_1^{-1}A_0$ are not valid because $A_0^{-1}(0) = -1$ is not in $\mathbb{N}$ (the domain of A_2) and because $A_0(0) = 1$ is not in $2\mathbb{Z}$ (the domain of A_1^{-1}).

Motivated by applications in group theory that we will describe in the next section, we wish to compute with these representations in an efficient manner. (Our choices of $\mathbb{Z}$ as the domains for A_0 and A_1 and our definition of A_0 represent small variations on the standard definitions of Ackermann functions, which are convenient for our applications.) One could just evaluate $w(0)$ using standard integer arithmetic, but this can be monumentally inefficient because of the sizes of the integers involved. Our first theorem is that is it possible to calculate efficiently in a rudimentary way with these representations of integers:

▶ **Theorem 1.** *Fix an integer $k \geq 0$. There is a polynomial-time algorithm, which on input a word w on $A_0^{\pm 1}, \ldots, A_k^{\pm 1}$, declares whether or not $w(0)$ represents an integer, and if so whether $w(0) < 0$, $w(0) = 0$ or $w(0) > 0$.*

(In fact our algorithm halts in time bounded above by a polynomial of degree $4 + k$. We have not attempted to optimize the degrees of the polynomial bounds on time complexity here or elsewhere in this work.)

2 The word problem, Dehn functions, and our second theorem

Elements of a group Γ with a generating set A can be represented by words—that is, products of elements of A and their inverses. To work with Γ, it is useful to have an algorithm which, on input a word, declares whether that word represents the identity element in Γ. After all, if we can recognize when a word represents the identity, then we can recognize when two words represent the same group element, and thereby begin to compute in Γ. The issue of whether there is such an algorithm is known as the *word problem* for (Γ, A) and was first posed by Dehn [9, 10] in 1912. (He did not precisely ask for an algorithm, of course, rather '*eine Methode angeben, um mit einer endlichen Anzahl von Schritten zu entscheiden...*'—that is, '*specify a method to decide in a finite number of steps....*')

Suppose a group Γ has a finite presentation $\langle\, a_1, \ldots, a_m \mid r_1, \ldots, r_n \,\rangle$. The *Dehn function* $\mathrm{Area} : \mathbb{N} \to \mathbb{N}$ quantifies the difficulty of a *direct attack* on the word problem: roughly speaking $\mathrm{Area}(n)$ is the minimal N such that if a word of length at most n represents the identity, then it does so 'as a consequence of' at most N defining relations.

Here is some notation that we will use to make this more precise. Associated to a set $\{a_1, a_2, \ldots\}$ (an *alphabet*) is the set of inverse letters $\{a_1^{-1}, a_2^{-1}, \ldots\}$. The inverse map is the involution defined on $\{a_1^{\pm 1}, a_2^{\pm 1}, \ldots\}$ that maps $a_i \mapsto a_i^{-1}$ and $a_i^{-1} \mapsto a_i$ for all i. The inverse map extends to words by sending $w = x_1 \cdots x_s \mapsto x_s^{-1} \cdots x_1^{-1} = w^{-1}$ when each $x_i \in \{a_1^{\pm 1}, a_2^{\pm 1}, \ldots\}$. Words u and v are *cyclic conjugates* when $u = \alpha\beta$ and $v = \beta\alpha$ for some subwords α and β. *Freely reducing* a word means removing all $a_j^{\pm 1} a_j^{\mp 1}$ subwords. For Γ presented as above, *applying a relation* to a word $w = w(a_1, \ldots, a_m)$ means replacing some subword τ with another subword σ such that some *cyclic conjugate* of $\tau\sigma^{-1}$ is one of $r_1^{\pm 1}, \ldots, r_n^{\pm 1}$.

For a word w representing the identity in Γ, $\mathrm{Area}(w)$ is the minimal $N \geq 0$ such that there is a sequence of *freely reduced* words $w_0, \ldots, w_N$ with w_0 the freely reduced form of w, and w_N is the empty word, such that for all i, w_{i+1} can be obtained from w_i by *applying a relation* and then *freely reducing*. The *Dehn function* $\mathrm{Area} : \mathbb{N} \to \mathbb{N}$ is defined by

$$\mathrm{Area}(n) \;:=\; \max \{\, \mathrm{Area}(w) \mid \text{words } w \text{ with } \ell(w) \leq n \text{ and } w = 1 \text{ in } \Gamma \,\}.$$

This is one of a number of equivalent definitions of the Dehn function. While a Dehn function is defined for a particular finite presentation for a group, its growth type—quadratic, polynomial, exponential etc.—does not depend on this choice. Dehn functions are important from a geometric point-of-view and have been studied extensively. There are many places to find background, for example [4, 5, 6, 10, 15, 16, 29, 31].

If $\mathrm{Area}(n)$ is bounded above by a recursive function $f(n)$, then it is possible to solve the word problem by an exhaustive search: to tell whether or not a given word w represents the identity, try all the possible ways of applying at most $f(n)$ defining relations and see whether one reduces w to the empty word. (There are finitely presented groups for which there is no algorithm to solve the word problem [3, 27].) Conversely, when a finitely presented group admits an algorithm to solve its word problem, $\mathrm{Area}(n)$ is bounded above by a recursive function (in fact $\mathrm{Area}(n)$ *is* a recursive function) [14].

There are finitely presented groups for which an extrinsic algorithm is far more efficient than this intrinsic brute-force approach. A simple example is $\mathbb{Z}^2 = \langle\, a, b \mid ab = ba \,\rangle$ (which has Dehn function $\mathrm{Area}(n) \simeq n^2$). Given a word on $a^{\pm 1}, b^{\pm 1}$, the extrinsic approach amounts to searching exhaustively through all the ways of shuffling letters $a^{\pm 1}$ past letters $b^{\pm 1}$ to see if there is one which brings each $a^{\pm 1}$ together with an $a^{\mp 1}$ to be cancelled, and likewise each $b^{\pm 1}$ together with a $b^{\mp 1}$. It is much more efficient to read through the word and check that the number of a is the same as the number of a^{-1}, and the number of b is the same as the number of b^{-1}.

