TAMING THE HYDRA: THE WORD PROBLEM AND EXTREME INTEGER COMPRESSION

W. DISON, E. EINSTEIN AND T.R. RILEY

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 & 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.

2010 Mathematics Subject Classification: 20F10, 20F65, 68W32, 68Q17
Key words and phrases: Ackermann functions, subgroup distortion, Dehn function, hydra, word problem, membership problem, polynomial time

CONTENTS

1. Introduction . . . 2
1.1. Ackermann functions and compressed integers . . . 2
1.2. The word problem and Dehn functions . . . 2
1.3. The membership problem and subgroup distortion . . . 5
1.4. The hydra phenomenon . . . 5
1.5. The organization of this article and an outline of our strategies . . . 6
1.6. Comparison with power circuits and straight-line programs . . . 7
2. Efficient calculation with Ackermann-compressed integers . . . 8
2.1. Preliminaries . . . 8
2.2. Examples and general strategy . . . 9
2.3. Our algorithm . . . 15
3. Efficient calculation with ψ-compressed integers . . . 24
3.1. ψ-functions and ψ-words . . . 24
3.2. An example . . . 27
3.3. Our algorithm in detail . . . 27

Date: October 6, 2018.
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) during the writing of this article.

4. An efficient solution to the membership problem for hydra groups 33
4.1. Our algorithm in outline 33
4.2. Examples 34
4.3. Constraining cancellation 37
4.4. The Piece Criterion 51
4.5. Our algorithm in detail 56
5. Conclusion 61
References 62

1. Introduction

1.1. **Ackermann functions and compressed integers.** Ackermann functions $A_i : \mathbb{N} \to \mathbb{N}$ are a family of increasingly fast-growing functions beginning $A_0 : n \mapsto n+1$, $A_1 : n \mapsto 2n$, and $A_2 : n \mapsto 2^n$, and with subsequent A_{i+1} defined recursively so that $A_{i+1}(n+1) = A_iA_{i+1}(n)$ and $A_{i+1}(0) = 1$. (More details follow in Section 2.)

Starting with zero and successively applying a few such 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

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

In this way Ackermann functions provide highly compact representations for some very large numbers.

In principle, we could compute with these representations by evaluating the integers they represent and then using standard integer arithmetic, but this can be monumentally inefficient because of the sizes of the integers. We will explain how to calculate efficiently in a rudimentary way with such representations of integers:

Theorem 1. *Fix an integer $k \geq 0$. There is a polynomial-time algorithm, which on input a word w on $A_0^{\pm1}, \ldots, A_k^{\pm1}$, declares whether or not $w(0)$ represents an integer, and if so whether $w(0) < 0$, $w(0) = 0$ or $w(0) > 0$.*

(The manner in which $w(0)$ might fail to represent an integer is that as it is evaluated from right to left, an $A_i^{\pm1}$ is applied to an integer outside its domain. Details are in Section 2.1. In fact our algorithm halts in time bounded above by a polynomial of degree $4+k$—see Section 2.3. We have not attempted to optimize the degrees of the polynomial bounds on time complexity here or elsewhere in this article.)

1.2. **The word problem and Dehn functions.** Our interest in Theorem 1 originates in group theory. 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 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 $\left\{a_1^{-1}, a_2^{-1}, \ldots\right\}$. The inverse map is the involution defined on $\left\{a_1^{\pm 1}, a_1^{\pm 2}, \ldots\right\}$ that maps $a_i \mapsto a_i^{-1}$ and $a_i^{-1} \mapsto a_i$ for all i. Write $w = w(a_1, a_2, \ldots)$ when w is a word on the letters $a_1^{\pm 1}, a_2^{\pm 1}, \ldots$. 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 $x_i \in \left\{a_1^{\pm 1}, a_2^{\pm 1}, \ldots\right\}$. 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, 30, 31].

If $\mathrm{Area}(n)$ is bounded above by a recursive function $f(n)$, then there is a 'brute force' algorithm to solve the word problem: to tell whether or not a given word w represents the identity, search through 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, 28].) 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 made up of the letters $a^{\pm 1}$ and $b^{\pm 1}$, 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 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 the first examples. in a series of papers [7, 8, 26] where Dehn functions were first defined. 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 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 $n \mapsto \exp^{(\ell)}(n)$ 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, 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

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

and 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 [29] 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 [27] 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 in this article are yet more extreme 'natural examples'. They arose in the study of *hydra groups* by Dison & Riley [12] . 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*. Take *HNN-extensions*

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

of G_k where the 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 [12] that for $k = 1, 2, \ldots$, the subgroup H_k is free of rank k and Γ_k has Dehn function growing like $n \mapsto A_k(n)$. Here we prove that nevertheless:

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

(In fact, our algorithm halts within time bounded above by a polynomial of degree $3k^2 + k + 2$—see Section 5.)

1.3. **The membership problem and subgroup distortion.** *Distortion* is the root cause of the Dehn function of Γ_k growing like $n \mapsto A_k(n)$. The massive gap 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.*

(Our algorithm actually halts within time bounded above by a polynomial of degree $3k^2 + k$—see Section 5.) We will use this to prove Theorem 2.

1.4. **The hydra phenomenon.** 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 [12].) Hercules is then declared victorious. For example, the hydra $a_2 a_3 a_1$ is annihilated in 5 steps:

$$a_2 a_3 a_1 \ \to \ a_3 a_2 a_1 \ \to \ a_2 a_1 a_1 \ \to \ a_1 a_1 \ \to \ a_1 \ \to \ \textit{empty word}.$$

Define $\mathcal{H}(w)$ to be the number of steps required to reduce a hydra w to the trivial 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 [12] that $\mathcal{H}_k$ and A_k grow at the same rate for all $k = 1, 2, \ldots$ since the two families exhibit a similar recursion relation.

Here is an outline of the argument from [12] 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.) This can be used to show 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 $\{a_1, \ldots, a_n, t\}^{\pm 1}$ of G_k and that on the right is a product of length $2\mathcal{H}_k(n) + 3$ of the generators $\{a_1t, \ldots, a_kt\}^{\pm 1}$ 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.

1.5. **The organization of this article and an outline of our strategies.** We prove Theorem 1 in Section 2. Here is an outline of the algorithm we construct. Given a word $w(A_0, \ldots, A_k)$ we attempt to pass to successive new words w' that are *equivalent* to w in that $w'(0)$ represents an integer if and only if $w(0)$ does, and when they both do, $w(0) = w'(0)$. These words are obtained by making substitutions that, for instance, replace a letter A_{i+1} in w by a subword $A_iA_{i+1}A_0^{-1}$ (this substitution stems from the recursion defining Ackermann functions), or we delete a subword $A_iA_i^{-1}$ or $A_i^{-1}A_i$. The aim of these changes is to eliminate all the letters $A_1^{-1}, \ldots, A_k^{-1}$ in w, as these present the greatest obstacle to checking whether such a word represents an integer. Once no $A_1^{-1}, \ldots, A_k^{-1}$ remain in w', when calculating $w'(0)$ letter-by-letter starting from the right, only $A_0^{\pm 1}$ can trigger decreases in absolute value. So to determine the sign of $w'(0)$ it suffices to evaluate $w'(0)$ letter-by-letter from the right, stopping if the integer calculated ever exceeds the length of w'.

In order to reach such a w' we 'cancel' away letters A_i^{-1} with some A_i somewhere further to the right in the word. We do this by manipulating suffixes of the form $A_i^{-1}uA_iv$ such that $u = u(A_0, \ldots, A_{i-1})$. Such suffixes either admit substitutions to make a similar suffix with the A_i^{-1} and A_i eliminated, or they can be recognized not to evaluate to an integer because u cannot carry the element $A_iv(0) \in \mathrm{Img}\, A_i$ to another element of $\mathrm{Img}\, A_i$ since the gaps between elements of $\mathrm{Img}\, A_i$ are large.

A number of difficulties arise. For instance, there are exceptional cases when replacing A_{i+1} by $A_iA_{i+1}A_0^{-1}$ fails to preserve validity. Another issue is that we must ensure that the process terminates, and so we may, for example, have to introduce an A_i 'artificially' to cancel with some A_i^{-1}.

To show that our algorithm halts in polynomial time, we argue that the lengths of the successive words remain bounded by a constant times $\ell(w)$ (the length of w), and integer arithmetic operations performed only ever involve integers of absolute value at most $3\ell(w)$.

The group theory in this paper (specifically Theorem 3) actually requires a variant of Theorem 1 (specifically, Proposition 3.4). Accordingly, in Section 3 we introduce a family of functions which we call ψ-functions, which are closely related to Ackermann functions, and we adapt the earlier results and proofs to these. (We believe Theorem 1 is of intrinsic interest because Ackermann functions are well-known and efficient computation with this form of highly compressed integers is novel. This is why we do not present Proposition 3.4 only.)

We give a polynomial-time solution to the membership problem for H_k in G_k in Section 4.1, proving Theorem 3. Here is an outline of our algorithm. Suppose $w(a_1, \ldots, a_k, t)$ is a word representing 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 $|r| \le \ell(w)$ and $v = v(a_1, \ldots, a_k)$ has length at most a constant times $\ell(w)^k$. Then 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 $a_1 t, \ldots, a_k t$ of H_k. Some examples can be found in Section 4.2.

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. So instead of keeping track of the power directly, we record it as a word on ψ-functions. Very 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 $a_1 t, \ldots, a_k t$ times some power of t, where the power is represented by a ψ-word. 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.

A complication is that the power of t is not carried through from left to right one letter at a time. Rather, v is partitioned into subwords which we call *pieces*. These pieces 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 it the a_k^{-1} is present, it is the last letter. The power of t is, in fact, carried through one piece at a time. Whether it can be carried through a piece $a_k^{\varepsilon_1} u a_k^{-\varepsilon_2}$ (here, $\varepsilon_1, \varepsilon_2 \in \{0, 1\}$ and $u = u(a_1, \ldots, a_{k-1})$ is reduced) 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 the correctness of our algorithm is the 'Piece Criterion' (Proposition 4.10), which also serves to determine whether a power t^r can pass through a piece π—that is, whether $t^r \pi = \sigma t^s$ for some $\sigma \in H_k$ and some $s \in \mathbb{Z}$—and, if it can, how to represent s by an ψ-word.

Reducing Theorem 2 to Theorem 3 is relatively straight-forward. It requires little more than a standard result about HNN-extensions, as we will explain in Section 5.

1.6. **Comparison with power circuits and straight-line programs.** Our methods compare and contrast with those used to solve the word problem for Baumslag's group in [27] and Higman's group in [11], where *power circuits* are the key tool. Power circuits provide concise representations of integers. Those of size n represent (some) integers up to size 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].

2. Efficient calculation with Ackermann-compressed integers

2.1. **Preliminaries.** Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. *Ackermann functions* $A_0, A_1 : \mathbb{Z} \to \mathbb{Z}$ and $A_i : \mathbb{N} \to \mathbb{N}$ for $i = 2, 3, \ldots$ are defined recursively by

(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 $i \geq 1$.

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, reflecting the definitions of the functions ψ_i to come in Section 4.1. 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.

	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	$\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\\}65536$	$\cdots$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$			

For all $i \geq 2$ and $n \geq 1$, $A_i(n) = A_{i-1}^n(1)$ by repeatedly applying (iv) and using $A_i(0) = 1$. So for all $n \geq 0$, $A_2(n) = 2^n$ and $A_3(n)$ is a n-fold iterated power of 2, in other words, a tower of powers of 2 of height n. The recursion (iv) causes the functions' extraordinarily fast growth. Indeed, because of the increasing nesting of the recursion, the A_i represent the successive graduations in a hierarchy of all primitive recursive functions due to Grzegorczyk.

The functions A_i are all strictly increasing and hence injective (see Lemma 2.1). So they 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) $A_i^{-1} : \operatorname{Img} A_i \to \mathbb{N}$ for all $i > 1$.

Parts (1–7) of the following lemma are adapted from Lemma 2.1 of [12] with modifications to account for the fact that A_0 is defined as $n \mapsto n + 1$ here rather than $n \mapsto n + 2$. Part (8) quantifies the spareness of the image of $A_2, A_3, \ldots$ in a way that will be vital to our proof of Theorem 1 (specifically, in our proof the correctness of the subroutine **`BasePinch`**). It will tell us that if $u = u(A_1, \ldots, A_{k-1})$ and $uA_k(n) \in \operatorname{Img} A_k$ but $uA_k(n) \neq A_k(n)$, then $\ell(u)$ must be relatively large.

Lemma 2.1.

$$
\begin{array}{llcll}
(1) & A_i(1) & = & 2 & \forall i \geq 0, \\
(2) & A_i(2) & = & 4 & \forall i \geq 1, \\
(3) & A_i(n) & \leq & A_{i+1}(n) & \forall i \geq 1; n \geq 0, \\
(4) & A_i(n) & < & A_i(n+1) & \forall i, n \geq 0, \\
(5) & n & \leq & A_i(n) & \forall i, n \geq 0,
\end{array}
$$

(*with equality in* (5) *if and only if* $i = 1$ *and* $n = 0$)

$$
\begin{array}{llcll}
(6) & A_i(n) + A_i(m) & \leq & A_i(n+m) & \forall i, n, m \geq 1, \\
(7) & A_i(n) + m & \leq & A_i(n+m) & \forall i, n, m \geq 0, \\
(8) & |A_i(n) - A_i(m)| & \geq & \frac{1}{2} A_i(n) & \forall i \geq 2 \textit{ and } n \neq m.
\end{array}
$$

Proof. Equations (1) and (2) follow from $A_{i+1}(n+1) = A_i A_{i+1}(n)$ by induction on i. It is easy to check that (3) holds if $i = 1$ or if $n = 0$ and that (4) and (5) hold if $i = 0$, if $i = 1$ or if $n = 0$. It is clear (6) holds if $i = 1$. The inequality (7) holds if $i = 0$, $i = 1$ or $m = 0$. The inductive arguments for the above inequalities are then identical to the corresponding ones in Lemma 2.1 of [12]. For (8), note that the result is true when $i = 2$ as $A_2(n) = 2^n$ for all $n \in \mathbb{N}$ and, given how each of the successive rows is constructed from those preceding them, it follows that it is true for all $i \geq 2$. □

When a word $w = w(A_0, \ldots, A_k)$ is non-empty, we let $\operatorname{rank}(w)$ denote the maximum i such that $A_i^{\pm 1}$ occurs in w and $\eta(w)$ denote the number of $A_1^{-1}, \ldots, A_k^{-1}$ in w. For example, if $w = A_4^{-1} A_3 A_0^{-1} A_1^{-1} A_2$, then $\operatorname{rank}(w) = 4$ and $\eta(w) = 2$.

As we said in Section 1.1, strings of Ackermann functions offer a means of representing integers. 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 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, and so $w(0)$ is a well-defined integer.

For example, $w := A_2^{-1} A_1 A_1 A_0$ is valid, and $w(0) = \log_2(2 \cdot 2 \cdot (0+1)) = 2$. But $A_2 A_0^{-1}$ and $A_1 A_1^{-1} A_0$ 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}).

For $m \in \mathbb{Z}$, the *sign* of m, denoted $\operatorname{sgn}(m)$, is $-$, 0, or $+$ depending on whether $m < 0$, $m = 0$, or $m > 0$, respectively. So Theorem 1 states that there is a polynomial-time algorithm to test validity of $w(A_0, \ldots, A_k)$ and, when valid, to determine the sign of $w(0)$.

We say $w(A_0, \ldots, A_k)$ and $w'(A_0, \ldots, A_k)$ are *equivalent* and write $w \sim w'$ when w and w' are either both invalid, or are both valid and $w(0) = w'(0)$.

2.2. **Examples and general strategy.** We fix an integer $k \geq 0$ throughout the remainder of this article.

We will motivate and outline our design of our algorithm **`Ackermann`** by means of some examples. The details of **`Ackermann`** and it subroutines (which we refer to parenthetically below) follow in Section 2.3.

First consider the case where the word $w(A_0, \ldots, A_k)$ in question satisfies $\eta(w) = 0$—that is, contains no $A_1^{-1}, \ldots, A_k^{-1}$. Such w are not hard to handle because, to check validity of

w, we only need to make sure that no A_i in w with $i \geq 2$ takes a negative input when $w(0)$ is evaluated. (Such w are handled by the subroutine **`Positive`**.) Here is an example.

Example 2.2. Let $w = A_0^{-6}A_1A_0^{-1}A_5A_0^{-4}A_2A_1A_2A_0$, which is a word of length 17 with $\eta(w) = 0$. We can evaluate directly working from right to left that, if valid, $w(0) = A_0^{-6}A_1A_0^{-1}A_5(12)$. At this point we are reluctant to calculate $A_5(12)$ as it is enormous, and instead recognize that $A_5(12)$ is larger than $\ell(w) = 17$ (**`Bounds`**), which as we will explain in a moment we can do suitably quickly. We then deduce that w is valid and $w(0) > 0$, because A_0^{-1} are the only letters further to the left which would lower the value, were the evaluation to continue, and there cannot be enough of them to reach 0 or a negative number.

In general, if $\eta(w) = 0$, our algorithm starts evaluating $w(0)$ working right to left. Let w_j denote the length-j suffix of w. The only letters in w which could decrease absolute value are $A_0^{\pm 1}$, so if $|w_j(0)| > \ell(w)$ for some j and w is valid, then $\text{sgn}(w_j(0)) = \text{sgn}(w(0))$. Moreover, if $\left|w_j(0)\right| > \ell(w)$, then the only way w fails to be valid is if $w_j(0) < 0$ and the prefix of w to the left of w_j contains one of $A_2, A_3, \ldots$. So after either exhausting w or reaching such a j and then scanning the remaining letters in w, the algorithm can halt and decide whether or not $w(0)$ is valid, and if so its sign.

This technique adapts to compare $w(0)$ with a constant –

Example 2.3. Take w as in Example 2.2. We see that $w(0) > 2$ by applying the same technique to find that $w(0) - 2 = A_0^{-2}w(0) > 0$. Here, the size of $A_5(12)$ still dwarfs $\ell(A_0^{-2}w) = 19$, so the computation carried out is essentially the same.