There are more dramatic examples of groups where Area(n) is a fast growing recursive function (so the 'brute force' algorithm succeeds but is extremely inefficient), but there are efficient ways to solve the word problem. Cohen, Madlener & Otto built extraordinary examples in a series of papers [7, 8, 25] where Dehn functions were first introduced (under then name *derivational complexity*). They designed their groups in such a way that the 'intrinsic' method of solving the word problem involves running a very slow algorithm which has been suitably 'embedded' in the presentation. But running this algorithm to see whether it halts on a given input is pointless as it is constructed to halt (eventually) on all inputs and so presents no obstacle to the word representing the identity. Their examples all admit algorithms to solve the word problem in running times that are at most $n \mapsto \exp^{(\ell)}(n) := \underbrace{\exp(\exp(\ldots \exp(n)))}_{\ell \text{ compositions of } \exp}$ for some ℓ. But for each $k \in \mathbb{N}$ they have examples which have Dehn functions growing like $n \mapsto A_k(n)$. Indeed, better, they have examples with Dehn function growing like $n \mapsto A_n(n)$.

Recently, yet more extreme examples were constructed by Kharlampovich, Miasnikov & Sapir [20]. By simulating Minsky machines in groups, for every recursive function $f : \mathbb{N} \to \mathbb{N}$, they construct a finitely presented group (which also happens to be residually finite and solvable of class 3) with Dehn function growing faster than f, but with word problem solvable in polynomial time.

There are also 'naturally arising' groups which have fast growing Dehn function but an efficient (that is, polynomial-time) solution to the word problem. A first example is $\langle\, a, b \mid b^{-1}ab = a^2 \,\rangle$. Its Dehn function grows exponentially (see, for example, [4]), but the group admits a faithful matrix representation

$$a \mapsto \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \qquad b \mapsto \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix},$$

so it is possible to check efficiently when a word on $a^{\pm 1}$ and $b^{\pm 1}$ represents the identity by multiplying out the corresponding string of matrices.

A celebrated 1-relator group due to Baumslag [1] provides a more dramatic example:

$$\langle\ a, b \ \mid\ (b^{-1}a^{-1}b)\, a\, (b^{-1}ab) = a^2\ \rangle.$$

Platonov [28] proved its Dehn function grows like $n \mapsto \overbrace{\exp_2(\exp_2 \cdots (\exp_2}^{\lfloor \log_2 n \rfloor}(1)) \cdots)$, where $\exp_2(n) := 2^n$. (Earlier results in this direction are in [2, 14, 15].) Nevertheless, Miasnikov, Ushakov & Won [26] solve its word problem in polynomial time. (In unpublished work I. Kapovich and Schupp showed it is solvable in exponential time [33].)

Higman's group

$$\langle\ a, b, c, d \ \mid\ b^{-1}ab = a^2,\ c^{-1}bc = b^2,\ d^{-1}cd = c^2,\ a^{-1}da = d^2\ \rangle$$

from [19] is another example. Diekert, Laun & Ushakov [11] recently gave a polynomial time algorithm for its word problem and, citing a 2010 lecture of Bridson, claim it too has Dehn function growing like a tower of exponentials.

The groups we focus on here are yet more extreme 'natural examples.' They arose in the study of *hydra groups* by Dison & Riley [13] . Let $\theta : F(a_1, \ldots, a_k) \to F(a_1, \ldots, a_k)$ be the automorphism of the free group of rank k such that $\theta(a_1) = a_1$ and $\theta(a_i) = a_i a_{i-1}$ for $i = 2, \ldots, k$. The family

$$G_k \ :=\ \langle\ a_1, \ldots, a_k, t \ \mid\ t^{-1}a_i t = \theta(a_i)\ \ \forall i > 1\ \rangle,$$

are called *hydra groups*. Define

$$\Gamma_k \ := \ \langle \, a_1, \ldots, a_k, t, p \ \mid \ t^{-1} a_i t = \theta(a_i), \ [p, a_i t] = 1 \ \forall i \geq 1 \, \rangle,$$

which is an *HNN-extension* of G_k in which an additional *stable letter* p commutes with all elements of the subgroup $H_k := \langle a_1 t, \ldots, a_k t \rangle$. It is shown in [13] that for $k = 1, 2, \ldots$, the subgroup H_k is free of rank k and Γ_k has Dehn function growing like $n \mapsto A_k(n)$. Our second theorem is that nevertheless:

▶ **Theorem 2.** *For all k, the word problem of Γ_k is solvable in polynomial time.*

(In fact, our algorithm halts in time at most a polynomial of degree $3k^2 + k + 2$.)

3 The membership problem, subgroup distortion, and our third theorem

A geometric feature known as *distortion* is the root cause of the Dehn function of the group Γ_k of the previous section growing like $n \mapsto A_k(n)$. The massive gap described in Theorem 2 between Dehn function and the time-complexity of the word problem for Γ_k is attributable to a similarly massive gap between a *distortion function* and the time-complexity of a *membership problem*. Here are more details.

Suppose H is a subgroup of a group G and G and H have finite generating sets S and T, respectively. So G has a *word metric* $d_S(g, h)$, the length of a shortest word on $S^{\pm 1}$ representing $g^{-1}h$, and H has a word metric d_T similarly. The *distortion* of H in G is

$$\mathrm{Dist}_H^G(n) \ := \ \max\{ \, d_T(1, g) \mid g \in H \text{ with } d_S(1, g) \leq n \, \}.$$

(Distortion is defined here with respect to specific S and T, but their choices do not affect the qualitative growth of $\mathrm{Dist}_H^G(n)$.) A fast growing distortion function signifies that H 'folds back on itself' dramatically as a metric subspace of G.

The *membership problem* for H in G is to find an algorithm which, on input of a word on $S^{\pm 1}$, declares whether or not it represents an element of H.

If the word problem of G is decidable (as it is for all G_k, because, for instance, they are free-by-cyclic) and we have a recursive upper bound on $\mathrm{Dist}_H^G(n)$, then there is a brute-force solution to the membership problem for H in G. If the input word w has length n, then search through all words on $T^{\pm 1}$ of length at most $\mathrm{Dist}_H^G(n)$ for one representing the same element as w. This is, of course, likely to be extremely inefficient, and especially so for H_k in G_k as the distortion $\mathrm{Dist}_{H_k}^{G_k}$ grows like $n \mapsto A_k(n)$. Nevertheless:

▶ **Theorem 3.** *For all k, the membership problem for H_k in G_k is solvable in polynomial time.*

(The algorithm we construct to prove this halts in time at most polynomial of degree $3k^2 + k$.)

Reducing Theorem 2 to Theorem 3 is straight-forward, requiring little more than a standard result about HNN-extensions. We detail this in Section 5 of [12].

4 Comparing our methods for Theorem 1 with power circuits and straight-line programs

Our strategy compares and contrasts with those used to solve the word problem for Baumslag's group in [26] and Higman's group in [11], where *power circuits* are the key tool. Power circuits provide concise representations of integers: power circuits of 'size' n represent (some) integers up to a height-n tower of powers of 2. There are efficient algorithms to perform addition, subtraction, and multiplication and division by 2 with power-circuit representations of integers, and to declare which of two power circuits represents the larger integer.

We too use concise representations of large integers, but in place of power circuits we use strings of Ackermann functions. These have the advantage that they may represent much larger integers. After all, $A_3(n) = \exp_2^{(n-1)}(1)$ already produces a tower of exponents, and the higher rank Ackermann functions grow far faster. However, we are aware of fewer efficient algorithms to perform operations with strings of Ackermann functions than are available for power circuits: we only have Theorem 1.

Our methods also bear comparison with the work of Lohrey, Schleimer and their coauthors [17, 18, 21, 22, 23, 24, 32] on efficient computation in groups and monoids where words are given in compressed forms using *straight-line programs* and are compared and manipulated using polynomial-time algorithms due to Hagenah, Plandowski and Lohrey. For instance Schleimer obtained polynomial-time algorithms solving the word problem for free-by-cyclic groups and automorphism groups of free groups and the membership problem for the handlebody subgroup of the mapping class group in [32].

5 The hydra phenomenon: connecting the group theory to Ackermann's functions

The reason G_k are named *hydra groups* is that the extreme distortion of H_k in G_k stems from a string-rewriting phenomenon which is a reimagining of the battle between Hercules and the Lernean Hydra, a mythical beast which grew two new heads for every one Hercules severed. Think of a *hydra* as a word w on $a_1, a_2, a_3, \ldots$. Hercules fights w as follows. He removes its first letter, then the remaining letters regenerate in that for all $i > 1$, each remaining a_i becomes $a_i a_{i-1}$ (and each remaining a_1 is unchanged). This repeats. An induction on the highest index present shows that every hydra eventually becomes the empty word. (Details are in [13].) Hercules is then declared victorious. For example, the hydra $a_2a_3a_1$ is annihilated in 5 steps:

$$a_2a_3a_1 \;\to\; a_3a_2a_1 \;\to\; a_2a_1a_1 \;\to\; a_1a_1 \;\to\; a_1 \;\to\; \textit{empty word.}$$

Define $\mathcal{H}(w)$ to be the number of steps required to reduce a hydra w to the empty word. (So $\mathcal{H}(a_3a_3a_1) = 5$.) Then, for $k = 1, 2, \ldots$, define functions $\mathcal{H}_k : \mathbb{N} \to \mathbb{N}$ by $\mathcal{H}_k(n) = \mathcal{H}(a_k^n)$. It is shown in [13] that $\mathcal{H}_k$ and A_k grow at the same rate for all k, since the two families of functions exhibit a similar recursion relation.

Here is an outline of the argument from [13] as to why $\mathrm{Dist}_{H_k}^{G_k}$ grows at least as fast as $n \mapsto \mathcal{H}_k(n)$ (and so as fast as $n \mapsto A_k(n)$). When $k \geq 2$ and $n \geq 1$, there is a reduced word $u_{k,n}$ on $\{a_1t, \ldots, a_kt\}^{\pm 1}$ of length $\mathcal{H}_k(n)$ representing $a_k^n t^{\mathcal{H}_k(n)}$ in G_k on account of the hydra phenomenon. (For example, $u_{2,3} = (a_2t)^2(a_1t)(a_2t)(a_1t)^3$ equals $a_2^3t^7$ in G_2 since a_2, a_2, a_1, a_2, a_1, a_1, and a_1 are the $\mathcal{H}_2(3) = 7$ initial letters removed by Hercules as he vanquishes the hydra a_2^3.) It follows that in G_k

$$a_k^n a_2 \; ta_1 \; a_2^{-1} a_k^{-n} \;=\; u_{k,n} \, (a_2t) \, (a_1t) \, (a_2t)^{-1} \, {u_{k,n}}^{-1}.$$

The word on the left is a product of length $2n + 4$ of the generators $a_1^{\pm 1}, \ldots, a_n^{\pm 1}, t^{\pm 1}$ of G_k and that on the right is a product of length $2\mathcal{H}_k(n) + 3$ of the generators $(a_1t)^{\pm 1}, \ldots, (a_kt)^{\pm 1}$ of H_k. As H_k is free of rank k and this word is reduced, it is not equal to any shorter word on these generators.

Hydra functions and Ackermann functions grow at the same rates, but do not precisely agree. So for Theorem 3 we, in fact, need a variation of Theorem 1, namely Proposition 3.4 in [12] which concerns a recursively defined family of functions ψ_i we call *ψ-functions*. Like strings of Ackermann functions, strings of ψ-functions (which we call *ψ-words*) can concisely represent extremely large integers. We do not have a direct proof of the equivalence of this proposition to Theorem 1, but they can be proved in essentially the same ways as the defining recurrence for the ψ_i is very similar to that for the A_i. We prefer to highlight Theorem 1 here because Ackermann functions have a long history and so are of intrinsic interest.

6 An outline of our strategy for Theorem 1

Here is a sketch of the algorithm we construct in Section 2 of [12] to prove Theorem 1. A more detailed high-level description is in Section 2.2 of [12].

Suppose we have a word w on $A_0^{\pm 1}, \dots, A_k^{\pm 1}$ and we seek to determine in polynomial time whether it is valid and, if so, whether the integer $w(0)$ is negative, zero, or positive.

We will attempt to pass to successive new words $w = w_0, w_1, \dots$ that are *equivalent* to w (denoted $w \sim w_j$) in that each w_j is valid if and only if w is, and when they both are, $w(0) = w_j(0)$. These words are obtained by making substitutions such as replacing a letter A_{i+1} in w by a subword $A_i A_{i+1} A_0^{-1}$ (the recursion defining the Ackermann functions), or deleting a subword $A_i A_i^{-1}$ or $A_i^{-1} A_i$. The lengths of these w_j will all be at most a constant times the length of w, which is important for our proof that our algorithm halts in polynomial time. The aim of the substitutions is to reach a $w' \sim w$ which contains no $A_1^{-1}, \dots, A_k^{-1}$. Eliminating these letters represents progress because they denote functions which have sparse domains and so present the greatest obstacle to checking whether a word is valid.

We will look at how to make these substitutions momentarily, but first here's what happens when we have reached such a w'. Consider calculating a succession of integers beginning with 0 and ending with $w'(0)$ by evaluating $w'(0)$ letter-by-letter starting from the right. Only $A_0^{\pm 1}$ can trigger decreases in absolute value. So, to determine the sign of $w'(0)$, we can stop our evaluation if the integer calculated ever exceeds the length of w': after all, whatever sign our evaluation then has will be the sign of $w'(0)$. This threshold for the integers in our calculation allow for a polynomial time bound.