So, how do we determine that $A_5(12) > 17$ or, indeed, $A_5(12) > 19$ for Examples 2.2 and 2.3? The recursion $A_{i+1}(n + 1) = A_iA_{i+1}(n)$ implies that $\text{Img}\, A_i \subseteq \text{Img}\, A_2$ for all $i \geq 2$. Suppose we wish to know whether $A_i(n)$ is less than some constant c. The cases $i = 0, 1$ are easy to handle as $A_0(n) = n + 1$ and $A_1(n) = 2n$ for all n. So are the cases $n = 0, 1, 2$ as $A_i(0) = 1$, $A_i(1) = 2$, and $A_i(2) = 4$ for all i. As for other values of i and n, the recursion allows a subroutine (**`Bounds`**) to list the $i \geq 2$ and $n \geq 3$ for which $A_i(n) < c$.

For instance, to find the $i \geq 2$ and $n \geq 0$ for which $A_i(n) < 17$, first calculate $A_2(n) = 2^n$ for all n for which $A_2(n) < 17$, filling in the first row of the following table.

	$n = 0$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
A_2	1	2	4	8	16
A_3	1	2	4	16	
A_4	1	2	4		

Now fill the table one row at a time. We start with $A_3(0) = 1$ and $A_3(1) = 2$, and then $A_3(2) = A_2A_3(0) = A_2(1) = 2$. Then $A_3(2) = A_2A_3(1)$, which is 4 because, as we already know, $A_3(1) = 2$ and $A_2(2) = 4$. Similarly, $A_3(3) = 16$. And $A_3(4) = A_2A_3(3) = A_2(16)$, which must be greater than 16 since $A_2(16)$ is not in the table. We carry out the same process for A_4. We discover that $A_4(3) = A_3A_4(2) = A_3(4)$ is at least 17 since $A_3(4)$ is not already in the table. At this point we halt, reasoning that $A_j(3) \geq A_i(3) \geq 17$ for all $j > i$ (see Lemma 2.1).

`Ackermann`'s strategy, on input a word w, is to reduce to the case $\eta(w) = 0$ by progressing through a sequence of equivalent words, facilitated by:

Lemma 2.4. *Suppose $u = u(A_0, \ldots, A_k)$ and $v = v(A_0, \ldots, A_k)$. The following equivalences hold if v is invalid or if v is valid and satisfies the further conditions indicated:*

$$\begin{aligned} uA_{i+1}v &\sim uA_iA_{i+1}A_0^{-1}v && v(0) > 0 \text{ and } i \geq 1, \\ uA_{i+1}^{-1}v &\sim uA_0A_{i+1}^{-1}A_i^{-1}v && v(0) > 1 \text{ and } i \geq 1, \\ uA_i^{-1}A_iv &\sim uv && v(0) \geq 0 \text{ and } i \geq 0. \end{aligned}$$

Proof. If v is invalid, then any word with suffix v is invalid, so $uA_{i+1}v \sim uA_iA_{i+1}A_0^{-1}v$ and $uA_{i+1}^{-1}v \sim uA_0A_{i+1}^{-1}A_iv$.

Assume v is valid. If $v(0) > 0$, then $A_0^{-1}v(0) \geq 0$ so that $A_{i+1}v$ and $A_iA_{i+1}A_0^{-1}v$ are valid words and by the recursion defining the functions,

$$A_{i+1}v(0) \;=\; A_iA_{i+1}(v(0)-1) \;=\; A_iA_{i+1}A_0^{-1}v(0).$$

Thus $uA_{i+1}v \sim uA_iA_{i+1}A_0^{-1}v$ since their validity is equivalent to the validity of u on input $A_{i+1}v(0)$.

Suppose $v(0) > 1$. If $v(0) = A_{i+1}(c)$ for some $c \in \mathbb{Z}$, then $c > 0$ because $i \geq 1$, so $v(0) = A_iA_{i+1}(c-1)$. Conversely, $v(0) = A_iA_{i+1}(c-1)$ implies $c \geq 1$. Thus

$$A_0A_{i+1}^{-1}A_i^{-1}v(0) \;=\; c \;=\; A_{i+1}^{-1}v(0),$$

and $uA_0A_{i+1}^{-1}A_i^{-1}v \sim uA_{i+1}^{-1}v$ because their validity is equivalent to validity of u on input $A_{i+1}^{-1}v(0)$.

That $uA_i^{-1}A_iv \sim uv$ under the given assumptions is apparent because the condition $v(0) \geq 0$ ensures $v(0)$ is in the domain of A_i, given that $i \geq 2$. □

We will frequently make tacit use of this fact, which is immediate from the definitions:

Lemma 2.5. *If $w(A_0, \ldots, A_k)$ and $w'(A_0, \ldots, A_k)$ can be expressed as $w = uv$ and $w' = uv'$ for some equivalent suffixes $v \sim v'$, then $w \sim w'$*

Here is an outline of what **`Ackermann`** does on input a valid word w. A description of how **`Ackermann`** checks the hypotheses of Lemma 2.4 and what it does when they fail is postponed until the end of the outline.

1. Locate the rightmost A_r^{-1} in w for which $r \geq 1$. We aim to eliminate this letter, to get a word w' with $\eta(w') < \eta(w)$ and $w \sim w'$ by 'cancelling' it with an A_r that lies somewhere to its right and with no higher rank letters in between. However there may be no such A_r, in which case we manufacture one. Accordingly —
   1.1. If every letter to the right of A_r^{-1} is of rank less than r, then append either $A_0^{-1}A_r$ if $r > 1$ or A_1 if $r = 1$ to create an equivalent word ending in A_r .
   1.2. Locate the first letter $A_{r'}$ that lies to the right of our A_r^{-1} and has $r' \geq r$. If $r' > r$, substitute $A_{r'-1}A_{r'}A_0^{-1}$ for this $A_{r'}$, then $A_{r'-2}A_{r'-1}A_0^{-1}$ for the resulting $A_{r'-1}$, and so on, as per Lemma 2.4 until we have created an A_r (**`Whole`**).
   Thereby, obtain a word equivalent to w which has suffix $s = A_r^{-1}uA_rv$ for some u and v with $\eta(u) = \eta(v) = 0$ and $\mathrm{rank}(u) < r$. (**`Reduce`**.)
2. We now invoke a subroutine (**`Pinch`**$_r$) which will either declare s (and so w) invalid, or will convert s to an equivalent word A_0^lv for some $l \in \mathbb{Z}$.
   Suppose first that $\mathrm{rank}(u) = r - 1 > 0$. We will explain how to eliminate an A_{r-1} from u. On repetition, this will give a word $A_0^mA_r^{-1}\tilde{u}A_rv \sim s$ such that $\mathrm{rank}(\tilde{u}) \leq r-2$. (**`CutRank`**$_r$.)
   2.1. Find the leftmost A_{r-1} in s and write
   $$s \;=\; A_r^{-1}u'A_{r-1}u''A_rv$$
   where $\mathrm{rank}(u') < r-1$ and $\mathrm{rank}(u'') \leq r-1$. Substitute $A_0A_r^{-1}A_{r-1}^{-1}$ for A_r^{-1} as per Lemma 2.4 to give
   $$A_0A_r^{-1}A_{r-1}^{-1}u'A_{r-1}u''A_rv \;\sim\; s.$$
   2.2. Apply **`Pinch`**$_{r-1}$ to the suffix $A_{r-1}^{-1}u'A_{r-1}u''A_rv$ to give an equivalent word $A_0^{l'}u''A_rv$ for some $l' \in \mathbb{Z}$. Thereby get
   $$A_0A_r^{-1}A_0^{l'}u''A_rv \;\sim\; s.$$

2.3. Likewise eliminate an A_{r-1} from u'' in $A_r^{-1}A_0^{l'}u''A_rv$, and so on, until we arrive at

$$A_0^m A_r^{-1}\tilde{u}A_r v \ \sim \ s$$

such that $m \in \mathbb{Z}$ and $\text{rank}(\tilde{u}) \leq r-2$.

To reduce the rank of the subword between the A_r^{-1} and the A_r further we manufacture an A_{r-1}^{-1} and an A_{r-1} and then proceed recursively. Accordingly —

2.4. Substitute for A_r^{-1} and A_r as per Lemma 2.4 to get

$$A_0^m\,(A_0A_r^{-1}A_{r-1}^{-1})\,\tilde{u}\,(A_{r-1}A_rA_0^{-1})\,v \ \sim \ s.$$

2.5. Call **Pinch**$_{r-1}$ on the suffix $A_{r-1}^{-1}\tilde{u}A_{r-1}A_rA_0^{-1}v$ to obtain

$$A_0^{m+1}A_r^{-1}A_0^{l''}A_rA_0^{-1}v \ \sim \ s$$

for some $l'' \in \mathbb{Z}$ (**FinalPinch**$_r$).

3. Eliminate A_r^{-1} and A_r from the suffix $A_r^{-1}A_0^{l''}A_rA_0^{-1}v$ using a method we will shortly explain via Example 2.7 to give an equivalent suffix $A_0^{l'''}A_0^{-1}v$ for some $l''' \in \mathbb{Z}$ (**BasePinch**). Thereby, if w' is the word obtained from w by substituting the suffix beginning with the final A_r^{-1} with $A_0^{m+1}A_0^{l'''}A_0^{-1}v$, then $w \sim w'$ and $\eta(w') < \eta(w)$, as required.
4. Repeat steps 1–3 until we have an equivalent word with no $A_1^{-1}, \ldots, A_k^{-1}$.
5. Use the strategy (**Positive**) from Example 2.2 above.

To make legitimate substitutions as per Lemma 2.4 in Steps 1.2, 2.1, and 2.4, we have to examine certain suffixes. In every instance we are:

1. either substituting $A_iA_{i+1}A_0^{-1}$ for an A_{i+1}, in which case we have to check that the suffix v (which has $\eta(v) = 0$) after that A_{i+1} has $v(0) > 0$,
2. or substituting $A_0A_{i+1}^{-1}A_i^{-1}$ for an A_{i+1}^{-1}, in which case we have to check that the suffix v after that A_{i+1}^{-1} (which again has $\eta(v) = 0$) has $v(0) > 1$.

So validity of v and the hypothesis $v(0) > 0$ or $v(0) > 1$ (and indeed whether $v(0) < 0$, whether $v(0) = 1$, or whether $v(0) \leq 0$, which we will soon also need) can be checked in the manner of Examples 2.2 and 2.3, and if v is invalid, then w is invalid.

Suppose, then, we are in Case i, v is valid, but $v(0) \leq 0$.

- If $i > 0$ and $v(0) < 0$, then $A_{i+1}v$, and so w, is invalid.
- If $i > 1$ and $v(0) = 0$, then $A_{i+1}v(0) = 1$ and so, instead of making the planned substitution, the suffix $A_{i+1}v$ can be replaced by the equivalent A_iv.
- If $i = 1$ and $v(0) = 0$, then we have a suffix A_2v which we replace by the equivalent $A_0A_1(v)$.
- When $i = 0$, no substitution is necessary because $A_1^{-1}uA_1v$ is valid if and only if $u(0)$ is even. If so $u = A_0^l$ for some even l and $A_1^{-1}uA_1v$ can be replaced by the equivalent $A_0^{l/2}v$.

Suppose, on the other hand, that we are in Case ii, v is valid, but v is valid and $v(0) \leq 1$. The algorithm actually only tries to make substitutions for A_{i+1}^{-1} when the input word has suffix $A_{i+1}^{-1}uA_{i+1}v_0$ for some subwords u and v_0 such that $\eta(u) = \eta(v_0) = 0$ and $\text{rank}(u) < i+1$ (and $v \equiv uA_{i+1}v_0$). It proceeds as follows:

- If $v(0) = 1$ and $i > 0$, output the equivalent $A_0^{-v_0(0)}v_0$.
- If $i = 0$ use the fact that $A_1^{-1}uA_1v_0$ is valid if and only if $u(0)$ is even. If $u(0)$ is even, $u = A_0^l$ for some even integer l replace the suffix $A_1^{-1}uA_1v_0$ by the equivalent $A_0^{l/2}v_0$.
- If $v(0) \leq 0$, then $A_{i+1}^{-1}v$ is invalid.

(In Case ii, it is not obvious that outputting $A_0^{-v_0(0)}v_0$ is better than simply returning the empty word to represent zero. However, the inductive construction of the algorithm requires that the output word retain a suffix v_0.)

Example 2.6. Let $w = A_0A_2^{-1}A_1A_0^2A_2A_0$. A quick direct calculation shows w is valid and $w(0) = 4$, but here is how our **Ackermann** handles it.

1. First aim to eliminate the A_2^{-1} (the subroutine **Reduce**). Look to the right of the A_2^{-1} for the first subsequent letter (if any) of rank at least 2, namely the A_2.
2. Try to 'cancel' the A_2^{-1} with the A_2 (**Pinch**$_2$) —
   2.1. Reduce the rank of the subword $A_1A_0^2$ between A_2^{-1} and A_2 as follows (**CutRank**$_2$).
      2.1.1. Use the technique of Example 2.2 (**Positive**) to check that the suffix $A_1A_0^2A_2A_0$ is valid and $A_1A_0^2A_2A_0(0) > 1$. So, by by Lemma 2.4, we can legitimately substitute $A_0^2A_2^{-1}A_1^{-1}$ for A_2^{-1} to obtain
$$A_0A_2^{-1}A_1^{-1}A_1A_0^2A_2A_0 \ \sim\ w.$$
      2.1.2. Cancel the $A_1^{-1}A_1$ (strictly speaking, this is done by calling **CutRank**$_2$ on $A_2^{-1}A_1^{-1}A_1A_0^2A_2A_0$, and then **Pinch**$_2$) to give
$$A_0^2A_2^{-1}A_0^2A_2A_0 \ \sim\ w.$$
   2.2. Next follow Step 2.4 from the outline above. Seek to replace the subword $A_2^{-1}A_0^2A_2$ by an appropriate power of A_0 (by calling **FinalPinch**$_2$ on the suffix $s := A_2^{-1}A_0^2A_2A_0$) as follows.
      2.2.1. Check $A_0(0) \neq 0$ and $A_0^2A_2A_0(0) \neq 1$, so we can substitute $A_0A_2^{-1}A_1^{-1}$ for A_2^{-1} and $A_1A_2A_0^{-1}$ for A_2 in s (as per Lemma 2.4) to get
$$A_0A_2^{-1}A_1^{-1}A_0^2A_1A_2A_0^{-1}A_0 \ \sim\ s.$$
      2.2.2. Convert the subword $A_1^{-1}A_0^2A_1$ to a power of A_0 (by calling **Pinch**$_1$ on $A_1^{-1}A_0^2A_1A_2A_0^{-1}A_0$, which calls **BasePinch**$_1$ since the subword between the A_1^{-1} and the A_1 is a power of A_0). It replaces $A_1^{-1}A_0^2A_1$ by A_0 (which is appropriate because $(2x+2)/2 = x+1$) to give
$$s' \ :=\ A_0A_2^{-1}A_0A_2A_0^{-1}A_0 \ \sim\ s.$$
      2.2.3. The exponent sum of the A_0 between A_2^{-1} and A_2 in s' is 1. (Were it non-zero and less than half of $A_2A_0^{-1}A_0(0) = 1$, then $A_2A_0^{-1}A_0(0)$ would be too far from another integer in the image of $A_2(n)$ for s' to be valid.) But, in this case, we evaluate $A_2^{-1}A_0A_2A_0^{-1}A_0(0)$ by computing that it is 2 directly from right to left, and then evaluating $A_2^{-1}(2) = 1$ (by calling **Bounds**$(2\ell(w))$). So $A_2^{-1}A_0A_2A_0^{-1}A_0(0) = 1$, and we can conclude that
$$s' \ \sim\ A_0^2A_0^{-1}A_0.$$
      (Preserving the suffix $A_0^{-1}A_0$ appears unnecessary here, but it reflects the recursive design of the algorithm.)

   So
$$w' \ :=\ A_0^4A_0^{-1}A_0 \ \sim\ w.$$
3. Now $\eta(w') = 0$. So evaluate w' from right-to-left in the manner of Example 2.2 (**Positive**) and declare that w is valid and $w(0) > 0$.

In our next example, the input word has the form $A_r^{-1}uA_{r'}v$ with $\eta(u) = \eta(v) = 0$ and $\text{rank}(u) < r < r'$. As there is no A_r with which we can 'cancel' the A_r^{-1}, we manufacture one by using Lemma 2.4 to create an A_r to the left of the $A_{r'}$ and thereby reduce to a situation similar to the preceding example. This example also serves to explain how we resolve the special case $A_r^{-1}A_0^lA_rv$ which is crucial for avoiding explicit computation of large numbers.

Example 2.7. Set $w = A_2^{-1}A_0^{-2}A_3A_0^{100}$.