So how do we reach this w'? The rough idea is to 'cancel' each A_i^{-1} (where $i \geq 1$) in w with some A_i (if present) further to the right in w'. We do this inductively on i by manipulating suffixes of the form $\sigma = A_i^{-1} u A_i v$ such that u is a word on $A_0^{\pm 1}, \dots, A_{i-1}^{\pm 1}$ and v a word on $A_0, \dots, A_k$. A number of complications may arise. For instance, there are exceptional cases when substituting a A_{i+1} with $A_i A_{i+1} A_0^{-1}$ fails to preserve validity. Another issue is that we may have to introduce an A_i 'artificially' to cancel with an A_i^{-1}.

It is only possible to give a few details of our algorithm in the space available here. We choose to present a subroutine **BasePinch**, which serves as the base case of this inductive process of manipulating suffixes (the instance where u only contains letters $A_0^{\pm 1}$). It displays the crucial idea that allows us to operate within polynomial time: because the gaps between elements of $\mathrm{Img} A_i$ are large, we can either recognize efficiently that σ (and hence w) is invalid on account of u not being able to carry $A_i v(0) \in \mathrm{Img} A_i$ to another element of $\mathrm{Img} A_i$ (this is what the commentary on line 12 below is about), or σ is long enough that computing letter-by-letter by usual integer arithmetic is possible in polynomial time.

BasePinch will call two other subroutines (from Section 2.3 of [12]):

- **Bounds** which, on input $\ell \in \mathbb{N}$ (expressed in binary), returns in time $O(\ell)$ a list of all the (at most $(\log_2 \ell)^2$) triples of integers $(r, n, A_r(n))$ such that $r \geq 2$, $n \geq 3$, and $A_r(n) \leq \ell$.

Algorithm 1 BasePinch.

- Input a word $\sigma = A_r^{-1} u A_r v$ where $r \geq 1$, u is a word on $A_0^{\pm 1}$, and v is a word on $A_0^{\pm 1}, \ldots, A_k$.
- Either return that σ is invalid, or return a valid word $\sigma' = A_0^{l'} v \sim \sigma$ such that $\ell(\sigma') \leq \ell(\sigma) - 2$.
- Halt in time $O(\ell(\sigma)^4)$.

```
1   l := u(0)  (so  A_r^{-1} A_0^l A_r v ~ w)
    if Positive(A_r v) = Invalid, halt and return invalid
    run Positive(v) to determine whether v(0) < 0
4   if r ≥ 2 and v(0) < 0, halt and return invalid
    if l = 0, halt and return σ' := v
    if r = 1, halt and return σ' := A_0^{l/2} v if i is even or invalid otherwise
7
    we now have l ≠ 0 and r > 1
    run Positive(A_0^l A_r v) to determine if A_0^l A_r v(0) ≤ 0 (so ∉ domain of A_r^{-1})
10          if so, halt and return invalid
    run Positive(A_0^{-2|l|} A_r v) to determine whether A_r v(0) > 2|l|
            if so, halt and return invalid
13
    we now have that 0 ≤ v(0) ≤ |l| and 0 < A_r v(0) ≤ 2|l| and A_r v(0) + l ≤ 3|l|
    calculate v(0) by running Positive(A_0^{-i} v) for i = 0, 1, ..., |l|
16  run Bounds(3|l|)
    search the output of Bounds(3|l|) to find A_r v(0)
    set m := A_r v(0) + l
19  search the output of Bounds(3|l|) for c with A_r(c) = m
                                        (so c = A_r^{-1} A_0^l A_r v(0) = σ(0))
            if such a c exists, halt and return σ' := A_0^{c - v(0)} v
22          else halt and return invalid
```

- **Positive** which, on input a word w on $A_0^{\pm 1}, A_1, \ldots, A_k$ in time $O(\ell(w)^3)$ either declares w invalid or declares whether $w(0) < 0$, $w(0) = 0$, or $w(0) > 0$.

We use these properties of Ackermann functions:

▶ **Lemma 4.**

$$A_i(n) + m \leq A_i(n+m) \qquad \forall i, n, m \geq 0, \tag{1}$$

$$|A_i(n) - A_i(m)| \geq \frac{1}{2} A_i(n) \qquad \forall i \geq 2 \textit{ and } n \neq m. \tag{2}$$

The proofs follow by inductive arguments applied to the definition of an Ackermann function. Refer to Lemma 2.1 of [12] for details.

Correctness of BasePinch. Here are the salient points line-by-line.

4: If $v(0) < 0$, then σ is invalid.

5: If $r < 2$ or $v(0) \geq 0$, $A_r^{-1} A_r v \sim v$.

6: Since A_1 is the function $n \mapsto 2n$, the parity of $A_0^l A_r v(0)$ is the parity of l when $r = 1$, and determines the validity of σ.

9, 11: We know $A_0^l A_r v$ and $A_0^{-2|l|} A_r v$ are valid at these points because $A_r v$ is valid.

12: Let $q = v(0)$. For all $p \neq q$ we have $|A_r(q) - A_r(p)| \geq \frac{1}{2} A_r(q)$ by Lemma 4, and so $|A_r(q) - A_r(p)| > |l|$. If $A_r^{-1} A_0^l A_r v$ is valid, then there exists $p \in \mathbb{N}$ such that $A_r(p) = A_0^l A_r v(0) = l + A_r(q)$, but then $|A_r(p) - A_r(q)| = |l|$ for some $p \neq q$ (since $l \neq 0$), contradicting $|A_r(q) - A_r(p)| > l$. Thus w is invalid.

14: The reason $0 < A_r v(0)$ is that $r > 1$ and so $\mathrm{Img} A_r$ contains only positive integers. And $A_r v(0) \leq 2\,|l|$ because of lines 11 and 12. It follows that $v(0) \leq |l|$ because $2v(0) = A_1 v(0) \leq A_r v(0) \leq 2\,|l|$. And $v(0) \geq 0$ since $v(0)$ is in the domain of A_r, which is $\mathbb{N}$ when $r > 1$. We have $A_0^l A_r v(0) \leq 3|l|$ here because $A_r v(0) \leq 2\,|l|$ and so $A_0^l A_r v(0) \leq l + 2\,|l|$.
20: If $m = A_r v(0) + l = A_0^l A_r v(0)$ is in the domain of A_r^{-1}, then $m > 0$. And, from line 14, we know $m \leq 3\,|l|$, so this will find c if it exists. If no such c exists, σ is invalid.
21: $A_0^{c-v(0)} v(0) = c = A_r^{-1}(l + A_r v(0)) = A_r^{-1} A_0^l A_r v(0)$.