1. Identify the rightmost A_i^{-1} with $i \geq 1$, namely the A_2^{-1}. Scanning to the right of A_2^{-1}, the first A_i we encounter with $i \geq 2$ is the A_3. (Send w to **Reduce**, which calls **Whole**.)
2. Use techniques from Example 2.2 (**Positive**) to check that $A_0^{100}(0) > 0$. So we can substitute $A_2A_3A_0^{-1}$ for A_3, as per Lemma 2.4, to obtain
$$w_0 \ := \ A_2^{-1}A_0^{-2}A_2A_3A_0^{-1}A_0^{100} \ \sim \ w.$$
3. We check we can make substitutions as in Lemma 2.4 for A_2^{-1} and A_2 to give
$$w_1 \ := \ (A_0A_2^{-1}A_1^{-1})\,A_0^{-2}\,(A_1A_2A_0^{-1})\,A_3A_0^{-1}A_0^{100} \ \sim \ w.$$
(Run **CutRank**$_2$ on w_0 which does nothing as $\mathrm{rank}(u) < 1$, and then start running **FinalPinch**$_2(w_0)$.)
4. We now want to reduce the rank of the subword between the A_2^{-1} and A_2 to zero (**Pinch**$_2$), and so we (**BasePinch**$_1$) process the suffix
$$A_1^{-1}A_0^{-2}A_1A_2A_0^{-1}A_3A_0^{-1}A_0^{100}$$
to replace $A_1^{-1}A_0^{-2}A_1$ by A_0^{-1} giving
$$w_2 \ := \ A_0A_2^{-1}A_0^{-1}A_2A_0^{-1}A_3A_0^{-1}A_0^{100} \ \sim \ w$$
(the equivalence being because $(2x-2)/2 = x-1$).
5. Now the subword of w_2 between A_2^{-1} and A_2 has rank 0 (which causes **Pinch**$_2$ to end and we return to **FinalPinch**$_2$, which in turn invokes **BasePinch**$_2$). As A_2 is the function $\mathbb{N} \to \mathbb{N}$ mapping $n \mapsto 2^n$, if $A_0^zA_2A_0^{-1}A_3A_0^{-1}A_0^{100}(0)$ is in the domain of A_2^{-1} for some $z \in \mathbb{Z} \smallsetminus \{0\}$, then the large gaps between powers of 2 ensure that $2|z| \geq A_2A_0^{-1}A_3A_0^{-1}A_0^{100}(0)$. In the case of w_2, we have $z = -1$ and so we see that w_2 is invalid by checking that $A_2A_0^{-1}A_3A_0^{-1}A_0^{100}(0) > 2$. We can do this efficiently in the manner of Example 2.3 by noting that $A_3A_0^{-1}A_0^{100}(0)$ exceeds the threshold $\ell(A_2A_0^{-1}A_3A_0^{-1}A_0^{100}) + 2 = 106$. So we declare w invalid.

A major reason **Ackermann** halts in polynomial time, is that as it manipulates words, it does not substantially increase their lengths. One subroutine it employs, **Bounds**, takes an integer as its input. All others input a word w and output an equivalent word w' and in every case but two, $\ell(w') \leq \ell(w)$. The exceptions are the subroutines **Whole** and **Reduce**, where $\ell(w') \leq \ell(w) + 2k$. But they are each called at most $\eta(w) \leq \ell(w)$ times when **Ackermann** is run on input w, so they do not cause length to blow up. The way this control on length is achieved is that while length is increased by making substitutions as per Lemma 2.4, those increases are offset by a process of replacing a suffix of the form $A_r^{-1}uA_rv$ (with $\eta(u) = \eta(v) = 0$ and $\mathrm{rank}(u) < r$) by an equivalent suffix of the form A_0^lv with $|l| \leq \ell(u)$.

The technique of exploiting the large gaps between powers of 2 to sidestep direct calculation applies to all words of the form $A_r^{-1}A_0^zA_rv$ where $r \geq 2$ and $z \neq 0$, after all the gaps in the range of A_r grow even faster when $r > 2$. In Lemma 2.1 (8), we showed that if $l \in \mathbb{Z}$ is non-zero and $A_r^{-1}A_0^lA_rv$ is valid, then $2|l| \geq A_rv(0)$. This condition can be efficiently checked if $\eta(v) = 0$. If $2|l| \geq A_rv(0)$, direct computation of the value of $A_r^{-1}A_0^lA_rv(0)$ (using **Bounds**$(2|l|)$) becomes efficient relative to $\ell(w)$ since $|l| \leq \ell(w)$.

Our final example is a circumstance where we are unable to make substitutions because a hypothesis of Lemma 2.4 fails.

Example 2.8. Let $w = A_3^{-1}A_0^{-1}A_3A_0$. Direct calculation shows that w is valid and $w(0) = 0$, but here is how our algorithm proceeds.

1. As before, we identify the A_3^{-1}, the subsequent A_3, and the subword A_0^{-1} that separates them. (Call **Pinch**$_3$ on $A_3^{-1}uA_3v$ where $u = A_0^{-1}$ and $v = A_0$.)
2. First we check that A_0 is valid and $A_0(0) \geq 0$ and so is in the domain of A_3. Then we check that $A_0^{-1}A_3A_0$ is valid (a necessary condition for validity of w) and $A_0^{-1}A_3A_0(0) \geq 0$ (a necessary condition to be in the domain of A_3^{-1}). (In both cases we use **Positive**.)
3. We notice that there are no $A_1^{\pm 1}$ or $A_2^{\pm 1}$ between A_3^{-1} and A_3 to remove. (**Pinch**$_3$ runs **CutRank**$_3(w)$, which does not change w.)
4. We seek to substitute $A_0A_3^{-1}A_2^{-1}$ for A_3^{-1} and $A_2A_3A_0^{-1}$ for A_3. (**Pinch**$_3$ calls **FinalPinch**$_3$.) But, by calculating that $A_0^{-1}A_0^{-1}A_3A_0(0) = 0$ (which is done by calling **Positive**$(A_0^{-1}A_0^{-1}A_3A_0)$), we discover that $A_0^{-1}A_3A_0(0) = 1$, violating a hypothesis of Lemma 2.4.
5. Invoke a subroutine (**OneToZero**) for this special case. We calculate the integer $m = v(0)$ by testing whether $A_0^{-m}v(0) = 0$ starting with $m = 1$ and incrementing m by 1 until we obtain a string equal to zero. In this example $v = A_0$, and so $m = 1$. We return $A_0^{-m}v = A_0^{-1}A_0$ where $A_0^{-m}v(0) = 0 = A_r^{-1}(1) = A_r^{-1}v(0)$. It would be simpler to return the empty word, but the recursive structure of **Pinch** requires the output of an equivalent word whose suffix is v.
6. $\eta(A_0^{-1}A_0) = 0$, so the algorithm explicitly affirms validity, finds the sign of $A_0^{-1}A_0(0)$, and returns 0. (**Positive**.)

2.3. **Our algorithm.** We continue to have an integer $k \geq 0$ fixed and work with words on the alphabet $A_0^{\pm 1}, \ldots, A_k^{\pm 1}$. The polynomial time bounds we establish in this section all depend on k.

Our first subroutine follows the procedure explained in Section 2.2, so we only sketch it here.

Algorithm 2.1 — Bounds.
- Input $\ell \in \mathbb{N}$ (expressed in binary).
- Return 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 $A_r(n) \leq \ell$.
- Halt in time $O(\ell)$.

---

list all values of $A_2(n) = 2^n$ for which $2 \leq n \leq \lfloor \log_2 \ell \rfloor$
recall (from Lemma 2.1) that $A_i(2) = 4$ for all $i \geq 2$
use the recursion $A_{i+1}(n+1) = A_iA_{i+1}(n)$ to calculate all $A_r(n) \leq \ell$ for $r \geq 3$ and $n \geq 3$, halting when $A_r(3) > \ell$

Correctness of **Bounds**. **Bounds** generates its list of triples by first listing the at most $\lfloor \log_2(\ell) \rfloor$ triples $(2, n, A_2(n))$ such that $n \geq 3$ and $A_2(n) = 2^n \leq \ell$, which it can do in time $O((\log_2 \ell)^2)$ since ℓ is expressed in binary. It then reads through this list and uses the recurrence relation (and the fact that $A_3(2) = 4$) to list all the $(3, n, A_3(n))$ for which $n \geq 3$ and $A_3(n) \leq \ell$. It then uses those to list the $(4, n, A_4(n))$ similarly, and so on. For all $r \geq 3$, $A_r(3) = A_{r-1}(4) \geq 2A_{r-1}(3)$, and so $A_r(3) \geq 2^r$. So the triples $(r, n, A_r(n))$ outputted by **Bounds** all have $r \leq \lfloor \log_2 \ell \rfloor$. As r increases, there are fewer n such that $A_r(n) \leq \ell$. So the complete list **Bounds** outputs comprises at most $(\log_2 \ell)^2$ triples of binary numbers each recorded by a binary string of length at most $\log_2 \ell$, and it is generated in time $O(\ell)$. □

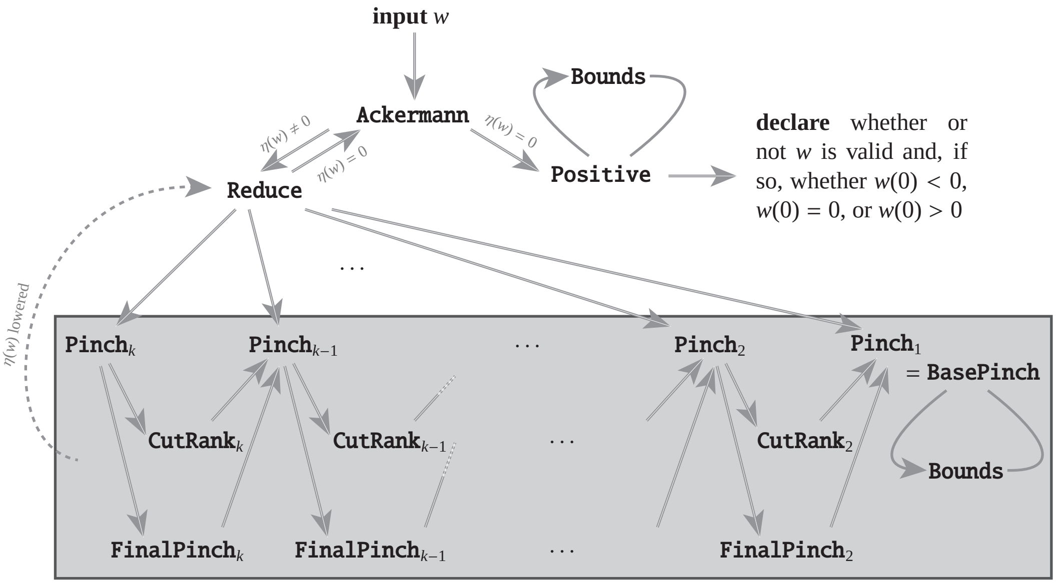

FIGURE 1. An outline of the design of **Ackermann**, indicating which routines call which other routines. Any routine may declare w invalid and halt the algorithm. From **Reduce**, the algorithm progresses to **Pinch**$_r$, where r is the subscript of the rightmost of $A_1^{-1}, \ldots, A_k^{-1}$ to remain in w. The progression through the **Pinch**$_i$, **CutRank**$_i$, and **FinalPinch**$_i$ (shown boxed) is involved (and not apparent from the diagram) but ultimately decreases $\eta(w)$ by one. A further routine **OneToZero** (which handles certain special cases) does not appear, but is called by a number of the routines shown. **Positive** also serves as a routine, but only its role in providing the final step in the algorithm is indicated in the figure.

(In fact, **Bounds** halts in time polynomial in $\log_2 \ell$, but we are content with the $O(\ell)$ bound because other terms will dominate our cost-analyses of the routines that call **Bounds**.)

Remark 2.9. **Bounds** does not give any $(r, n, A_r(n))$ for which $A_r(n) \geq \ell$ but $r \leq 1$ or $n \leq 2$. Nevertheless, such triples require negligible computation to identify. After all, $A_r(0) = 1$, $A_r(1) = 2$ and $A_r(2) = 4$ for all $r \geq 1$ and $A_0(n) = n + 1$ and $A_1(n) = 2n$ for all $n \in \mathbb{Z}$.

Correctness of **Positive**. As w is a word on $A_0^{\pm 1}, A_1, \ldots, A_k$ (that is, $\eta(w) = 0$), decreases in absolute value only occur in increments of 1 as $w(0)$ is evaluated from right to left. The domains of A_0, A_0^{-1} and A_1 are $\mathbb{Z}$, and of $A_2, A_3, \ldots$ are $\mathbb{N}$, so w is invalid only when some A_i with $i \geq 1$ meets a negative input. If the threshold, $+n$, is exceeded, then w must be valid and $w(0) > 0$, as subsequent letter-by-letter evaluation could never reach a negative value. If $x_i \ldots x_1(0) < -n$ for some i (which is easily tested as it can only first happen when x_i is A_0^{-1} or A_1), then w is valid if and only if none of the subsequent letters are $A_2, \ldots, A_k$; moreover, if w is valid, then $w(0) < 0$. If w is exhausted, then the algorithm has fully calculated $w(0)$ (and $|w(0)| < n$) and has confirmed w as valid.

Positive calls **Bounds** once with input $n = \ell(w)$, which produces its list of at most $(\log_2 n)^2$ triples in time $O(n)$. The thresholds employed in **Positive** ensure that it performs arithmetic operations (adding one, doubling, comparing absolute values) with integers of absolute value at most n. Each such operation takes time $O(n^2)$, so they and the necessary searches of the output of **Bounds** take time $O(n^3)$. □

Algorithm 2.2 — Positive.

- Input a word $w = x_n x_{n-1} \cdots x_1$ where $x_1, \ldots, x_n \in \left\{A_0^{\pm 1}, A_1, \ldots, A_k\right\}$.
- Return **invalid** when w is invalid and $\mathrm{sgn}(w(0))$ when w is valid.
- Halt in time $O(\ell(w)^3)$.

---

run $\texttt{Bounds}(n)$
evaluate $x_1(0)$, then $x_2 x_1(0)$, and so on until
- either $w(0)$ has been evaluated
- or some $x_i \ldots x_1(0) > n$ (checked by consulting the output of $\texttt{Bounds}(n)$)
- or some $x_i \ldots x_1(0) < -n$ (that is, $x_i \neq A_0^{\pm 1}$ and $x_i \ldots x_1(0) < 0$)
- or some $x_i \ldots x_1$ is found to be invalid (that is, $x_i \neq A_0^{\pm 1}$ and $x_i \ldots x_1(0) < 0$)

then, respectively, **return**
- $\mathbf{sgn}(w(0))$
- $\mathbf{sgn}(w(0)) = +$
- if $x_{i+1}, \ldots, x_n \notin \{A_2, \ldots, A_k\}$, then $\mathbf{sgn}(w(0)) = -$, else **invalid**
- **invalid**

Our next subroutine is the $\mathrm{rank}(u) = 0$ case of $\texttt{Pinch}_r$, to come.

Algorithm 2.3 — BasePinch.

- Input a word $w = A_r^{-1} u A_r v$ with $r \geq 1$, $u = u(A_0)$, $v = v(A_0, \ldots, A_k)$ and $\eta(v) = 0$.
- Either return that w is invalid, or return a valid word $w' = A_0^{l'} v \sim w$ such that $\ell(w') \leq \ell(w) - 2$.
- Halt in time $O(\ell(w)^4)$.

---

set $l := u(0)$ (so A_0^l is u with all $A_0^{\pm 1} A_0^{\mp 1}$ subwords removed and $A_r^{-1} A_0^l A_r v \sim w$)
if $\texttt{Positive}(A_r v) =$ **invalid**, **halt** and **return invalid**
3: **if** $r \geq 2$ and $v(0) < 0$ (checked using $\texttt{Positive}$), **halt** and **return invalid**
if $l = 0$, **halt** and **return** $w' := v$
if $r = 1$, **halt** and **return** $w' := A_0^{l/2} v$ or **invalid** depending on whether l is even or odd
6:
we now have $l \neq 0$ and $r > 1$
run $\texttt{Positive}(A_0^l A_r v)$ to determine if $A_0^l A_r v(0) \leq 0$ (so outside the domain of A_r^{-1})
9: **if** so, **halt** and **return invalid**
run $\texttt{Positive}(A_0^{-2|l|} A_r v)$ to determine whether $A_r v(0) > 2|l|$
if so, **halt** and **return**
12:
we now have that $0 \leq v(0) \leq |l|$ and $0 < A_r v(0) \leq 2|l|$ and $A_r v(0) + l \leq 3|l|$
calculate $v(0)$ by running $\texttt{Positive}(A_0^{-i} v)$ for $i = 0, 1, \ldots, |l|$
15: **run** $\texttt{Bounds}(3\,|l|)$
search the output of $\texttt{Bounds}(3\,|l|)$ to find $A_r v(0)$
set $m := A_r v(0) + l$
18: search the output of $\texttt{Bounds}(3\,|l|)$ for c with $A_r(c) = m$ (so $c = A_r^{-1} A_0^l A_r v(0) = w(0)$)
if such a c exists, **halt** and **return** $w' := A_0^{c - v(0)} v$
else halt and **return invalid**

Correctness of **BasePinch**. The idea is that when w is valid, either $l = 0$ or the sparseness of the image of A_r implies that l is large enough that $w(0)$ can be calculated efficiently. Here is why the algorithm runs as claimed.

3: If $v(0) < 0$, then w is invalid.
4: If $r \geq 2$, then $A_r^{-1} A_r v \sim v$ by Lemma 2.4.

5: 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 w.

8, 10: 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.

11: Let $q = v(0)$. For all $p \neq q$ we have $|A_r(q) - A_r(p)| \geq \frac{1}{2} A_r(m)$ by Lemma 2.1 (8), 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.

13 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 10 and 11. 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 $A_0^l A_r v(0) \leq l + 2\,|l|$.

18: 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 13, we know $m \leq 3\,|l|$, so this will find c if it exists. If no such c exists, w is invalid.

19: $A_0^{c - v(0)} v(0) = c = A_r^{-1}(l + A_r v(0)) = A_r^{-1} A_0^l A_r v(0)$.

We must show that $\ell(w') \leq \ell(w) - 2$. In the cases of lines 4 and 5, this is immediate, so suppose $r \geq 2$. As for line 19, we will show that $|c - v(0)| \leq |l|$, from which the result will immediately follow.

First suppose $l \geq 0$. By Lemma 2.1 and the fact that $v(0) \geq 0$, we have $A_r(v(0) + l) \geq A_r(v(0)) + l$. So $v(0) + l \geq A_r^{-1}(A_r v(0) + l) = c$. So $c - v(0) \leq l = |l|$. And $0 \leq c - v(0)$ because $A_r(c) = A_r(v(0)) + l \geq A_r(v(0))$. So $|c - v(0)| \leq |l|$, as required.

Suppose, on the other hand, $l < 0$. Then

$$c \;=\; A_r^{-1} A_o^l A_r v(0) \;\leq\; A_r^{-1} A_r v(0) \;=\; v(0)$$

and so $|c - v(0)| = v(0) - c$. But then $|c - v(0)| \leq v(0)$ because $v(0), c \geq 0$. So if $v(0) + l \leq 0$, then $|c - v(0)| \leq -l = |l|$, as required. Suppose instead that $v(0) + l > 0$. We have that $A_r(v(0) + l) \leq A_r(v(0)) + l$ because $A_r(p - m) \leq A_r(p) - m$ by Lemma 2.1 (7) for all $p \geq m \geq 0$. So $v(0) + l \leq A_r^{-1}(A_r(v(0)) + l) = c$. So $l \leq c - v(0)$. And $c - v(0) < 0$ because $A_r(c) = A_r v(0) + l < A_r v(0)$. So $|c - v(0)| \leq |l|$, again as required.

Next we explain why the integer calculations performed by the algorithm involve integers of absolute value at most $3\ell(w)$. The algorithm calls **Positive** on words of length at most $3\ell(w)$, and so (by the properties of **Positive** established), each time it is called, **Positive** calculates with integers no larger than $3\ell(w)$. On input $3\,|l| \leq 3\ell(w)$, **Bounds** calculates with integers of absolute value at most $3\ell(w)$. The only remaining integer manipulations concern $m, l, 2\,|l|\,, A_r v(0)$, all of which have absolute value at most $3\ell(w)$.

Finally, that **BasePinch** halts in time $O(\ell(w)^4)$ is straightforward given the previously established cubic and linear halting times for **Positive** and **Bounds**, respectively, and the following facts. It may add a pair of positive binary numbers each at most $2\ell(w)$, may determine the parity of a number of absolute value at most $\ell(w)$, and may halve an even positive number less than $\ell(w)$. It calls **Positive** at most $|l| + 3 \leq \ell(w) + 3$ times, each time on input a word of length at most $2\ell(w)$. It calls **Bounds** at most once—in that event the input to **Bounds** is a non-negative integer that is at most $3\ell(w)$ and the output of **Bounds** is searched at most twice and has size $O((\log_2 \ell(w))^2)$. □

Algorithm 2.4 — **`OneToZero`**.
- Input a valid word $w = A_r^{-1} u A_r v$ with $\eta(u) = \eta(v) = 0$, $u \neq \epsilon$, $uA_r v(0) = 1$ and $r \geq 2$.
- Return a word $A_0^{-v(0)} v \sim w$ of length at most $\ell(w) - 2$.
- Halt in time $O(\ell(w)^4)$.

run **`Positive`**$(A_0^{-m} v)$ for $m = 0, 1, \ldots$ **until** it declares that $A_0^{-m} v = 0$
halt and **output** $A_0^{-m} v$

Correctness of **`OneToZero`**.

1: As w is valid, $v(0)$ is in the domain of A_r, which is $\mathbb{N}$ as $r \geq 2$. So $m = v(0)$ will eventually be found.
2: $w(0) = A_r^{-1}(1) = 0$ and so $A_0^{-m} v \sim w$ as required, since $A_0^{-m} v(0) = 0$.

Since $\eta(u) = 0$, the only letter u may contain which decreases the value in the course of evaluating $uA_r v(0)$ is A_0^{-1}. So, as $uA_r v(0) = 1$ and $A_r v(0) \geq v(0) + 1$, there must be at least $v(0)$ letters A_0^{-1} in u. So $\ell(u) \geq v(0)$. So $\ell(A_0^{-v(0)} v) \leq \ell(w) - 2$, as required.

`OneToZero` calls **`Positive`** $m = v(0) \leq \ell(u) \leq \ell(w)$ times, each time on input of length at most $\ell(w)$. So, by the established properties of **`Positive`**, it halts in time $O(\ell(w)^4)$. □

The input w to **`OneToZero`** necessarily has $w(0) = 0$, so it would seem it should just output the empty word rather than $A_0^{-v(0)} v$. However, **`OneToZero`** is used by **`Pinch`**$_r$, which we will describe next and whose inductive construction requires the suffix v.

`Pinch`$_r$ for $r \geq 1$ is a family of subroutines which we will construct alongside further families **`CutRank`**$_r$ and **`FinalPinch`**$_r$ for $r \geq 2$. **`Pinch`**$_{r-1}$ is a subroutine of **`CutRank`**$_r$ and of **`FinalPinch`**$_r$. **`CutRank`**$_r$ and **`FinalPinch`**$_r$ are subroutines of **`Pinch`**$_r$. It may appear that we could discard **`CutRank`**$_r$ and use **`FinalPinch`**$_r$ instead, by expanding **`FinalPinch`**$_r$ to allow inputs with $\text{rank}(u) = r - 1$ and expanding **`Pinch`**$_r$ to allow inputs where $\text{rank}(u) = r$. But this would cause problems with maintaining the suffix v.

Algorithm 2.5 — **`Pinch`**$_r$ for $r \geq 1$.
- Input a word $w = A_r^{-1} u A_r v$ with $\eta(u) = \eta(v) = 0$ and $\text{rank}(u) \leq r - 1$.
- Either return that w is invalid, or return a valid word $w' = A_0^{l'} v \sim w$ such that $\ell(w') \leq \ell(w) - 2$.
- Halt in $O(\ell(w)^{4+(r-1)})$ time.

if $r = 1$ **run** **`BasePinch`**(w) and then **halt**
run **`Positive`**(v) to determine whether v is invalid or $v(0) < 0$
  if so **halt** and **return invalid**
run **`Positive`**$(uA_r v)$ to determine whether $uA_r v$ is valid or $uA_r v(0) \leq 0$
  if so **halt** and **return invalid**
run **`CutRank`**$_r(w)$
  it either declares w invalid, in which case **halt** and **return invalid**
  or it returns a word $w' = A_0^i A_r^{-1} u' A_r v$ such that
  $w' \sim w$, $\ell(w') \leq \ell(w)$, $\eta(u') = 0$, $u' \neq \epsilon$ and $\text{rank}(u') < r - 1$
run **`FinalPinch`**$_r(A_r^{-1} u' A_r v)$
  if it declares $A_r^{-1} u' A_r v$ invalid, **halt** and **return invalid**
  else it outputs $A_0^l v$ for some l, in which case **set** $w'' := A_0^{i+l} v$
run **`Positive`**(w'')
  if it declares w'' invalid, **halt** and **return invalid**
  else return w''

Algorithm 2.6 — **CutRank**$_r$ for $r \geq 2$.
- Input a word $w = A_r^{-1} u A_r v$ with $\eta(u) = \eta(v) = 0$ and $\text{rank}(u) \leq r-1$.
- Either declare w invalid, or return $w' = A_0^l v$ where $\ell(w') \leq \ell(w) - 2$, or return $w' = A_0^i A_r^{-1} u' A_r v \sim w$ where $\text{rank}(u') \leq r-2$, $\eta(u') = 0$, and $\ell(w') \leq \ell(w)$.
- Halt in time $O(\ell(w)^{4+(r-1)})$.

set $i = 0$ and re-express w as $A_0^i A_r^{-1} u A_r v$
if $v(0) < 0$ (checked using **Positive**), **halt** and **return invalid**
3: **if** u is the empty word, **halt** and **return** v
while $\text{rank}(u) = r-1$ **do**
    run Positive$(A_0^{-1} u A_r v)$ to test whether $uA_r v(0) = 1$
6:        **if** so **halt** and **return** the output $w' = A_0^l v$ of **OneToZero**(w)
    run Positive$(uA_r v)$ to test whether $uA_r v(0) \leq 0$
        if so, **halt** and **return invalid**
9:    **express** u as $u' A_{r-1} u''$ where $\text{rank}(u') < r-1$ (i.e. locate the leftmost A_{r-1} in u)
    increment i by 1
    set $w := A_0^i A_r^{-1} A_{r-1}^{-1} u' A_{r-1} u'' A_r v$ (i.e. substitute $A_0 A_r^{-1} A_{r-1}$ for A_r^{-1} in w)
12:    **run Pinch**$_{r-1}(A_{r-1}^{-1} u' A_{r-1} u'' A_r v)$
        if it returns invalid **halt**, **return invalid**
        else let $w_0 := A_0^s u'' A_r v$ be the (valid) word returned
15:        **set** $w := A_0^i A_r^{-1} w_0$
        set $u := A_0^s u''$ so that $w = A_0^i A_r^{-1} u A_r v$
end while
18: **return** w

Correctness of **Pinch**$_{r-1}$ *implies the correctness of* **CutRank**$_r$ *for all* $r \geq 2$. The idea of **CutRank**$_r$ is that each pass around the while loop eliminates one A_{r-1} from u. So in the output, $\text{rank}(u) < r-1$.

2: If $r \geq 2$, then the domain of A_r is $\mathbb{N}$, and so w is invalid when $v(0) < 0$.
3: Since $v(0) \geq 0$ now, Lemma 2.4 applies.
6: $\ell(w') \leq \ell(w) - 2$ by the specifications of **OneToZero**.
8: If $uA_r v(0) \leq 0$, it is outside the domain of A_r^{-1} (as $r \geq 2$), so the algorithm's input is invalid.
11: Substituting gives an equivalent word here by Lemma 2.4, since $uA_r v(0) \geq 1$. At this point, $\ell(w)$ is at most 2 more than its initial length.
16: Now w is no longer than it was at the start of the **while** loop because **Pinch**$_{r-1}$ (assuming it does not halt) trims at least 2 letters, offsetting the gain at line 11. The word w here at the end of the **while** loop is equivalent to the w at the start because of our remark on line 11 and because we are replacing a suffix $A_{r-1}^{-1} u' A_{r-1} u'' A_r v$ by an equivalent word produced by **Pinch**$_{r-1}$.
18: It follows from our remarks on lines 11 and 16 that $\ell(w)$ here is at most the length of the w originally inputted.

The while loop is traversed at most $\ell(w)$ times. Each time, **Positive** (twice), **OneToZero** and **Pinch**$_{r-1}$ may be called, and by the remarks above, their inputs are always of length at most $\ell(w)$. So, as each of these subroutines halt in time $O(\ell(w)^{4+(r-2)})$, **CutRank**$_r$ halts in $O(\ell(w)^{4+(r-1)})$ time. □

Correctness of **Pinch**$_{r-1}$ *implies correctness of* **FinalPinch**$_r$ *for* $r \geq 2$.

2: If $uA_r v(0) < 1$, then it is outside the domain of A_r^{-1}.
4: $uA_r v$ is valid if and only if $A_0^{-1} u A_r v$ is valid.
8: In this case $v(0)$ is outside the domain of A_r.

Algorithm 2.7 — **`FinalPinch`**$_r$ for $r \geq 2$.
- Input a word $w = A_r^{-1} u A_r v$ with $\eta(u) = \eta(v) = 0$, $u \neq \epsilon$ and $\mathrm{rank}(u) < r-1$.
- Either declare w invalid or return a word $A_0^l v \sim w$ of length at most $\ell(w) - 2$.
- Halt in $O(\ell(w)^{4+(r-2)})$ time.

run `Positive`$(A_0^{-1} u A_r v) = 0$ to decide among the following cases
    if $A_0^{-1} u A_r v$ is invalid or $u A_r v(0) < 1$, **halt** and **return invalid**
3: **if** $u A_r v(0) = 1$, **halt** and **return `OneToZero`**$_r(w)$
we now have that $u A_r v$ is valid and $u A_r v(0) > 1$

6: **run `Positive`**(v) to determine whether $v(0) < 0$, $v(0) = 0$, or $v(0) > 0$

if $v(0) < 0$, **halt** and **return invalid**
9:
if $v(0) = 0$
    if $r = 2$, **run `BasePinch`**$(A_r^{-1} u A_r v)$
12:         **if** it returns **invalid**, **halt** and do likewise
        else halt and return its result $A_0^{l'} v$, which will satisfy $\ell(A_0^{l'} v) \leq \ell(w) - 2$
    if $r > 2$, **run `Pinch`**$_{r-1}(A_{r-1}^{-1} u A_{r-1} v)$
15:         **if** it returns **invalid**, **halt** and do likewise
        else it returns $A_0^l v$ for some $|l| \leq \ell(u)$
    if $l \leq 0$, **halt** and **return invalid**
18:     **run `BasePinch`**$(A_r^{-1} A_0^{l-1} A_r v)$
        if it returns **invalid**, **halt** and do likewise
        else it returns $A_0^{l'} v$ for some $|l'| \leq |l-1| = l-1$,
21:             in which case **halt** and **return** $A_0^{l'+1} v$

if $v(0) > 0$
24:     **run `Pinch`**$_{r-1}(A_{r-1}^{-1} u A_{r-1} A_r A_0^{-1} v)$
        if it returns **invalid**, **halt** and do likewise
        else it returns $A_0^l A_r A_0^{-1} v$ for some $|l| \leq \ell(u)$
27:     **run `BasePinch`**$(A_r^{-1} A_0^l A_r A_0^{-1} v)$
        if it returns **invalid**, **halt** and do likewise
        else it returns $A_0^{l''} A_0^{-1} v$ for some $|l''| \leq |l|$,
30:             in which case **halt** and **return** $A_0^{l''} v$

11: If $r = 2$, the rank of u is zero, so **`BasePinch`** applies.
13: $\ell(A_0^{l'} v) \leq \ell(w) - 2$ by properties of **`BasePinch`**.
16: $w \sim A_0 A_r^{-1} A_{r-1}^{-1} u A_{r-1} v$ when $r > 2$ and $v(0) = 0$, because $A_r v \sim A_{r-1} v$ and we can substitute $A_0 A_r^{-1} A_{r-1}^{-1}$ for A_r^{-1} as per Lemma 2.4, given that $u A_r v(0) > 1$. So if $A_{r-1}^{-1} u A_{r-1} v$ is invalid, then so is w. And if **`Pinch`**$_{r-1}$ gives us that $A_{r-1}^{-1} u A_{r-1} v \sim A_0^l v$, then $w \sim A_0 A_r^{-1} A_0^l v$.
17: If $l \leq 0$, then w is invalid because $A_0^l v(0) \leq 0$ and lies outside of the domain of A_r^{-1} (since $r \geq 2$).
19: Next, working from $w \sim A_0 A_r^{-1} A_0^l v$ established in our comment above on line 16, we get that $w \sim A_0 A_r^{-1} A_0^{l-1} A_r v$ because $A_0^{-1} A_r v \sim v$, given that $r \geq 2$ and $v(0) = 0$. So, if **`BasePinch`** tells us that $A_r^{-1} A_0^{l-1} A_r v$ is invalid, then so is w.
20: $|l - 1| = l - 1$ here because $l > 0$ here.
21: Similarly, if $A_r^{-1} A_0^{l-1} A_r v \sim A_0^{l'} v$, then $w \sim A_0^{l'+1} v$. Now, $|l' + 1| \leq |l'| + 1 \leq l$ by line 20, and $l \leq \ell(u)$ in the case $r > 2$ of line 16. So $\ell(A_0^{l'+1} v) \leq \ell(w) - 2$, as required.

23: $w \sim A_0 A_r^{-1} A_{r-1}^{-1} u A_{r-1} A_r A_0^{-1} v$ because Lemma 2.4 tells us that substituting $A_{r-1} A_r A_0^{-1}$ for A_r and $A_0 A_r^{-1} A_{r-1}^{-1}$ for A_r^{-1} in w gives an equivalent word as $v(0) > 0$ and $uA_{r-1}v(0) > 1$. This word is longer than w by 2.
25: So, if the suffix $A_{r-1}^{-1} u A_{r-1} A_r A_0^{-1} v$ is invalid, then so is w.
26: Similarly, if the suffix $A_{r-1}^{-1} u A_{r-1} A_r A_0^{-1} v \sim A_0^{l} A_r A_0^{-1} v$, then $w \sim A_0 A_r^{-1} A_0^{l} A_r A_0^{-1} v$.
28: If the suffix $A_r^{-1} A_0^{l} A_r A_0^{-1} v$ is invalid, then so is w.
30: If the suffix $A_r^{-1} A_0^{l} A_r A_0^{-1} v \sim A_0^{l''} A_0^{-1} v$, then $w \sim A_0 A_0^{l''} A_0^{-1} v \sim A_0^{l''} v$ and has length at most $\ell(w) - 2$ since $|l''| \leq |l|$ and (from line 26) $|l| \leq \ell(u)$ (or to put it another way, we have taken $A_0 A_r^{-1} A_{r-1}^{-1} u A_0 A_1 v$ (see the comment on line 23) which is four letters longer than w, and $\mathtt{Pinch}_{r-1}$ and $\mathtt{BasePinch}$ have each shortened it by two).

$\mathtt{FinalPinch}_r$ halts in $O(\ell(w)^{4+(r-2)})$ time because it makes at most four calls on subroutines ($\mathtt{Positive}$, $\mathtt{OneToZero}$, $\mathtt{Pinch}_{r-1}$ or $\mathtt{BasePinch}$) and, each time, the subroutine has input of length at most $\ell(w) + 2$ and halts in $O(\ell(w)^{4+(r-2)})$ time. □

Correctness of $\mathtt{CutRank}_r$ *and* $\mathtt{FinalPinch}_r$ *implies correctness of* $\mathtt{Pinch}_r$ *for* $r \geq 2$.

3: If v is invalid, then so is w. If $v(0) < 0$, then $v(0)$ is outside the domain of A_r (as $r \geq 2$) and so w is invalid.
5: If $uA_r v$ is invalid, then so is w. If $uA_r v(0) \leq 0$, then $v(0)$ is outside the domain of A_r^{-1} (as $r \geq 2$) and so w is invalid.
10: $\ell(A_r^{-1} u' A_r v) \leq \ell(w') \leq \ell(w)$, the second inequality being by an established property of $\mathtt{CutRank}_r$.
11: If the suffix $A_r^{-1} u' A_r v$ of w' is invalid, then so is w', and hence so is w.
12: $w'' \sim w$ because it is obtained by replacing the suffix $A_r^{-1} u' A_r v$ of w' by an equivalent word.
13: $\eta(w'') = 0$, so we can use $\mathtt{Positive}$ to determine validity of w''. Also, $\ell(w'') \leq i + \ell(A_0^{l} v) \leq i + \ell(A_r^{-1} u' A_r v) - 2 = \ell(w') - 2 < \ell(w)$, the second and final inequalities follow from established properties of $\mathtt{FinalPinch}_r$ and $\mathtt{CutRank}_r$, respectively.