We must show that $\ell(\sigma') \leq \ell(\sigma) - 2$. In the cases of lines 5 and 6, this is immediate, so suppose $r \geq 2$. As for line 21, by Lemma 4:

$$|c - v(0)| \leq |A_r(v(0) + c - v(0)) - A_r v(0)| = |A_r(c) - A_r(v(0)| = |l|$$

from which $\ell(\sigma') \leq \ell(\sigma) - 2$ follows immediately.

The integer calculations performed by the algorithm involve integers of absolute value at most $3\ell(\sigma)$. See [12] for details.

That **BasePinch** halts in time $O(\ell(\sigma)^4)$ follows the following. **Positive** and **Bounds** halt in cubic and linear time, respectively. **BasePinch** may add a pair of positive binary numbers each at most $2\ell(\sigma)$, may determine the parity of a number of absolute value at most $\ell(\sigma)$, and may halve an even positive number less than $\ell(\sigma)$. It calls **Positive** at most $|l| + 3 \leq \ell(\sigma) + 3$ times, always on a word of length at most $2\ell(\sigma)$. It calls **Bounds** at most once and on a non-negative integer that is at most $3\ell(\sigma)$. The output of **Bounds** is then searched at most twice and has size $O((\log_2 \ell(\sigma))^2)$. ◀

7 An outline of our strategy for Theorem 3

Here is an outline of our algorithm solving the membership problem for H_k in G_k from Section 4 of [12], proving Theorem 3. For a more detailed high-level description, see Section 4.1 of [12].

Suppose w is a word on $a_1^{\pm 1}, \ldots, a_k^{\pm 1}, t^{\pm 1}$, so represents an element of G_k. To tell whether or not w represents an element of H_k, first collect all the $t^{\pm 1}$ at the front by shuffling them to the left through the word, applying $\theta^{\pm 1}$ as appropriate to the intervening a_i so that the element of G_k represented does not change. The result is a word $t^r v$ where $|r| \leq \ell(w)$ and v, a word on $a_1^{\pm 1}, \ldots, a_k^{\pm 1}$ has length at most a constant times $\ell(w)^k$ since θ is a free group automorphism of such polynomial growth.

Here is an example (one of a number in Section 4.2 of [12]). Suppose $w = a_3^4 a_2 t a_1 a_2^{-1} a_3^{-4}$. This equals tv in G_3 where $v = (a_3 a_2)^4 a_2 a_1^2 a_2^{-1} a_3^{-4}$ because $a_2 t = a_2 a_1$ and $a_3 t = a_3 a_2$.

We next look to carry the t^r back through v working from left to right, converting (if possible) what lies to the left of the power of t to a word on the generators $(a_1 t)^{\pm 1}, \ldots, (a_k t)^{\pm 1}$ of H_k. However the power of t being carried along will vary as this proceeds and, in fact, can get extremely large as a result of the hydra phenomenon. Similarly, the length of the word on the generators of H_k appearing to the left can be impractically long. For instance, in our example, the calculation outlined in Section 5 shows that w equals an element of the subgroup H_3 of G_3 which has length $2^{47} \cdot 3 - 1$ as a reduced word on the generators $(a_1 t)^{\pm 1}$, $(a_2 t)^{\pm 1}$, $(a_3 t)^{\pm 1}$ of H_3.

So, instead of keeping track of the power of t directly, we record it as a word on ψ-functions (the functions that are analogues of Ackermann functions, as we explained in Section 5). Roughly speaking, checking whether this process ever gets stuck (in which case $w \notin H_k$) amounts to checking whether an associated ψ-word is valid. If the end of the word is reached, we then have a word on $(a_1t)^{\pm 1}, \ldots, (a_kt)^{\pm 1}$ times some power of t, where the power is represented by a ψ-word whose length is at most a polynomial function of the length of w. We then determine whether or not $w \in H_k$ by checking whether or not that ψ-word represents 0. Both tasks can be accomplished suitably efficiently thanks to Proposition 3.4 in [12] (a variation of Theorem 1 as we explained in Section 5).

A complication is that we do not carry the power of t through from left to right one letter at a time. Rather, we partition v into subwords we call *rank k-pieces* and are determined by the locations of the a_k and a_k^{-1} in v. Each contains at most one a_k and at most one a_k^{-1}, and if the a_k is present in a piece, it is the first letter of that piece, and if the a_k^{-1} is present, it is the last letter. For instance, in our example $k = 3$ and $v = (a_3a_2)(a_3a_2)(a_3a_2)(a_3a_2^2a_1^2a_2^{-1}a_3^{-1})(a_3^{-1})(a_3^{-1})(a_3^{-1})$. We look to carry the power of t through v one piece at a time. Lemma 6.2 of [13] details how $v \in \bigcup_{s\in\mathbb{Z}} H_kt^s$ if and only if this is possible.

Whether the power of t can be carried through a piece $a_k^{\varepsilon_1}ua_k^{-\varepsilon_2}$ (here, $\varepsilon_1, \varepsilon_2 \in \{0, 1\}$ and u is a reduced word on $a_1^{\pm 1}, \ldots, a_{k-1}^{\pm 1}$) depends on u in a manner that can be recursively analyzed by decomposing u into pieces with respect to the locations of the $a_{k-1}^{\pm 1}$ it contains. The main technical result behind our algorithm is our 'Piece Criterion' (Proposition 4.10 in [12]). This determines whether a power t^r can pass through a piece π—that is, whether $t^r\pi \in H_kt^s$ for some $s \in \mathbb{Z}$—and, if it can, how to represent s by a ψ-word. The way this plays out in our example is:

$$
\begin{aligned}
t(a_3a_2) &\in H_kt^{f_1(0)} && \text{where} \quad f_1 = \psi_1\psi_1^{-1},\\
t^{f_1(0)}(a_3a_2) &\in H_kt^{f_2(0)} && \text{where} \quad f_2 = \psi_2\psi_3f_1,\\
t^{f_2(0)}(a_3a_2) &\in H_kt^{f_3(0)} && \text{where} \quad f_3 = \psi_2\psi_3f_2,\\
t^{f_3(0)}(a_3a_2^2a_1^2a_2^{-1}a_3^{-1}) &\in H_kt^{f_4(0)} && \text{where} \quad f_4 = \psi_3^{-1}\psi_2^{-1}(\psi_1)^2\psi_2^2\psi_3f_3.\\
t^{f_4(0)}(a_3^{-1}) &\in H_kt^{f_5(0)} && \text{where} \quad f_5 = \psi_3^{-1}f_4,\\
t^{f_5(0)}(a_3^{-1}) &\in H_kt^{f_6(0)} && \text{where} \quad f_6 = \psi_3^{-1}f_5,\\
t^{f_6(0)}(a_3^{-1}) &\in H_kt^{f_7(0)} && \text{where} \quad f_7 = \psi_3^{-1}f_6.
\end{aligned}
$$

(The integers encoded here are $f_1(0) = 0$, $f_2(0) = -3$, $f_3(0) = -45$, $f_4(0) = -46$, $f_5(0) = -4$, $f_6(0) = -1$, and $f_7(0) = 0$. The conclusion is that $w \in H_3$ since $f_7(0) = 0$.)