That $\mathtt{Pinch}_r$ runs in $O(\ell(w)^{4+(k-1)})$ time follows directly from the time bounds for the subroutines $\mathtt{Positive}$, $\mathtt{CutRank}_r$, $\mathtt{BasePinch}$ and $\mathtt{FinalPinch}_r$ as it calls these at most six times in total and on each occasion, the input has length at most $\ell(w)$—see the comments above on lines 10 and 13. □

Correctness of $\mathtt{Pinch}_r$ *for* $r \geq 1$ *and of* $\mathtt{CutRank}_r$ *and* $\mathtt{FinalPinch}_r$ *for* $r \geq 2$. For $r = 1$, the correctness of $\mathtt{Pinch}_1$ follows from that of $\mathtt{BasePinch}$. As explained above, for $r \geq 2$, the correctness of $\mathtt{CutRank}_r$ and $\mathtt{FinalPinch}_r$ implies that of $\mathtt{Pinch}_r$, and the correctness of $\mathtt{Pinch}_{r-1}$ implies that of $\mathtt{CutRank}_r$ and $\mathtt{FinalPinch}_r$. So, by induction on r, $\mathtt{Pinch}_r$ is correct for all $r \geq 1$. □

Correctness of $\mathtt{Reduce}$. The idea is to eliminate the rightmost A_r^{-1} with $1 \leq r \leq k$ from w by either using $\mathtt{Pinch}_r$ directly on a suffix of w or by manipulating w into an equivalent word with a suffix that can be input into $\mathtt{Pinch}_r$.

4: $A_0^{-1} A_r(0) = 0$ (since $r \geq 2$), so $w_2 A_0^{-1} A_r \sim w_2$.
6: $A_0^{l} \sim A_r^{-1} w_2 A_0^{-1} A_r$ and so $w' \sim w$. Evidently, $\eta(w') = \eta(w) - 1$. And $\ell(w') = \ell(w_1) + |l| \leq \ell(w_1) + \ell(w_2) + 1 = \ell(w) \leq \ell(w) + 2k$, as required.
8: $A_1(0) = 0$, so $w_2 A_1 \sim w_2$.
10: $A_0^{l} \sim A_1^{-1} w_2 A_1 \sim A_1^{-1} w_2$ and so $w' \sim w$, as required. Also, evidently, $\eta(w') = \eta(w) - 1$, and $\ell(w') \leq \ell(w) + 2k$, as required.
13: Moreover, $\eta(w_3) = \eta(w_4) = 0$ because $\eta(w_2) = 0$, as will be required in line 15.
15: The length of w'' is at most $\ell(w) - \ell(w_1) - 2$ by properties of $\mathtt{Pinch}_r$.

Algorithm 2.8 — **Reduce**.
∘ Input a word w with $\eta(w) > 0$.
∘ Either return that w is invalid, or return a word $w' \sim w$ with $\ell(w') \leq \ell(w) + 2k$ and $\eta(w') = \eta(w) - 1$.
∘ Halt in $O(\ell(w)^{4+(k-1)})$ time.

---

express w as $w_1 A_r^{-1} w_2$ where $r \geq 1$ and $\eta(w_2) = 0$
(i.e. locate rightmost $A_1^{-1}, A_2^{-1}, \ldots, A_k^{-1}$ in w)
3:
if $\mathrm{rank}(w_2) < r$ and $r \geq 2$, **run Pinch**$_r(A_r^{-1} w_2 A_0^{-1} A_r)$
  if it declares $A_r^{-1} w_2 A_0^{-1} A_r$ invalid, **halt** and **return invalid**
6: **else** it returns A_0^l for some $|l| \leq \ell(w_2) + 1$, in which case **return** $w' := w_1 A_0^l$

if $\mathrm{rank}(w_2) = 0$ and $r = 1$, **run Pinch**$_1(A_1^{-1} w_2 A_1)$
9: **if** it declares $A_1^{-1} w_2 A_1$ invalid, **halt** and **return invalid**
  else it returns A_0^l for some $|l| \leq \ell(w_2)$, in which case **return** $w' := w_1 A_0^l$

12: **if** $\mathrm{rank}(w_2) \geq r$
  express w_2 as $w_3 A_s w_4$ where $r \leq s$ and $\mathrm{rank}(w_3) < r$
  run Positive(w_4) to decide among the following cases
15:   if $r = s = 1$, set $w'' =$ **Pinch**$_1(A_r^{-1} w_3 A_s w_4)$
    else if w_4 is invalid or $v(0) < 0$, **halt** and **return invalid**
    else if $w_4(0) = 0$, $r = 1$ and $s > r$, set $w'' =$ **Pinch**$_r(A_r^{-1} u A_0 A_r v)$
18:   **else if** $w_4(0) = 0$ and $r > 1$, set $w'' =$ **Pinch**$_r(A_r^{-1} w_3 A_r w_4)$
    else $w_4(0) > 0$, so set $w'' =$ **Pinch**$_r(A_r^{-1} w_3 A_r\ A_{r+1} A_0^{-1}\ A_{r+2} A_0^{-1}\ \cdots A_s A_0^{-1}\ w_4)$

  if $w'' =$ **invalid**, **halt** and **return invalid**
21: **else return** $w' := w_1 w''$

16: If $w_3(0) < 0$, then w is invalid because $s \geq 2$
17: In this case $A_r^{-1} w_3 A_0 A_r w_4 \sim A_r^{-1} w_3 A_s w_4$ since $A_0 A_r(0) = A_s(0)$. As required, if $w'' \neq$ **invalid**, it has length at most $\ell(A_r^{-1} u A_0 A_r v) = \ell(w) - \ell(w_1) + 1 < \ell(w) - \ell(w_1) + 2k$ and contains no $A_1^{-1}, \ldots, A_k^{-1}$ by the properties established for **Pinch**$_r$.
18: Similarly, in this case $A_r^{-1} w_3 A_r w_4 \sim A_r^{-1} w_3 A_s w_4$ since $A_r(0) = A_s(0)$, and the output has the required properties.
19: If $w_4(0) > 0$, then $A_r^{-1} w_3 A_s w_4$ and $A_r^{-1} w_3 A_{s-1} A_s A_0^{-1} w_4$ are equivalent by Lemma 2.4. As $v(0) - 1 \geq 0$, and so is in the domain of A_s, the word $A_s A_0^{-1} v$ is valid. And, as $A_s A_0^{-1} v(0) = A_s(v(0) - 1) > 0$, we may replace the A_{s-1} by $A_{s-2} A_{s-1} A_0^{-1}$ to get another equivalent word. Indeed, we may repeat this process $s - r \leq k$ times, to yield an equivalent word

$$A_r^{-1} w_3 A_r\ A_{r+1} A_0^{-1}\ A_{r+2} A_0^{-1}\ \cdots A_s A_0^{-1}\ w_4$$

of length $\ell(w) - \ell(w_1) + 2(s - r)$. Applying **Pinch**$_r$ then returns (if valid) an equivalent word

$$w'' \ = \ A_0^l\ A_{r+1} A_0^{-1}\ A_{r+2} A_0^{-1}\ \cdots A_s A_0^{-1}\ w_4$$

whose length is at most $\ell(w) - \ell(w_1) + 2(s - r) - 2$.
20: If the suffix $A_r^{-1} w_3 A_s w_4$ of w is invalid, then w is invalid.
21: By the above $\ell(w'') \leq \ell(w) - \ell(w_1) + 2(s - r)$, we have that $w'' \sim A_r^{-1} w_3 A_s w_4$, $\eta(w'') = 0$ and $\ell(w'') \leq \ell(A_r^{-1} w_3 A_s w_4) + 2r = 1 + \ell(w_2) + 2r$. It follows that $w \sim w_1 w''$ and $\ell(w_1 w'') = \ell(w_1) + \ell(w'') \leq \ell(w_1) + 1 + \ell(w_2) + 2r \leq \ell(w) + 2k$, as required. Also, again evidently, $\eta(w') = \eta(w) - 1$.

Reduce halts in $O(\ell(w)^{4+(k-1)})$ time since **Pinch**$_r$ and **Positive** do and they are each called at most once and only on words of length at most $\ell(w) + 2k$, and otherwise **Reduce** scans w and compares non-negative integers that are at most k. □

Proof of Theorem 1. Here is our algorithm **Ackermann** satisfying the requirements of Theorem 1: it declares, in polynomial time in $\ell(w)$, whether or not a word $w(A_0, \ldots, A_k)$ is valid, and if so, it gives $\mathrm{sgn}(w)$.

Algorithm 2.9 — Ackermann.
- Input a word w.
- Return whether w is valid and if it is, return $\mathrm{sgn}(w(0))$.
- Halt in $O(\ell(w)^{4+k})$ time.

---

if $\eta(w) > 0$, **run Reduce** successively until
  it either returns that w is **invalid**,
  or it returns some $w' \sim w$ with $\eta(w') = 0$
run Positive(w')

After at most $\eta(w) \leq \ell(w)$ iterations of **Reduce**, we have a word w' with $\eta(w') = 0$ such that $w'(0) = w(0)$. We then apply **Positive** to w' to obtain the result.

The correctness of **Ackermann** is immediate from the correctness of **Reduce** and **Positive**.

Reduce is called at most $\ell(w)$ times as it decreases $\eta(w)$ by one each time. Each time it is run, it adds at most $2k$ to the length of the word. So the lengths of the words inputted into **Reduce** or **Positive** are at most $\ell(w) + 2k\ell(w)$. So, as **Reduce** and **Positive** run in $O(\ell(w)^{4+(k-1)})$ time in the lengths of their inputs, **Ackermann** halts in $O(\ell(w)^{4+k})$ time. □

3. Efficient calculation with ψ-compressed integers

3.1. ψ-functions and ψ-words. Similarly to Ackermann functions in Section 2.1, we define *ψ-functions* by

$$\begin{aligned}
&\psi_1 : \mathbb{Z} \to \mathbb{Z} && n \mapsto n - 1 && \\
&\psi_2 : \mathbb{Z} \to \mathbb{Z} && n \mapsto 2n - 1 && \\
&\psi_i : -\mathbb{N} \to -\mathbb{N} && && \forall i \geq 3 \\
&\qquad \psi_i(0) \ := \ -1 && && \forall i \geq 1 \\
&\qquad \psi_{i+1}(n) \ := \ \psi_i \psi_{i+1}(n+1) - 1 && && \forall n \in -\mathbb{N}, \forall i \geq 2.
\end{aligned}$$

Having entered the $i = 1$ row and $n = 0$ column as per the definition, a table of values of $\psi_i(n)$ can be completed by determining each row from right-to-left from the preceding one using the recurrence relation:

$\cdots$	n	$\cdots$	-4	-3	-2	-1	0	
$\cdots$	$n-1$	$\cdots$	-5	-4	-3	-2	-1	ψ_1
$\cdots$	$2n-1$	$\cdots$	-9	-7	-5	-3	-1	ψ_2
$\cdots$	$2 - 3 \cdot 2^{-n}$	$\cdots$	-46	-22	-10	-4	-1	ψ_3
	$\vdots$		$\vdots$	$1 - 3 \cdot 2^{95}$	-95	-5	-1	ψ_4
				$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
						$-i-1$	-1	ψ_i
						$\vdots$	$\vdots$	$\vdots$

The following proposition explains why we defined ψ-functions with the given domains. It details the key property of ψ-functions, which is that they govern whether and how a power of t *pushes past* an a_i on its right, to leave an element of H_k times a new power of t without changing the element of G_k represented.

Proposition 3.1. *Suppose r, i and k are integers such that $1 \leq i \leq k$. Then $t^r a_i \in H_k t^s$ in G_k if and only if r is in the domain of ψ_i and $s = \psi_i(r)$.*

Proof. First we prove the 'if' direction by inducting on pairs (i, r), ordered lexicographically. We start with the cases $i = 1$ and $i = 2$. As $a_1 t \in H_k$ and $t^{-1} a_1 t = a_1$,

$$t^r a_1 \;=\; a_1 t\, t^{r-1} \in H_k t^{r-1} \;=\; H_k t^{\psi_1(r)}$$

for all $r \in \mathbb{Z}$. And as, $a_2 t \in H_k$ and $t^{-1} a_2 t = a_2 a_1$ also,

$$t^r a_2 \;=\; t^r a_2 t^{-r} t^r \;=\; a_2 a_1^{-r} t^r \;=\; a_2 t\,(a_1 t)^{-r}\; t^{2r-1}$$

for all $r \in \mathbb{Z}$. Next the case where $r = 0$ and $1 \leq i \leq k$:

$$t^r a_i \;=\; a_i \;=\; a_i t\, t^{-1} \;\in\; H_k t^{-1} \;=\; H_k t^{\psi_i(0)},$$

since $a_i t \in H_k$ and $\psi_i(0) = -1$. Finally, induction gives us that

$$t^r a_i \;=\; t^{r+1} a_i a_{i-1} t^{-1} \;\in\; H_k t^{\psi_i(r+1)} a_{i-1} t^{-1} \;=\; H_k t^{\psi_{i-1}\psi_i(r+1)-1} \;=\; H_k t^{\psi_i(r)}$$

for all $i \geq 2$ and $r \leq 0$, as required.

For the 'only if' direction suppose $t^r a_i \in H_k t^s$ for some $s \in \mathbb{Z}$. Then

$$t^r a_i t^{-r} \;=\; \theta^{-r}(a_i) \;\in\; H_k t^{s-r}$$

for some $s \in \mathbb{Z}$. Lemma 7.3 in [12] tells us that in the cases $i = 1, 2$ this occurs when $r \in \mathbb{Z}$, and in the cases $i \geq 3$ it occurs when $r \in -\mathbb{N}$. In other words, it occurs when r is in the domain of ψ_i. Now, given that r is in the domain of ψ_i, we have that $t^r a_i \in H_k t^{\psi_i(r)}$ from the calculations earlier in our proof, and so $H_k t^{\psi_i(r)} = H_k t^s$, but this implies that $s = \psi_i(r)$ by Lemma 6.1 in [12]. □

For example, painful calculation can show that

$$t^{-2} a_3 a_1 \;=\; (a_3 t)(a_2 t)(a_1 t)(a_2 t)(a_1 t)^5 t^{-11} \in H_3 t^{-11},$$

but Proposition 3.1 immediately gives:

$$t^{-2} a_3 a_1 \in H_3 t^{\psi_1 \psi_3(-2)} \;=\; H_3 t^{-11}.$$

The following criterion for whether and how a power of t pushes past an a_i^{-1} on its right, to leave an element of H_k times a new power of t can be derived from Proposition 3.1.

Corollary 3.2. *Suppose i and k are integers such that $1 \leq i \leq k$. Then $t^s a_i^{-1} \in H_k t^r$ in G_k if and only if r is in the domain of ψ_i and $s = \psi_i(r)$.*

Proof. $t^s a_i^{-1} \in H_k t^r$ if and only if $t^r a_i \in H_k t^s$. □

The connection between ψ-functions and hydra groups is also apparent in that they relate to the functions ϕ_i of [12] by the identity $\psi_i(n) = n - \phi_i(-n)$ for all $n \in -\mathbb{N}$ and all $i \geq 1$. We will not use this fact here, so we omit a proof, except to say that the recurrence $\phi_{i+1}(n) = \phi_{i+1}(n-1) + \phi_i(\phi_{i+1}(n-1) + n - 1)$ for all $i \geq 1$ and $n \geq 1$ of Lemma 3.1 in [12] translates to the defining recurrence of ψ-functions.

Lemma 3.3.

$$\psi_2(n) = 2n-1 \qquad \forall n \le 0, \tag{9}$$
$$\psi_3(n) = 2-3\cdot 2^{-n} \qquad \forall n \le 0, \tag{10}$$
$$\psi_i(-1) = -i-1 \qquad \forall i \ge 1, \tag{11}$$
$$\psi_i(n) \ge \psi_{i+1}(n) \qquad \forall i \ge 1, n \le 0, \tag{12}$$
$$\psi_i(n) > \psi_i(n-1) \qquad \forall i \ge 1, n \le 0, \tag{13}$$
$$n > \psi_i(n) \qquad \forall i \ge 1, n \le 0, \tag{14}$$
$$\psi_i(m)+\psi_i(n) \ge \psi_i(m+n) \qquad \forall n,m \le -2,\ i \ge 2, \tag{15}$$
$$|\psi_i(m)-\psi_i(n)| \ge \frac{1}{2}|\psi_i(n)| \qquad \forall i \ge 3, m \ne n. \tag{16}$$

Proof. (9–15) are evident from the manner in which the table of values of $\psi_i(n)$ above is constructed. Formal induction proofs could be given as for Lemma 2.1.

For (16), when $m > n$ (so that $|n| > |m|$),

$$\begin{aligned} |\psi_3(m)-\psi_3(n)| &= |3\cdot 2^{-m}-3\cdot 2^{-n}| \ge |3\cdot 2^{-n}-3\cdot 2^{-n-1}| = \frac{1}{2}\cdot 3\cdot 2^{-n} \\ &\ge \frac{1}{2}\cdot 3\cdot 2^{-n}-1 = \frac{1}{2}(3\cdot 2^{-n}-2) = \frac{1}{2}|\psi_3(n)|, \end{aligned}$$

and when $m < n$ (so that $|n| < |m|$), by the preceding

$$|\psi_3(m)-\psi_3(n)| = |\psi_3(n)-\psi_3(m)| \ge \frac{1}{2}|\psi_3(m)| \ge \frac{1}{2}|\psi_3(n)|,$$

using (13) for the last inequality. So the result holds for $i = 3$. That it also holds for all $i > 3$ then follows. We omit the details. □

By (13), ψ-functions are injective and so have inverses ψ_i^{-1} defined on the images of ψ_i:

$$\begin{array}{lll} \psi_1^{-1} & : \mathbb{Z} \to \mathbb{Z} & n \mapsto n+1, \\ \psi_2^{-1} & : 2\mathbb{Z}+1 \to \mathbb{Z} & n \mapsto (n+1)/2, \\ \psi_i^{-1} & : \mathrm{Img}\psi_i \to -\mathbb{N} & n \mapsto \psi_i^{-1}(n). \end{array}$$

So, like Ackermann functions, they can specify integers. A *ψ-word* is a word $f = f_n f_{n-1}\cdots f_1$ where each $f_i \in \{\psi_1^{\pm 1}, \psi_2^{\pm 1}, \ldots\}$. We let

$$\eta(f) := \#\{i \mid 1 \le i \le n,\ f_i = \psi_j^{-1} \text{ for some } j \ge 2\}.$$

If $f_{j-1}\cdots f_1(0)$ is in the domain of f_j for all $2 \le j \le n$, then f is *valid* and *represents* the integer $f(0)$. When f is non-empty, rank(f) denotes the highest i such that $\psi_i^{\pm 1}$ is a letter of f. We define an equivalence relation $\sim$ on words as in Section 2.1.

Proposition 3.1 and Corollary 3.2 combine to tell us, for example, that:

$$t^{-3}a_2^{-1}a_1 \in H_2 t^{\psi_1\psi_2^{-1}(-3)}$$

if $-3 \in \mathrm{Img}\psi_2$ and $\psi_2^{-1}(-3)$ is in the domain of ψ_1—in other words, if $\psi_1\psi_2^{-1}\psi_1^3$ is valid. In fact these provisos are met: $\psi_2^{-1}(-3) = -1$ and $\psi_1(-1) = -2$, so $t^{-3}a_2^{-1}a_1 \in H_2t^2$. And, given that $H_kt^r = H_k$ if and only if $r = 0$ by Lemma 6.1 in [12], determining whether $t^{-3}a_2^{-1}a_1 \in H_2$ amounts to determining whether $\psi_1\psi_2^{-1}\psi_1^3(0) = 0$. (In fact it equals 2, as we just saw, so $t^{-3}a_2^{-1}a_1 \notin H_2$.) This suggests that efficiently testing validity of ψ-words and when valid, determining whether a ψ-word represents zero, will be a step towards a polynomial time algorithm solving the membership problem for H_k in G_k. (Had $\psi_1\psi_2^{-1}\psi_1^3$ been invalid, we could not have immediately concluded that that $t^{-3}a_2^{-1}a_1 \notin H_2$ or indeed that $t^{-3}a_2^{-1}a_1 \notin \bigcup_{r\in\mathbb{Z}} H_2t^r$. We will address this delicate issue in Section 4.4.) So we will work towards proving this analogue to Theorem 1:

Proposition 3.4. *There exists an algorithm* **Psi** *that takes as input a ψ-word $f = f(\psi_1, \ldots, \psi_k)$ and determines in time $O(\ell(f)^{4+k})$ whether or not f is valid and if so, whether $f(0)$ is positive, negative or zero.*

Expressing the recursion relation in terms of ψ-words will be key. So, analogously to Lemma 2.4, we have:

Lemma 3.5. *Suppose u, v are ψ-words. The following equivalences hold if v is invalid or if v is valid and satisfies the further conditions indicated:*

$$\begin{array}{rcll} u\psi_{i+1}v & \sim & u\psi_1\psi_i\psi_{i+1}\psi_1^{-1}v & v(0) < 0 \text{ and } i \geq 2 \\ u\psi_{i+1}^{-1}v & \sim & u\psi_1\psi_{i+1}^{-1}\psi_i^{-1}\psi_1^{-1}v & v(0) < -1 \text{ and } i \geq 1 \\ u\psi_i^{-1}\psi_i v & \sim & uv & v(0) \geq 0 \text{ and } i \geq 1. \end{array}$$

3.2. An example.

Let

$$f = \psi_3^{-1}\psi_2^{-1}\psi_1^2\psi_2^2\psi_3(\psi_2\psi_3)^2\psi_1\psi_1^{-1}.$$

Here is how **Psi** checks its validity and determines the sign of $f(0)$.

1. First we locate the rightmost ψ_i^{-1} in f with $i \geq 2$, namely the ψ_2^{-1}, and look to 'cancel' it with the first ψ_2 to its right. In short, this is possible because
$$((2x-1)-2-1)/2 = x-1,$$
allowing us to replace $\psi_2^{-1}\psi_1^2\psi_2$ with ψ_1 to give
$$\psi_3^{-1}\psi_1\psi_2\psi_3(\psi_2\psi_3)^2\psi_1\psi_1^{-1} \sim f.$$
2. Next we identify the new rightmost ψ_i^{-1} with $i \geq 2$, namely the ψ_3^{-1} and we look to 'cancel' it with the ψ_3 to its right. To this end we first reduce the rank of the subword between the ψ_3^{-1} and ψ_3 (like **CutRank**). We check by direct calculation that
$$\psi_1\psi_2\psi_3(\psi_2\psi_3)^2\psi_1\psi_1^{-1}(0) < -1$$
(like **Positive**), so the substitution $\psi_1\psi_3^{-1}\psi_2^{-1}\psi_1^{-1}$ for ψ_3^{-1} is legitimate by Lemma 3.5 and
$$\psi_1\psi_3^{-1}\psi_2^{-1}\psi_1^{-1}\psi_1\psi_2\psi_3(\psi_2\psi_3)^2\psi_1\psi_1^{-1} \sim f.$$
By Lemma 3.5, cancelation of the ψ_1^{-1} with ψ_1, ψ_2^{-1} with ψ_2, and then ψ_3^{-1} with ψ_3 then gives
$$\psi_1(\psi_2\psi_3)^2\psi_1\psi_1^{-1} \sim f.$$
3. This contains no $\psi_2^{-1}, \ldots, \psi_k^{-1}$ and direct evaluation from right to left (like **Positive**) tells us that $\psi_1(\psi_2\psi_3)^2\psi_1\psi_1^{-1}$ is valid and represents a negative integer.

3.3. Our algorithm in detail.

Fix an integer $k \geq 1$.

Subroutines of **Psi** correspond to subroutines of **Ackermann**. We first have an analogue of **Bounds**, to calculate relatively small evaluations of the ψ_k.

Algorithm 3.1 — BoundsII.
◦ Input $\ell \in \mathbb{N}$.
◦ Return a list of all the (at most $(\log_2 \ell)^2$) triples of integers $(r, n, \psi_r(n))$ such that $r \geq 3$, $n \leq -2$, and $|\psi_r(n)| \leq \ell$.
◦ Halt in time $O(\ell)$.

With these minor changes, it works exactly like **Bounds**: replace A_i by ψ_{i+1}, calculate values of $\psi_r(n)$ for $n \leq -2$, and use the recursive relation for ψ-functions. The correctness argument for **BoundsII** is virtually identical to that for **Bounds**.

Similarly to **Ackermann**, **Psi** works right-to-left through a ψ-word eliminating letters ψ_r^{-1} for $r \geq 2$, which like (the A_r^{-1} for $r \geq 1$) greatly decrease absolute value when evaluating the integer represented by a valid ψ-word. Once all have been eliminated, giving a ψ-word f with $\eta(f) = 0$, a subroutine **PositiveII** determines the validity of f.

Algorithm 3.2 — PositiveII.
◦ Input a ψ-word f with $\eta(f) = 0$.
◦ Either return that f is **invalid**, or that f is valid and declare whether $f(0) > 0$, $f(0) = 0$, or $f(0) < 0$.
◦ Halt in time $O(\ell(f)^3)$.

PositiveII can be constructed analogously to **Positive** with the following changes:

1. The role of ψ_i corresponds to the role of A_{i-1}.
2. Unlike Ackermann functions, $\psi_i : -\mathbb{N} \to -\mathbb{N}$, so appropriate signs and inequalities need to be altered.
3. We still evaluate letter-by-letter. However, in place of using **Bounds** to check whether an evaluation by A_i is above some (positive) threshold, we use **BoundsII** to check that ψ_k evaluated on a negative number is below some (negative) threshold.
4. Similarly, the case where a partial letter-by-letter evaluation is negative should be replaced by a case where the partial letter-by-letter evaluation is positive.

Then **PositiveII** can be justified similarly to **Positive**.

Next **BasePinchII** processes words of the form $\psi_k^{-1}\psi_1^l\psi_k v$. We make one major change: we have a stricter bound that **BasePinch** on the length of the returned word f'. The substitution suggested by Lemma 3.5 requires a substitution of 4 letters for 1 rather than the 3 for 1 substitution suggested by Lemma 2.4 for the Ackermann case. Here and in **PinchII**, stricter bounds on the length of the output compensate for the longer substitution and thus prevent the length of words processed by recursive calls to **PinchII** from growing too large.

Algorithm 3.3 — BasePinchII.
◦ Input a word $f = \psi_r^{-1}u\psi_r v$ with $k \geq 1$, $\mathrm{rank}(u) \leq 1$, v a ψ-word, and $\eta(v) = 0$.
◦ Either return **invalid** when f is invalid or return a word $f' = \psi_1^{l'}v \sim f$ such that $\ell(f') \leq \ell(f) - 2$ if u is empty, $\ell(f') \leq \ell(f) - 4$ if $r > 2$, and otherwise, $\ell(f') \leq \ell(f) - 3$.
◦ Halt in time $O(\ell(f)^4)$.

Construct **BasePinchII** like **BasePinch** with the following changes:

1. Replace all called subroutines by their ψ-versions.
2. ψ_{i+1} replaces A_i for all $i \geq 0$.
3. Signs and inequalities are adjusted to reflect that $\psi_{i+1} : -\mathbb{N} \to -\mathbb{N}$ and that $\psi_1(n) = n - 1$ (in contrast to $A_0(n) = n + 1$).
4. For the case $r = 2$, whenever $\psi_2 v(0)$ is valid, it is odd (since $\psi_2(n) = 2n - 1$) and hence the parity of l determines the parity of $u\psi_2 v(0)$. For validity, we need $u\psi_1 v(0)$ to be odd, and this is sufficient since $\psi_2^{-1}(n) = (n + 1)/2$. When l is even,

return the equivalent word $f' := \psi_1^{l/2}v$. Otherwise f is invalid. The restrictions on the length of l follow directly from the fact that $|l/2| \leq |l| - 1$ if $l = 0$. Henceforth, assume that $r \geq 3$.

5. The inequality
$$|\psi_r(m) - \psi_r(p)| \;\geq\; \frac{1}{2}|\psi_r(m)|$$
which holds for all $r \geq 3$ and $m \neq p$ takes the place of the analogous inequality for Ackermann functions:
$$|A_r(p) - A_r(n)| \;\geq\; \frac{1}{2}A_r(n)$$
which holds for all $r \geq 2$ and $m \neq p$. Following similar arguments for **`BasePinch`**, we instead need $0 \geq \psi_r v(0) \geq -2|l|$ to account for the fact that the ψ_i are functions $-\mathbb{N} \to -\mathbb{N}$.
6. If the algorithm outputs $f' \sim f$ with $f'(0) = c \in \mathbb{Z}$, then $f' = \psi_1^{v(0)-c}v$.

Correctness of **`BasePinchII`**. The argument is essentially the same as that for **`BasePinch`** except that we need to verify the stronger assertions on $\ell(f')$. If $l = 0$, the algorithm eliminates ψ_r^{-1} and ψ_r, reducing length by 2.

For the case $l \neq 0$, consider the following: we claim that
$$|\psi_r(n) - \psi_r(n-1)| \;\geq\; |\psi_3(0) - \psi_3(-1)| \;=\; 3.$$

Explicitly, for $r = 3$, we have:
$$|\psi_r(n) - \psi_r(n-1)| \;=\; 3 \cdot 2^{-n} - 3 \cdot 2^{-(n-1)} \;=\; 3 \cdot 2^{-n} \geq 3 \cdot 2^0 \;=\; 3$$
because $n \leq 0$. For $r > 3$, assume the result holds for all ranks less than r. We have:
$$\begin{aligned}|\psi_r(n) - \psi_r(n-1)| \;&=\; |\psi_{r-1}(\psi_r(n)) - \psi_{r-1}\psi_r(n-1)| \\ &\geq\; |\psi_{r-1}\psi_r(n) - \psi_{r-1}(\psi_r(n) - 1)| \;\geq\; |\psi_3(0) - \psi_3(-1)|\end{aligned}$$
where the final two inequalities follow from the fact that ψ_{r-1} is non-decreasing and the inductive hypothesis, respectively.

By extending this argument inductively and using that ψ_r is non-decreasing:
$$|\psi_r(n) - \psi_r(n+m)| \;\geq\; 3m.$$

So, for $r > 3$ and $l \neq 0$ where $f' = \psi_0^{c-v(0)}v(0)$, we have that $\psi_r(c) - \psi_r(v(0)) = l$ implies that $|c - v(0)| \leq \frac{1}{3}|l|$. In particular, if $l \neq 0$, then $|l| \geq 3$. Therefore,
$$\ell(f') \;=\; |c - v(0)| + \ell(v) \;\leq\; \frac{1}{3}|l| + \ell(v) \;\leq\; |l| - 2 + \ell(v) \;=\; \ell(f) - 4$$
since $|l| - 2 \geq \frac{1}{3}|l|$ if $|l| \geq 3$. Thus we have verified the assertions concerning $\ell(f')$. □

`OneToZeroII` is essentially the same as **`OneToZero`** with A_0 replaced by ψ_1.

Algorithm 3.4 — `OneToZeroII`.
- ◦ Input a valid word word of the form $f = \psi_r^{-1}u\psi_r v$ with $r \geq 3$, u not the empty word, and $\eta(u) = \eta(v) = 0$ such that $u\psi_r v(0) = -1$.
- ◦ Return an equivalent word of the form $f' = \psi_1^{v(0)}v$ with $\ell(f') \leq \ell(f) - 3$.
- ◦ Halt in time $O(\ell(f)^4)$.

Proof that $\ell(f') \leq \ell(f) - 3$ in **OneToZeroII**. Now $v(0) \leq 0$ since $v(0)$ is in the domain of ψ_r and $r \geq 3$. Consider first the case $v(0) \leq -1$. First observe that $\psi_r(x) \leq x - 3$ when $x \leq -1$ and $r \geq 3$. Since $\eta(u) = 0$, ψ_1^{-1} is the only letter it can contain which decreases the absolute value as $f(0)$ is evaluated. So, given that $u\psi_r v(0) = -1$, u must contain ψ_1^{-1} at least $|v(0) - 3| - 1 = |v(0)| + 2$ times. So $\ell(u) \geq |v(0)| + 2$ and therefore

$$\ell(f) - \ell(f') \ = \ 2 + \ell(u) - |v(0)| \ \geq \ 4,$$

and so $\ell(f') < \ell(f) - 3$ as required.

If $v(0) = 0$, **OneToZeroII** returns $f' = v$. Since u is not the empty word, $\ell(f') \leq \ell(f) - 3$ as required. □

PinchII$_r$ is an analogue to **Pinch**$_r$. As in the previous situation, the proof is by induction and uses **BasePinchII** as its base case. As in **BasePinchII**, there are now stronger restrictions on the length of a returned equivalent word.

Algorithm 3.5 — PinchII$_r$ for $r \geq 2$.
◦ Input a word $f = \psi_r^{-1} u \psi_r v$ with $r \geq 2$, rank$(u) \leq r - 1$, v a ψ-word, and $\eta(v) = 0$.
◦ Either return that f is invalid, or return a word $f' = \psi_1^{l'} v$ equivalent to f such that $\ell(f') \leq \ell(f) - 2$ if u is empty, $\ell(f') \leq \ell(f) - 4$ if $r > 2$ and rank$(u) = 1$, and otherwise, $\ell(f') \leq \ell(f) - 3$.
◦ Halt in $O(\ell(f)^{4+(k-1)})$ time.

The construction of **PinchII**$_r$ is the same as **Pinch**$_r$ except that:

1. We replace A_r by ψ_{r+1} for $r \geq 0$.
2. We replace all called subroutines by their ψ-word versions.
3. In line 4, when **PositiveII** checks the value of $u\psi_r v$, declare the word invalid if the result was invalid, positive or 0. Otherwise, run **CutRankII**$_r$(w) followed by **FinalPinchII**$_r$ when the result of **CutRankII**$_r$ is not invalid.

Before discussing the correctness of **PinchII**$_r$, we construct and analyze its subroutines **CutRankII**$_r$ and **FinalPinchII**$_r$.

Algorithm 3.6 — CutRankII$_r$ for $r \geq 2$.
◦ Input a ψ-word of the form $f := \psi_r^{-1} u \psi_r v$ with $\eta(u) = \eta(v) = 0$ and rank$(u) \leq r - 1$.
◦ Either declare that f is invalid, or **halt** and return $f' := \psi_1^l v \sim f$, or return $f' := \psi_r^{-1} u' \psi_r v \sim f$ where rank$(u') \leq r - 2$. In all cases $\ell(f') \leq \ell(f)$ and if $f' := \psi_1^l v$, then $\ell(f') \leq \ell(f) - 3$.
◦ Halt in $O(\ell(f)^{4+(k-1)})$ time.

The construction of **CutRankII**$_r$ is the same as **CutRank**$_r$ except that:

1. We replace A_r by ψ_{r+1} for $r > 0$, A_0 by ψ_1^{-1}. We replace all called subroutines by their ψ-word versions.
2. In line 6, check whether $u\psi_r v(0) = -1$. If so, run and return the result of **OneToZeroII**(w).
3. In line 11, instead of the substitution $A_r = A_{r-1} A_r A_0^{-1}$ which encodes the defining recursion relation for Ackermann functions, use Lemma 3.5 and make the substitution $\psi_r^{-1} = \psi_1 \psi_r^{-1} \psi_{r-1}^{-1} \psi_1^{-1}$ to convert w to $\psi_1 \psi_r^{-1} \psi_{r-1}^{-1} \psi_1^{-1} u' \psi_{r-1} u'' \psi_r v$ where $\eta(u) = \eta(u') = \eta(u'') = 0$ and u' has rank strictly less than $r - 1$.