Like in the previous section, we do not have space here to present many of the details, and so will only give an illustrative subroutine, namely '$\mathbf{Back}_m$.' This attempts to pass a power t^r through a rank m-piece which has the special form $ua_m^{-\epsilon_2}$ where $\epsilon_2 \in \{0, 1\}$, u is a word $a_1^{\pm 1} \cdots a_{m-1}^{\pm 1}$ and $m \geq 3$. There are several precursors to the construction of $\mathbf{Back}_m$:

- The construction is inductive on m. $\mathbf{Back}_m$ calls an algorithm $\mathbf{Push}_{m-1}$ (of Section 4.5 of [12]) which takes as input a word v on $a_0^{\pm 1}, \ldots, a_{m-1}^{\pm 1}$ and a ψ-word f representing an integer, and declares whether $t^{f(0)}v \in H_kt^s$ for some $s \in \mathbb{Z}$; if so, $\mathbf{Push}_{m-1}$ also returns a ψ-word g so that $t^{f(0)}v \in H_kt^{g(0)}$.
- The Piece Criterion (Proposition 4.10 in [12]) stipulates (in particular) that if $t^rua_m^{-\epsilon_2} \in H_kt^s$ for some $s \in \mathbb{Z}$, exactly one of the following three conditions must hold:
  (a) $\epsilon_2 = 0$ and $t^rua_m^{-\epsilon_2} = t^ru \in H_kt^s$ (the trivial case).
  (b) $\epsilon_2 = 1$, $s \leq 0$ and $t^ru \in H_kt^{\psi_m(s)}$.
  (c) $\epsilon_2 = 1$, $s > 0$, $t^rua_m^{-1}\theta^s(a_m) \in H_kt^{s-1}$ and $\theta^{s-1}(a_m^{-1})$ is a suffix of $ua_m^{-\epsilon_2}$.
  (Here, θ is the free group automorphism we defined in Section 2.)
- A routine $\mathbf{Prefix}_m$ (of Section 4.5 of [12]) inputs a rank-m piece $\pi = a_mua_m^{-\epsilon_2}$ where $m \geq 3$. It returns the largest integer $i > 0$ (if any) such that $\theta^{i-1}(a_m)$ is a prefix of π and halts in time in $O(\ell(\pi)^2)$.

Algorithm 2 $\mathbf{Back}_m$.

∘ Input a rank-m piece $\pi = ua_m^{-\epsilon_2}$ with $m \geq 3$ (so u is a reduced word on $a_1^{\pm1}, \ldots, a_{m-1}^{\pm1}$ and $\epsilon_2 \in \{0,1\}$) and a valid ψ-word f on $\psi_1^{\pm1}, \ldots, \psi_k^{\pm1}$. Let $r := f(0)$.
∘ Declare whether or not $t^r\pi \in \bigcup_{s\in\mathbb{Z}} H_k t^s$. And, if it is, return a valid ψ-word f' such that $t^{f(0)}\pi \in H_k t^{f'(0)}$, $\ell(f') \leq \ell(f) + 2(m-1)\ell(\pi) + 1$ and rank $(f') \leq \max\{\text{rank } (f), m\}$.
∘ Halt in time $O((\ell(\pi) + \ell(f))^{2m+k})$.

```
   run Push_{m-1}(u, f) to test whether or not t^r u ∈ ⋃_{s∈Z} H_k t^s
2    if so, let g be the valid ψ-word it outputs such that t^r u ∈ H_k t^{g(0)}
   if ε_2 = 0,
     if t^r u ∈ H_k t^{g(0)} (so, (a) of the Piece Criterion holds with s = g(0)),
5            return f' := g
     else declare  t^r π ∉ ⋃_{s∈Z} H_k t^s
     halt
8
   we now have that ε_2 = 1
   run Psi(ψ_m^{-1} g) to check validity of ψ_m^{-1} g (so whether g(0) ∈ Img ψ_m)
11   and, if so, to check  ψ_m^{-1} g(0) ≤ 0
             (i.e. whether (2) of the Criterion holds with s = ψ_m^{-1} g(0))
     if so, halt and return f' := ψ_m^{-1} g
14
   run Prefix_m(π^{-1}) to determine the maximum i (if any)
         such that a_{m-1}^{-1} θ^{i-1}(a_m^{-1}) is a suffix of π
17   if there is no such i, halt and declare t^r π ∉ ⋃_{s∈Z} H_k t^s
   for s = 1 to i:
     run Push_{m-1}(u', f) where u' is the reduced word representing u a_m^{-1} θ^s(a_m)
20     if it outputs a ψ-word h, run Psi(ψ_1^{s-1} h) to check if h(0) = s - 1
         if so halt and return f' := ψ_1 h
   declare that t^{f(0)} w ∉ ⋃_{s∈Z} H_k t^s
```

- **Psi** is our algorithm (of Section 3.3 of [12]) determining in polynomial time whether a ψ-word is valid and, if so, whether the integer it represents is negative, zero, or positive. We discussed its Ackermann-function analogue in the previous section.

Proof of correctness. Here is our justification line-by-line.

2: It follows from the workings of $\mathbf{Push}_{m-1}$ (proved in Section 4.5 of [12]) that $\ell(g) \leq \ell(u) + \ell(f)$ and **rank** $(g) \leq \max\{\mathbf{rank}\ (f), m\}$.

3–6: $\mathbf{Push}_{m-1}$ in lines 1–2 tests whether or not $t^r u$ is in $\bigcup_{s\in\mathbb{Z}} H_k t^s$ and, if so, it identifies the s such that $t^r u \in H_k t^s$. The Piece Criterion then tells us that the answer to whether $t^r\pi \in \bigcup_{s\in\mathbb{Z}} H_k t^s$ is the same, and if affirmative the s agrees. By our comment on line 2, $\ell(f') \leq \ell(f) + \ell(u) = \ell(f) + \ell(\pi)$, and $\text{rank}(f') \leq \max\{\text{rank}(f), m\}$, as required.