Correctness of $\texttt{CutRankII}_r$ *assuming correctness of* $\texttt{PinchII}_{r-1}$. In the case $\texttt{OneToZeroII}$ is used, all claims follow from the specifications of that algorithm.

We show $\ell(f') \leq \ell(f)$. The only changes from $\texttt{CutRank}_r$ occur in the **while** loop used to remove successive ψ_{r-1}. As for $\texttt{CutRank}_r$, it suffices to check that each iteration of this loop has output no longer than its input.

$\texttt{CutRankII}_r$ returns $f' = f$ if u has rank less than $r-1$, so assume ψ_{r-1} appears in u so $\text{rank}(u) = r-1$. If $u\psi_r v(0) = -1$, then as we show for $\texttt{CutRank}_r$, after each iteration of the loop, there is no increase in length. If $u\psi_r v(0) \neq -1$, express f as $\psi_r^{-1}u'\psi_{r-1}u''\psi_r v$ where $\eta(u') = \eta(u'') = 0$, $\text{rank}(u') < k-1$ and $\text{rank}(u'') \leq k-1$. Substituting $\psi_1\psi_{r-1}\psi_r\psi_1^{-1}$ for ψ_r adds 3 letters. There is at least one letter between ψ_{r-1}^{-1} and ψ_{r-1}, so applying $\texttt{PinchII}_{r-1}$ then decreases length by at least 3. Hence when $\texttt{CutRankII}_r$ does not encounter any special cases in the **while** loop, $\ell(f') \leq \ell(f)$.

□

To adapt $\texttt{FinalPinchII}_r$ to give $\texttt{FinalPinch}_r$:

1. In line 3, check whether $u\psi_r v(0) = -1$ and, if so, run and return the result of $\texttt{OneToZeroII}(f)$.
2. In line 24, use Lemma 3.5 instead of Lemma 2.4 to make the analogous substitutions, $\psi_r^{-1} = \psi_1\psi_r^{-1}\psi_{r-1}^{-1}\psi_1^{-1}$ and $\psi_r = \psi_1\psi_{r-1}\psi_r\psi_1^{-1}$.

Algorithm 3.7 — $\texttt{FinalPinchII}_r$ for $r \geq 2$.
- ◦ Input a word of the form $\psi_r^{-1}u\psi_r v$ with $\eta(u) = \eta(v) = 0$ and $\text{rank}(u') < r-1$.
- ◦ Either return **invalid** or return an equivalent word of the form $\psi_1^l v$.
- ◦ Halt in $O(\ell(f)^{4+(r-2)})$ time.

Correctness of $\texttt{FinalPinchII}_r$ *assuming correctness of* $\texttt{PinchII}_r$. Consider the special cases:

- u **is the empty word**: the argument is similar to the case where u is the empty word in the main routine.
- $u\psi_r v(0) = -1$ **and** u **is not the empty word**: the argument is similar to the case where u is the empty word in $\texttt{PinchII}_r$.
- $v(0) = 0$: substituting $\psi_1\psi_r^{-1}\psi_{r-1}^{-1}\psi_1^{-1}$ for ψ_r^{-1} adds 3 letters. Substituting for ψ_r by ψ_{r-1} results in no increase in length in this case. As in $\texttt{CutRankII}_r$, the substitution for ψ_r^{-1} ensures that there is at least one letter between ψ_{r-1}^{-1} and ψ_{r-1}, so if $\texttt{PinchII}_r$ returns an equivalent word, that word is at least 4 letters shorter than the input word by the induction hypothesis.
- $u\psi_r v(0) < -1$ and $v(0) < 0$: substituting $\psi_1\psi_{r-1}\psi_r\psi_1^{-1}$ and $\psi_1\psi_r^{-1}\psi_{r-1}^{-1}\psi_1^{-1}$ for ψ_r and ψ_r^{-1}, respectively, adds 6 letters. Applying $\texttt{PinchII}_{r-1}$ to
$$\psi_{r-1}^{-1}\psi_1^{-1}u\psi_1\psi_{r-1}\psi_r\psi_1^{-1}\psi_1^{-1}v,$$
whose length is at most $\ell(f)+6$. There are non-trivial letters between $\psi_{r-1}^{-1}, \psi_{r-1}$. So the equivalent word returned by $\texttt{PinchII}_{r-1}$ is at least three letters shorter. Therefore, the result is of the form
$$\psi_1\psi_r^{-1}\psi_1^l\psi_r\psi_1^{-1}v$$
for some $l \in \mathbb{Z}$ and has length at most $\ell(f)+3$. If $l = 0$, running $\texttt{BasePinchII}$ triggers a trivial case where $f' = v$ is returned and $\ell(v) \leq \ell(f) - 3$ since u is non-empty. Otherwise, applying $\texttt{BasePinchII}$ to $\psi_r^{-1}\psi_1^l\psi_r\psi_1^{-1}v$, if an equivalent

word of the form $\psi_1^{l'}\psi_1^{-1}v$ is returned, its length is 4 letters shorter than the input to **BasePinchII**. Hence we have a word equivalent to f of the form

$$\psi_1\psi_1^{l'}\psi_1^{-1}v$$

whose length is at most $\ell(f)-1$, and the word is equivalent to:

$$\psi_1^{l'}v$$

yielding an equivalent word whose length is at most $\ell(f)-3$. □

Correctness of **PinchII**$_r$ *assuming the correctness of* **PinchII**$_{r-1}$. Correctness can be proved by mimicking our proof of correctness for **Pinch**$_r$. However, the substitution $A_r = A_{r-1}A_rA_0^{-1}$ for Ackermann functions increases the length of the word by 2 letters, but the substitution $\psi_r^{\pm1} = (\psi_1\psi_{r-1}\psi_r^{-1}\psi_1^{-1})^{\pm1}$ increases length by 3 letters, so we will need to account carefully for this difference.

When $r=2$, the bound on $\ell(f')$ comes directly from the bound for **BasePinchII**.

Let $r \geq 3$. The calls to **PositiveII** in the main routine are on words no longer than f. We also have the special case where u is the empty word, where the algorithm halts and returns v which has length $\ell(f)-2$. If $u\psi_kv(0) = -1$ and u is not the empty word, by part of the justification for **BasePinchII**, $\psi_kv(0) \leq v(0)-3$. Since $\eta(u) = 0$, the only letter in u that decreases absolute value when evaluating $f(0)$ letter-by-letter from right to left is ψ_1^{-1}. If $u\psi_rv(0) = -1$, then $\psi_rv(0) \leq v(0)-3$ by the specifications of **OneToZeroII**. So u must contain ψ_1^{-1} at least $|v(0)|+2$. Therefore, the $\ell(\psi_r^{-1}u\psi_r) \geq |v(0)|+4$. Thus $f' = \psi_1^{v(0)}v$ has $\ell(f') \leq \ell(f)-4$ as required. □

Correctness and construction of **ReduceII** are nearly immediate by following those of **Reduce**, replacing A_i by ψ_{i+1} and changing the subroutines to the ψ-word versions. The bound $\ell(f') \leq \ell(f)+3k$ contrasts with the bound $\ell(w') \leq \ell(w)+2k$ of **Reduce** because Lemma 3.5 requires a substitution that results in a gain of 3 letters rather than the gain of 2 required by Lemma 2.4.

Algorithm 3.8 — ReduceII.
- Input a ψ-word f with $\eta(f) > 0$.
- Either declare that f is invalid or return an equivalent word of the form f' with $\ell(f') \leq \ell(f)+3k$ and $\eta(f') = \eta(f)-1$.
- Halt in $O(\ell(f)^{4+(k-1)})$ time.

Finally, **Psi** can be constructed similarly to **Ackermann** by replacing all A_i by ψ_{i+1} and replacing subroutines by their counterparts. The proof of its correctness then essentially follows that of **Ackermann**. (The special case $k=1$ is trivial; we distinguish it to make an estimate at the end of Section 4.5 cleaner.)

Algorithm 3.9 — Psi.
- Input a ψ-word f.
- Either return that f is invalid, or return that it is valid and declare whether $f(0) > 0$, $f(0) = 0$, or $f(0) < 0$.
- Halt in $O(\ell(f)^{4+k})$ time when $k > 1$ and $O(\ell(f))$ time when $k = 1$.

4. An efficient solution to the membership problem for hydra groups

4.1. **Our algorithm in outline.** Our aim is to give a polynomial-time algorithm $\mathtt{Member}_k$ which, given a word $w = w(a_1, \ldots, w_k, t)$ on the generators of the hydra group

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

where $\theta(a_i) = a_i a_{i-1}$ for all $i > 1$ and $\theta(a_1) = a_1$, will tell us whether or not w represents an element of $H_k = \langle a_1 t, \ldots, a_k t \rangle$.

The first step is to convert w into a normal form: we use the defining relations for G_k to collect all the $t^{\pm 1}$ at the front, and then we freely reduce, to give $t^r v$ where r is an integer with $|r| \leq \ell(w)$ and $v = v(a_1, \ldots, a_m)$ is reduced. Pushing a $t^{\pm 1}$ past an a_i has the effect of applying $\theta^{\pm 1}$ to a_i, so it follows from the lemma below that

$$\ell(v) \;\leq\; \ell(w)(\ell(w) + 1)^{k-1}$$

and that $t^r v$ can be produced in time $O(\ell(w)^k)$.

Lemma 4.1. *For all $k = 1, 2, \ldots$ and all $n \in \mathbb{Z}$,*

$$\ell(\theta^n(a_k)) \;\leq\; (|n| + 1)^{k-1}.$$

Proof. For $n \in \mathbb{N}$ define $f(n,k) := \ell(\theta^n(a_k))$ and $g(n,k) = \ell(\theta^{-n}(a_k))$. To establish the lemma we will show by induction on k that $f(n,k)$ and $g(n,k)$ are each at most $(n+1)^{k-1}$.

For the case $k = 1$, note that $f(n,1) = g(n,1) = 1$ because $\theta^n(a_1) = a_1$ for all $n \in \mathbb{Z}$.

For the induction step, consider $k > 1$. As $\theta^n(a_k) = \theta^{n-1}(\theta(a_k)) = \theta^{n-1}(a_k)\theta^{n-1}(a_{k-1})$, we have

$$\begin{aligned} f(n,k) &= f(n-1,k) + f(n-1,k-1) \\ &= f(0,k) + f(0,k-1) + \cdots + f(n-1,k-1) \\ &\leq 1 + 1^{k-2} + \cdots + n^{k-2} \\ &\leq (n+1)^{k-1} \end{aligned}$$

where the first inequality uses $f(0,k) = \ell(\theta^0(a_k)) = \ell(a_k) = 1$ and the induction hypothesis, and the second that each of the $n+1$ terms in the previous line is at most $(n+1)^{k-2}$.

Next, note that $\theta^{-1}(a_k) = a_k \theta^{-1}(a_{k-1}^{-1})$ because $\theta(a_k) = a_k a_{k-1}$. So, for all $n \in \mathbb{Z}$

$$\theta^{-n}(a_k) \;=\; \theta^{-(n-1)}\theta^{-1}(a_k) \;=\; \theta^{-(n-1)}(a_k \theta^{-1}(a_{k-1}^{-1})) \;=\; \theta^{-(n-1)}(a_k)\theta^{-n}(a_{k-1}^{-1})$$

and therefore

$$\ell(\theta^{-n}(a_k)) \;=\; \ell(\theta^{-(n-1)}(a_k)) + \ell(\theta^{-n}(a_{k-1}^{-1})) \;=\; \ell(\theta^{-(n-1)}(a_k)) + \ell(\theta^{-n}(a_{k-1})).$$

So for all $n > 0$

$$\begin{aligned} g(n,k) &\leq g(n-1,k) + g(n,k-1) \\ &\leq g(0,k) + g(1,k-1) + \cdots + g(n,k-1) \\ &\leq 1 + 1^{k-2} + \cdots + (n+1)^{k-2} \\ &\leq (n+1)^{k-1} \end{aligned}$$

since $g(0,k) = 1$ and $1 + 1^{k-2}$ and each of the other n terms in the penultimate line is at most $(n+1)^{k-2}$. □

Next $\mathtt{Member}_k$ calls a subroutine $\mathtt{Push}_k$ which 'pushes' the power of t back through v from the left to the right (the power varying in the process), leaving the prefix to its left as a word on $a_1t, \ldots, a_kt$. The powers of t that occur as this proceeds are recorded by ψ-words, as they may be too large to record explicitly in polynomial time.

Here are some more details on how we 'push the power of t through v.' We do not try to progress the power of t past one $a_i^{\pm 1}$ at a time. (There are words representing elements of H_k for which that is impossible.) Instead, we first consider the locations of the $a_k^{\pm 1}$, then the $a_{k-1}^{\pm 1}$, and so on. Following [12], we define the *rank-k decomposition of v into pieces* as the (unique) way of expressing v as a concatenation $\pi_1 \cdots \pi_p$ of the minimal number of subwords ('*pieces*') π_i of the form $a_k^{\epsilon_1} u a_k^{-\epsilon_2}$ where $\text{rank}(u) \leq k-1$ and $\epsilon_1, \epsilon_2 \in \{0, 1\}$. For example, the rank-5 decomposition of

$$a_5a_3a_5^{-1}a_2a_5a_1a_5^{-1}a_1a_5^{-1}$$

is

$$(a_5a_3a_5^{-1})(a_2)(a_5a_1a_5^{-1})(a_1a_5^{-1}).$$

We use pieces because $t^r v \in H_k t^s$ for some $s \in \mathbb{Z}$ if and only if it is possible to advance the power of t^r through v one piece at a time, leaving behind an element of H_k. More precisely, $t^r v \in H_k t^s$ if and only if there exists a sequence $r = r_0, \ldots, r_p = s$ such that $t^{r_i}\pi_{i+1} \in H_k t^{r_{i+1}}$ (Lemma 6.2 of [12]).

Let $f_0 := \psi_1^{-r}$, so $f_0(0) = r$. Then, for each successive i, we determine, using a subroutine $\mathbf{Piece}_k$, whether or not there exists $r_i \in \mathbb{Z}$ (unique if it exists) such that

$$t^{f_{i-1}(0)}\pi_i \ \in \ H_k t^{r_i}$$

and if so, it gives a ψ-word f_i such that $f_i(0) = r_i$. $\mathbf{Piece}_k$ expresses π_i as $a_k^{\epsilon_1} u a_k^{-\epsilon_2}$ where $\epsilon_1, \epsilon_2 \in \{0, 1\}$. It operates in accordance with Proposition 4.10 which is a technical result that we call 'The Piece Criterion.' $\mathbf{Piece}_k$ has two subroutines. The first, $\mathbf{Front}_k$, reduces the problem of whether r_i exists to determining whether, for a certain ψ-word f'_{i-1} and a certain rank-k piece π'_i which does not have a_m as its first letter, there exists $r'_i \in \mathbb{Z}$ such that $t^{f'_{i-1}(0)}\pi'_i \in H_{k-1}t^{r'_i}$. Then the second, $\mathbf{Back}_k$, makes a similar reduction to a situation when there is no a_m^{-1} at the end. It then inductively calls $\mathbf{Push}_{k-1}$ on the modified piece (which is now a word of rank less than k) to find a ψ-word f'_i representing r'_i, and then modifies f'_i to get f_i. It detects that the r_i fails to exist by recognizing (using $\mathbf{Psi}$) an emerging ψ-word not being valid, or noticing that π_i fails to have a suffix or prefix of a particular form.

This inductive construction has base cases $\mathbf{Push}_1$ and $\mathbf{Piece}_2$, which use elementary direct manipulations.

If $r_1, \ldots, r_p$ all exist, then $\mathbf{Psi}$ determines whether or not $f_p(0) = 0$, and concludes that w does or does not represent an element of H_k, accordingly.

4.2. **Examples.** The algorithms and subroutines named here are those we will construct in Section 4.5.

Example 4.2. Let $w = a_3^4a_2ta_1a_2^{-1}a_3^{-4}$. As we saw in Section 1.4, $w = u_{3,4}\,(a_2t)\,(a_1t)\,(a_2t)^{-1}\,u_{3,4}{}^{-1}$ in G_3 which has length $2\mathcal{H}_3(4) + 3 = 2^{47} \cdot 3 - 1$ as a word on the generators a_1t, a_2t, a_3t of H_3. Here is how our algorithm $\mathbf{Member}_k$ discovers that w represents an element of H_3 without working with this prohibitively long word.

1. Convert w to a word tv representing the same element of G_3 by using that $a_it = t\theta(a_i)$ in G_3 for all i to shuffle the t to the front. This produces
$$v \ = \ \theta(a_3)^4\theta(a_2)a_1a_2^{-1}a_3^{-4} \ = \ (a_3a_2)^4a_2a_1^2a_2^{-1}a_3^{-4}.$$
2. Define $f_0 := \psi_1^{-1}$, to express the power $f(0) = 1$ of t here.