10–13: Again, we refer back to lines 1–2 for whether or not $t^r u$ is in $\bigcup_{s_0\in\mathbb{Z}} H_k t^{s_0}$. Assuming that it is, in fact, in $H_k t^{g(0)}$, then Condition *2*, is satisfied if and only if $g(0) = \psi_m(s)$ for some $s \leq 0$. And that is checked in line 10. The Piece Criterion then tells us that the answer to this is the same as the answer to whether $t^r u \in \bigcup_{s\in\mathbb{Z}} H_k t^s$, and, if affirmative, the s agrees. By our comment on line 2, $\ell(f') = \ell(g) + 1 \leq \ell(f) + \ell(u) + 1 = \ell(f) + \ell(\pi)$ and $\text{rank}(f') \leq \max\{\text{rank}(f), m\}$, as required.

16–21: The aim here is to determine whether Condition *3* holds—that is, whether

$$t^r u a_m^{-1} \theta^s(a_m) \in H_k t^{s-1}$$

and $a_{m-1}^{-1}\theta^{s-1}(a_m^{-1})$ is a suffix of π for some $s > 0$—and, if so, output a ψ-word f' such that $f'(0) = s$. (This s must be unique, if it exists, because, by the Criterion, it is the s such that $t^r\pi \in H_k t^s$, and we know that is unique.)

The possibilities for s are limited to the range $1, \ldots, i$ by the suffix condition and the requirement that $s > 0$, where i is as found in line 16 and must be at most $\ell(\pi)$. If there is such a suffix $a_{m-1}^{-1}\theta^{i-1}(a_m^{-1})$ of π, then $a_{m-1}^{-1}\theta^{s-1}(a_m^{-1})$ is a suffix of π for all $s \in \{1, \ldots, i\}$. If there is no such suffix, then Condition *3* fails, and, as we know at this point that Conditions *1* and *2* also fail, we declare in line 17 that (by the Criterion), $t^r\pi \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$.

For each s in the range $1, \ldots, i$, lines 18–21 address the question of whether or not $t^r u a_m^{-1}\theta^s(a_m) \in H_k t^{s-1}$. First $\mathbf{Push}_{m-1}$ is called, which can be done because, on freely reducing $u a_m^{-1}\theta^s(a_m)$, the a_m^{-1} cancels with the a_m at the start of $\theta^s(a_m)$ to give a word of rank at most $m-1$. $\mathbf{Push}_{m-1}$ either tells us that $t^r u a_m^{-1}\theta^s(a_m) \notin \bigcup_{s'\in\mathbb{Z}} H_k t^{s'}$, or it gives a ψ-word h such that $t^r u a_m^{-1}\theta^s(a_m) \in H_k t^{h(0)}$. In the latter case, **Psi** is then used to test whether or not $h(0) = s - 1$.

By the specifications of $\mathbf{Push}_{m-1}$, $\ell(h) \leq \ell(f) + 2(m-1)\ell(u')$. And, as $\pi = u a_m^{-1}$ has suffix $\theta^{s-1}(a_m^{-1})$, when we form u' by freely reducing $u a_m^{-1}\theta^s(a_m)$, at least half of $\theta^s(a_m) = \theta^{s-1}(a_m)\theta^{s-1}(a_{m-1})$ cancels into π. So $\ell(u') \leq \ell(\pi)$, and

$$\ell(f') \ = \ \ell(h) + 1 \ \leq \ \ell(f) + 2(m-1)\ell(u') + 1 \ \leq \ \ell(f) + 2(m-1)\ell(\pi) + 1,$$

as required. Also, it is immediate that $\mathrm{rank}(f') \leq \max\{\mathrm{rank}(f), m\}$, as required.

22: At this point, we know *1*, *2* and *3* fail for all $s \in \mathbb{Z}$, so $t^r\pi \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$.

$\mathbf{Back}_m$ runs $\mathbf{Push}_{m-1}(u, f)$ once (with $\ell(u) \leq \ell(\pi)$), $\mathbf{Psi}(\psi_m^{-1}g)$ at most once (with $\ell(g) \leq \ell(\pi) + \ell(f)$), $\mathbf{Prefix}_m(\pi^{-1})$ at most once, $\mathbf{Push}_{m-1}(u', f)$ at most $i \leq \ell(\pi)$ times (with $\ell(u') < \ell(\pi)$), and $\mathbf{Psi}(\psi_1^{s-1}h)$ at most $i \leq \ell(\pi)$ times (with $1 \leq s \leq \ell(\pi)$ and $\ell(h) < \ell(f) + \ell(\pi)$). Other operations such as free reductions of words etc. do not contribute significantly to the running time. Referring to the specifications of $\mathbf{Push}_{m-1}$, **Psi**, and $\mathbf{Prefix}_m$, we see that they (respectively) contribute:

$$\begin{aligned}\ell(\pi)O((\ell(\pi)+\ell(f))^{2(m-1)+k+1}) + \ell(\pi)O((\ell(f)+2\ell(\pi))^{4+k}) + O(\ell(\pi)^2)\\ = \ O((\ell(\pi)+\ell(f))^{2m+k})\end{aligned}$$

which is the claimed bound on the halting time of $\mathbf{Back}_m$. ◀

There is also an algorithm $\mathbf{Front}_m$ which takes a rank-m-piece π and a ψ-word f and determines (in a manner similar to $\mathbf{Back}_m$, see [12] Algorithm 4.2 for details) whether $t^{f(0)}$ can efficiently pass an initial a_m (if it exists) in π. If so, $\mathbf{Front}_m$ outputs a word of the form $u a_m^{-1}$ suitable for input into $\mathbf{Back}_m$ and a valid ψ word g such that checking whether $t^{f(0)}\pi \in Ht^s$ for some $s \in \mathbb{Z}$ is equivalent to checking whether $t^{g(0)} u a_m^{-1} \in Ht^s$ for some $s \in \mathbb{Z}$. If $t^{f(0)}$ does not pass through an initial a_m of π in one of three ways, Proposition 4.10 in [12] says that $t^{f(0)}\pi \notin H_k t^S$ for all $s \in \mathbb{Z}$. Putting together the algorithm $\mathbf{Front}_m$ with $\mathbf{Back}_m$ and implicitly $\mathbf{Push}_{m-1}$, we can construct the algorithm $\mathbf{Push}_m$. That way, given a word v on $a_1^{\pm 1}, \ldots, a_m^{\pm 1}$ and a ψ-word f, we have a polynomial time algorithm to determine whether or not $t^{f(0)}v \in H_k t^s$ and if so, to give a ψ-word g such that $g(0) = s$. We can then use **Psi** to determine whether g represents zero, and so whether $t^{f(0)}v$ represents an element of H_k.

A Reference to the technical details

The technical details are set out in full in *Taming the hydra: the word problem and extreme integer compression* [12], which is available from the arXiv repository at `http://arxiv.org/abs/1509.02557`.

References

1. G. Baumslag. A non–cyclic one–relator group all of whose finite quotients are cyclic. *J. Austral. Math. Soc.*, 10:497–498, 1969.
2. A. A. Bernasconi. *On HNN–extensions and the complexity of the word problem for one-relator groups*. PhD thesis, University of Utah, 1994. `http://www.math.utah.edu/~sg/Papers/bernasconi-thesis.pdf`.
3. W. W. Boone. Certain simple unsolvable problems in group theory I, II, III, IV, V, VI. *Nederl. Akad. Wetensch Proc. Ser. A.* **57**, 231–236, 492–497 (1954), **58**, 252–256, 571–577 (1955), **60**, 22-26, 227-232 (1957).
4. N. Brady, T. R. Riley, and H. Short. *The geometry of the word problem for finitely generated groups*. Advanced Courses in Mathematics CRM Barcelona. Birkhäuser–Verlag, 2007.
5. M. R. Bridson. The geometry of the word problem. In M. R. Bridson and S. M. Salamon, editors, *Invitations to Geometry and Topology*, pages 33–94. O.U.P., 2002.
6. M. R. Bridson and A. Haefliger. *Metric Spaces of Non-positive Curvature*. Number 319 in Grundlehren der mathematischen Wissenschaften. Springer Verlag, 1999.
7. D. E. Cohen. The mathematician who had little wisdom: a story and some mathematics. In *Combinatorial and geometric group theory (Edinburgh, 1993)*, volume 204 of *London Math. Soc. Lecture Note Ser.*, pages 56–62. Cambridge Univ. Press, Cambridge, 1995.
8. D. E. Cohen, K. Madlener, and F. Otto. Separating the intrinsic complexity and the derivational complexity of the word problem for finitely presented groups. *Math. Logic Quart.*, 39(2):143–157, 1993.
9. M. Dehn. Über unendliche diskontunuierliche Gruppen. *Math. Ann.*, 71:116–144, 1912.
10. M. Dehn. *Papers on group theory and topology*. Springer-Verlag, New York, 1987. Translated from the German and with introductions and an appendix by J. Stillwell, With an appendix by O. Schreier.
11. V. Diekert, J. Laun, and A. Ushakov. Efficient algorithms for highly compressed data: The Word Problem in Higman's group is in P. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 218–229, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.STACS.2012.218`.
12. W. Dison, E. Einstein, and T. R. Riley. Taming the hydra: the word problem and efficient calculation with ackermann-compressed integers. URL: `http://arxiv.org/abs/1509.02557`.
13. W. Dison and T. R. Riley. Hydra groups. *Comment. Math. Helv.*, 88(3):507–540, 2013. `doi:10.4171/CMH/294`.
14. S. M. Gersten. Isodiametric and isoperimetric inequalities in group extensions. Preprint, University of Utah, 1991.
15. S. M. Gersten. Isoperimetric and isodiametric functions of finite presentations. In G. Niblo and M. Roller, editors, *Geometric group theory I*, number 181 in LMS lecture notes. Camb. Univ. Press, 1993.
16. M. Gromov. Asymptotic invariants of infinite groups. In G. Niblo and M. Roller, editors, *Geometric group theory II*, number 182 in LMS lecture notes. Camb. Univ. Press, 1993.

17 N. Haubold and M. Lohrey. Compressed word problems in HNN-extensions and amalgamated products. *Theory Comput. Syst.*, 49(2):283–305, 2011. `doi:10.1007/s00224-010-9295-2`.

18 N. Haubold, M. Lohrey, and C. Mathissen. Compressed decision problems for graph products and applications to (outer) automorphism groups. *Internat. J. Algebra Comput.*, 22(8):1240007, 53, 2012. `doi:10.1142/S0218196712400073`.

19 G. Higman. A finitely generated infinite simple group. *J. London Math. Soc.*, 26:61–64, 1951.

20 O. Kharlampovich, A. Miasnikov, and M. Sapir. Algorithmically complex residually finite groups. `arXiv:1204.6506`.

21 M. Lohrey. Word problems and membership problems on compressed words. *SIAM J. Comput.*, 35(5):1210–1240, 2006. `doi:10.1137/S0097539704445950`.

22 M. Lohrey. Compressed word problems for inverse monoids. In *Mathematical foundations of Computer Science 2011*, volume 6907 of *Lecture Notes in Comput. Sci.*, pages 448–459. Springer, Heidelberg, 2011. `doi:10.1007/978-3-642-22993-0_41`.

23 M. Lohrey. *The compressed word problem for groups.* Springer Briefs in Mathematics. Springer, New York, 2014. `doi:10.1007/978-1-4939-0748-9`.

24 M. Lohrey and S. Schleimer. Efficient computation in groups via compression. In *Proc. Computer Science in Russia (CSR 2007)*, volume 4649 of *Lecture Notes in Computer Science*, pages 249–258. Springer, 2007.

25 K. Madlener and F. Otto. Pseudonatural algorithms for the word problem for finitely presented monoids and groups. *J. Symbolic Comput.*, 1(4):383–418, 1985.

26 A. G. Miasnikov, A. Ushakov, and D. W. Won. Power circuits, exponential algebra, and time complexity. *Internat. J. Algebra Comput.*, 22(6):1250047, 51, 2012. `doi:10.1142/S0218196712500476`.

27 P. S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudt Mat. Inst. Stkelov*, 44:1–143, 1955.

28 A. N. Platonov. An isoperimetric function of the Baumslag–Gersten group. *Vestnik Moskov. Univ. Ser. I Mat. Mekh.*, 3:12–17, 70, 2004. Translation in Moscow Univ. Math. Bull. 59 (2004).

29 T. R. Riley. What is a Dehn function? Chapter for the forthcoming *Office hours with a geometric group theorist*, Princeton University Press, M. Clay and D. Magalit, eds.

30 H. E. Rose. *Subrecursion: functions and hierarchies*, volume 9 of *Oxford Logic Guides*. The Clarendon Press Oxford University Press, New York, 1984.

31 M. Sapir. Asymptotic invariants, complexity of groups and related problems. *Bull. Math. Sci.*, 1(2):277–364, 2011. `doi:10.1007/s13373-011-0008-1`.

32 S. Schleimer. Polynomial-time word problems. *Comment. Math. Helv.*, 83(4):741–765, 2008. `doi:10.4171/CMH/142`.

33 P. Schupp. personal communication.