3. The rank-3 decomposition of v into pieces is:
$$v = (a_3a_2)(a_3a_2)(a_3a_2)(a_3a_2^2a_1^2a_2^{-1}a_3^{-1})(a_3^{-1})(a_3^{-1})(a_3^{-1}).$$
Accordingly, define
$$\pi_1 := \pi_2 := \pi_3 := a_3a_2, \qquad \pi_4 := a_3a_2^2a_1^2a_2^{-1}a_3^{-1}, \qquad \pi_5 := \pi_6 := \pi_7 := a_3^{-1}.$$
A subroutine $\mathtt{Push}_3$ now aims to find ψ-words $f_1, \ldots, f_7$ such that $t^{f_{i-1}(0)}\pi_i \in H_3t^{f_i(0)}$ for $i = 1, \ldots, 7$, by 'pushing the power of t through successive pieces.'
4. So first a subroutine $\mathtt{Piece}_3$ is called to try to pass $t^{f_0(0)}$ through π_1. The subroutine $\mathtt{Front}_k$ calls a further subroutine $\mathtt{Prefix}_3$ to find the longest prefix (if one exists) of π_1 of the form $\theta^{i-1}(a_3)a_2$ for some $i \geq 1$. $\mathtt{Prefix}_3$ does so by generating $\theta^0(a_3)a_2$, $\theta^1(a_3)a_2$, and so on, and comparing, until the length of π_1 is exceeded. In this instance $\mathtt{Prefix}_3$ returns $i = 1$. It follows from the Piece Criterion that $t^{f_0(0)}\pi_1 = a_3t \in H_3t^0 = H_3t^{\psi_1\psi_1^{-1}(0)}$. Accordingly define $f_1 := \psi_1\psi_1^{-1}$.
5. $\mathtt{Piece}_3$ next looks to pass $t^{f_1(0)} = t^0$ through π_2. $\mathtt{Front}_k$ uses $\mathtt{Psi}$ to check that $f_1(0) = 0 \leq 0$. By the Piece Criterion, it then follows from the fact that there are no inverse letters in π_2 that $ta_3a_2 \in Ht^{\psi_2\psi_3(0)}$. So define $f_2 := \psi_2\psi_3\psi_1\psi_1^{-1}$.
6. Next $\mathtt{Piece}_3$ tries to pass $t^{f_2(0)}$ through $\pi_3 = a_3a_2$. Likewise this is possible as $f_2(0) \leq 0$, and it defines $f_3 := (\psi_2\psi_3)^2\psi_1\psi_1^{-1}$.
7. Next, $\mathtt{Piece}_3$ tries to pass $t^{f_3(0)}$ through π_4.
   7.1. $\mathtt{Front}_3$ uses $\mathtt{Psi}$ to check that $f_3(0) \leq 0$. It follows that $t^{f_3(0)}a_3 \in H_3t^{\psi_3 f_3(0)}$ and the problem is reduced (by the Piece Criterion) to finding an $s \in \mathbb{Z}$ (if one exists) such that
$$t^{\psi_3 f_3(0)}a_2^2a_1^2a_2^{-1}a_3^{-1} \in H_3t^s.$$
This will represent progress as (unlike π_4) $a_2^2a_1^2a_2^{-1}a_3^{-1}$ is a piece without an a_m at the front.
   7.2. Then the subroutine $\mathtt{Back}_3$ recursively calls $\mathtt{Piece}_2$ to find the $s \in \mathbb{Z}$ (if there is one) such that $t^{\psi_3 f_3(0)}a_2^2a_1^2a_2^{-1} \in H_3t^s$. It returns $\psi_2^{-1}(\psi_1)^2\psi_2^2\psi_3 f_3$. (We omit the steps $\mathtt{Piece}_2$ goes through.) $\mathtt{Back}_3$ then uses $\mathtt{Psi}$ to test whether $f_4 := \psi_3^{-1}\psi_2^{-1}(\psi_1)^2\psi_2^2\psi_3 f_3$ is valid, which it is: we examined it in Section 3.2. Also $\mathtt{Psi}$ declares that $f_4(0) \leq 0$. It follows (using the Piece Criterion) that $t^{f_3(0)}\pi_4 \in H_3t^{f_4(0)}$.
8. Next $\mathtt{Piece}_3$ tries to pass $t^{f_4(0)}$ through π_5. This is done by $\mathtt{Back}_3$. By the Piece Criterion, it suffices to check that $f_5 := \psi_3^{-1} f_4$ is valid, which is done using $\mathtt{Psi}$.
9. $\mathtt{Piece}_3$ likewise passes $t^{f_5(0)}$ through π_6 giving $f_6 := \psi_3^{-2} f_4$, and then $t^{f_6(0)}$ through π_7 giving $f_7 := \psi_3^{-3} f_4$.
10. Finally, let $g := f_7$. We have that $w = tv \in H_3t^{g(0)}$. So use $\mathtt{Psi}$ to check that $g(0) = 0$. On success, declare that $w \in H_3$.

In the example above $f_i(0) \leq 0$ for all i—we never looked to push a positive power of t through a piece. Next we will see an example of $\mathtt{Member}_k$ handling such a situation.

Example 4.3. Let $w = ta_3a_2t^2a_1^{-1}a_2^{-2}a_3^{-1}a_1^2t^{-1}a_3^{-1}$. We will show how $\mathtt{Member}_k$ discovers that $w \in H_3$.

1. Shuffle the $t^{\pm1}$ in w to the front, applying $\theta^{\pm1}$ to letters they pass, so as to convert w to the word t^2v representing the same element of G_3, where $v = a_3a_2^2a_1^2a_2^{-1}a_3^{-1}a_1^2a_3^{-1}$. Let $f = \psi_1^{-2}$ so that $f(0) = 2$ records the power of t.
2. Express v as its the rank-3 decomposition into pieces: $v = \pi_1\pi_2$ where
$$\pi_1 := a_3a_2^2a_1^2a_2^{-1}a_3^{-1}, \qquad \pi_2 := a_1^2a_3^{-1}.$$
Set $f_0 := f$. $\mathtt{Push}_3$ now looks for valid ψ-words f_1 and f_2 such that $t^{f_0(0)}\pi_1 \in H_3t^{f_1(0)}$ and $t^{f_1(0)}\pi_2 \in H_3t^{f_2(0)}$, by twice calling its subroutine $\mathtt{Piece}_3$.

3. **Piece**$_3$ calls **Front**$_3$ to 'try to move $t^{f_0(0)}$ past π_1.' As a_3 is the first letter of π_1, **Front**$_3$ calls **Psi** to determine the sign of $f_0(0)$, which is positive. The Piece Criterion then says that to pass t^2 past a_3 requires that π_1 has a prefix $\theta^{i-1}(a_3)a_2$ for some i which is 'approximately' $\theta^2(a_3) = a_3a_2^2a_1$. The subroutine **Prefix**$_3$ looks for this prefix by generating $\theta^0(a_3)a_2 = a_3a_2$, then $\theta^1(a_3)a_2 = a_3a_2^2$, then $\theta^2(a_3)a_2 = a_3a_2^2a_1a_2$, and so on, until the length of π is exceeded, and comparing with the start of π_1. Here, a_3a_2 and $a_3a_2^2$ are prefixes of π_1, but $a_3a_2^2a_1a_2$ is not, and **Prefix**$_3$ returns $i = 2$.
4. Call **Psi** to check that i is at least $f_0(0) = 2$.
5. Intuitively speaking, as this prefix $a_3a_2^2$ is 'approximately' $\theta^2(a_3)$, the length of the 'correction' $a_1a_1^{-1}$ that has to be made for the discrepancy between $\theta^2(a_3)$ and the prefix $a_3a_2^2$ is minimal compared to the length of the prefix that the power of t advances past. In this instance:

$$t^2\pi_1 \;=\; t^2\theta^2(a_3)a_1a_1^{-1}a_1a_2^{-1}a_3^{-1} \;=\; (a_3t)ta_1a_2^{-1}a_3^{-1}.$$

 and have reduced the problem to pushing t past $a_1a_2^{-1}a_3^{-1}$. The power of t being advanced through the word is now t^1, and this is recorded by $\psi_1 f_0$, as $\psi_1 f_0(0) = 1$.
6. Next **Piece**$_3$ calls **Back**$_3$ on input $a_1a_2^{-1}a_3^{-1}$ and $\psi_1 f$ to try to advance t past $a_1a_2^{-1}a_3^{-1}$.
7. First, it searches for an $s \leq 0$ such that $ta_1a_2^{-1}a_3^{-1} \in H_kt^s$. It calls **Push**$_2$, which calls **Piece**$_2$ to attempt to push t through $a_1a_2^{-1}$. **Piece**$_2$ calls Ψ to find out whether $\psi_2^{-1}\psi_1\psi_1 f$ is valid. It is not, and it follows from the Piece Criterion that there is no $s \leq 0$ such that $ta_1a_2^{-1}a_3^{-1} \in H_kt^s$.
8. So, instead **Piece**$_3$ searches for an $s > 0$ such that $ta_1a_2^{-1}a_3^{-1} \in H_kt^s$ or, equivalently, $t^sa_3a_2a_1^{-1} \in H_3t$.
9. We check for $s = 1, 2, \ldots$ whether we can move t^s past $a_3a_2a_1^{-1}$. Use the same approach that we used for the prefix in Step 5. First try $s = 1$. Detect the prefix a_3a_2 of $a_3a_2a_1^{-1}$ and as, $ta_3a_2 = t\theta(a_3) = (a_3t) \in H_3$, the problem reduces to determining whether $t^0a_1^{-1} \in H_3t$ or, equivalently, $ta_1 \in H_3t^0$. This shown to be the case by **Push**$_2$ which finds that $ta_1 = (a_1t) \in H_3$ and returns $\psi_1\psi_1 f$, which satisfies $\psi_1\psi_1 f(0) = 0$, to indicate the coset H_3t^0 of H_3. Finally, **Back**$_3$ checks that $H_3t^0 = H_3t^{\psi_1\psi_1 f_0}$ by calling **psi** on $\psi_1^0\psi_1\psi_1 f_0(0) = 0$, and returns $f_1 := \psi_1^{-1}\psi_1^2 f_0$ (which satisfies $f_1(0) = 1$) to indicate that $\pi_1 \in H_3t^{f_1(0)}$.

   (In this instance, we were successful with $s = 1$, but in general, we may have to repeat the process for $s = 2, 3, \ldots$. This does not continue indefinitely: we can stop when s exceeds the length of of the word inputted into **Back**$_3$ because the prefixes we check for must be no longer than that word.)
10. We now seek to pass $t^{f_1(0)}$ through π_2 by another call on **Piece**$_3$. Recall $\pi_2 = a_1^2a_3^{-1}$ and $f_1 := \psi_1^{-1}\psi_1^2 f_0$, and $f_1(0) = 1$.
11. **Piece**$_3$ first calls **Front**$_3$ but the first letter of π_2 is not a_3, so **Front**$_3$ does nothing.
12. **Piece**$_3$ then calls **Back**$_3$. It first looks for $s \leq 0$ such that $t^{f_1(0)}\pi_2 \in H_kt^s$, which it succeeds in finding as follows.
    - 12.1. **Push**$_2$ tries to pass $t^{f_1(0)}$ through a_1^2, which is elementary since a_1 commutes with t: $ta_1^2 = (a_1t)(a_1t)t^{-1}$ and so **Push**$_2$ returns $\psi_1^2 f_1$, representing $\psi_1^2 f_1(0) = -1$.
    - 12.2. Call **Psi** to check that $\psi_1^2 f_1$ is valid. Then to pass t through a_3^{-1}, call **Psi** to check that $\psi_3^{-1}\psi_1^2 f_1$ is valid. Return $f_2 := \psi_3^{-1}\psi_1^2 f_1$ to indicate that $t^{f_1(0)}\pi_2 \in H_3t^{f_2(0)}$.
13. **Member**$_3$ checks that $f_2(0) = 0$ and declares that $w \in H_3$.

These examples illustrate the tests $\mathtt{Member}_k$ uses and give a sense of how it works in general. But, it is difficult to show that these tests amount to the *only* conditions under which a word $t^r v$ is in Ht^s for some $s \in \mathbb{Z}$. A result we call the 'Piece Criterion' is at the heart of that and presentation and proof of is involved and will occupy the next two sections.

4.3. **Constraining cancellation.** This section contains preliminaries toward Proposition 4.10 (The Piece Criterion), which will be the subject of the next section.

When discussing words representing elements of $F(a_1, \ldots, a_m)$, we use $\theta^r(a_m^{\pm 1})$, for $m \geq 1$ and $r \in \mathbb{Z}$, to refer to the freely reduced word on $a_1, \ldots, a_m$ equal to $\theta^r(a_m^{\pm 1})$. The following lemma will be useful for calculating with iterations of θ.

Lemma 4.4. *If $r > 0$ and $m > 1$, then*

$$\theta^r(a_m) \;=\; a_m \theta^0(a_{m-1})\theta^1(a_{m-1})\theta^2(a_{m-1}) \cdots \theta^{r-1}(a_{m-1}) \tag{17}$$

as words. Moreover, if $r < m$, then the final letter of $\theta^r(a_m)$ is a_{m-r}, and if $r \geq m$, then $\theta^{r-m+1}(a_1) = a_1$, $\theta^{r-m+2}(a_2), \ldots, \theta^{r-1}(a_{m-1})$ are all suffixes of $\theta^r(a_m)$.

If $r < 0$ and $m > 1$, then

$$\theta^r(a_m) \;=\; a_m \theta^{-1}(a_{m-1}^{-1})\theta^{-2}(a_{m-1}^{-1}) \cdots \theta^r(a_{m-1}^{-1}), \tag{18}$$

as words, and its first letter is a_m and its final letter is a_{m-1}^{-1}.

Proof. For (17), observe that the identity $\theta^r(a_m) = \theta^{r-1}(a_m)\theta^{r-1}(a_{m-1})$ and inducting on r gives that the words are equal in the free group. The words are identical because that on the right is positive (that is, contains no inverse letters) and so is freely reduced. If $r < m$, the same identity shows that the final letter of $\theta^r(a_m)$, is the same as that of $\theta^{r-1}(a_{m-1})$, and so the same as that of $\theta^{r-2}(a_{m-2}), \ldots$, and of $\theta^{r-r}(a_{m-r}) = a_{m-r}$. If, on the other hand, $r \geq m$, then (17) shows that $\theta^{r-1}(a_{m-1})$ is a suffix of $\theta^r(a_m)$, and therefore, so are $\theta^{r-2}(a_{m-2})$, $\theta^{r-3}(a_{m-3}), \ldots, \theta^{r-m+1}(a_1)$.

Lemma 7.1 in [12] tells us that the two words in (18) are freely equal. Induct on m as follows to establish the remaining claims. In the case $m = 2$ we have

$$\theta^r(a_2) \;=\; a_2\theta^{-1}(a_1^{-1})\theta^{-2}(a_1^{-1}) \cdots \theta^r(a_1^{-1}) \;=\; a_2 a_1^r,$$

and the result holds. For $m > 2$, the induction hypothesis tells us that the first letter of each subword $\theta^{-i}(a_{m-1}^{-1})$ is a_{m-2} and the final letter is a_{m-1}^{-1}, and it follows that the word on the right of (18) is freely reduced. It is then evident that its first letter is a_m and its final letter is a_{m-1}^{-1}. □

The remainder of this section concerns words w expressed as

$$w \;=\; \theta^{e_0}(a_{i_0}^{\epsilon_0})\theta^{e_1}(a_{i_1}^{\epsilon_1}) \cdots \theta^{e_{l+1}}(a_{i_{l+1}}^{\epsilon_{l+1}})$$

where $\epsilon_x \in \{\pm 1\}$ for $x = 0, \ldots, l+1$, and $a_{i_x}^{\epsilon_x} \neq a_{i_{x+1}}^{-\epsilon_{x+1}}$ and

$$e_{x+1} \;=\; \begin{cases} e_x & \text{if } \epsilon_x = -\epsilon_{x+1} \\ e_x - 1 & \text{if } \epsilon_x = \epsilon_{x+1} = 1 \\ e_x + 1 & \text{if } \epsilon_x = \epsilon_{x+1} = -1 \end{cases} \tag{19}$$

for $x = 0, \ldots, l$. We refer to the $a_{i_0}^{\epsilon_0}, \ldots, a_{i_{l+1}}^{\epsilon_{l+1}}$ in the subwords $\theta^{e_0}(a_{i_0}^{\epsilon_0}), \theta^{e_1}(a_{i_1}^{\epsilon_1}), \ldots, \theta^{e_{l+1}}(a_{i_{l+1}}^{\epsilon_{l+1}})$ of w as the *principal letters* of w.

Lemma 4.5. *If w (as above) freely equals the empty word, then $a_{i_x} = a_{i_{x+1}}$ and $\epsilon_{i_x} = -\epsilon_{x+1}$ for some $0 \leq x < l+1$.*

Proof. The point of the hypotheses is that w is the word obtained by shuffling all $t^{\pm 1}$ rightwards in

$$\begin{cases} t^{-e_0}(a_{i_0}t)^{\epsilon_0}\cdots(a_{i_{l+1}}t)^{\epsilon_{j+1}} & \text{if } \epsilon_0 = 1 \\ t^{-e_0+1}(a_{i_0}t)^{\epsilon_0}\cdots(a_{i_{l+1}}t)^{\epsilon_n} & \text{if } \epsilon_0 = -1, \end{cases}$$

and then discarding the power of t that emerges on the right.

Now $(a_{i_0}t)^{\epsilon_1}\cdots(a_{i_{l+1}}t)^{\epsilon_{l+1}} = 1$ in H_k because $w = 1$ in G_k and $H_k \cap \langle t\rangle = \{1\}$ (Lemma 6.1 in [12]). The result then follows from the fact that H_k is free on $a_1t, \ldots, a_kt$ (Proposition 4.1 in [12]). $\square$

The following definition and Proposition 4.7 concerning it are for analyzing free reduction of w. They will be used in our proof of Proposition 4.9, where we will subdivide a word such as w into subwords of certain types and argue that all free reduction is contained within them. There are two ideas behind the definitions of these types. One is that the rank-1 and rank-2 letters are the most awkward for understanding free reduction, but in these subwords such letters are *controlled* by being buttressed by higher rank words. The other idea concerns where new letters appear when $\theta^{\pm 1}$ is applied to some $a_n^{\pm 1}$. It is evident from the definition of θ that when $i \geq 0$, the lower rank letters produced by applying θ^i to a_n or a_n^{-1} appear to the right of a_n and to the left of a_n^{-1}. The same is true when $i < 0$ — see Lemma 7.1 of [12].

Definition 4.6. We will define various *types* a subword

$$z \ = \ \theta^{e_p}(a_{i_p}^{\epsilon_p})\cdots\theta^{e_q}(a_{i_q}^{\epsilon_q})$$

of w may take, and will denote the freely reduced form of z by z'. To the left, below, are the conditions that define the types. To the right are facts established in the proposition that follows: what z' is in cases ii and ii^{-1}, and prefixes and suffixes it has in cases *i*–*iv*. When it appears below, u denotes a (possibly empty) subword $\theta^{e_x}(a_{i_x}^{\epsilon_x})\cdots\theta^{e_y}(a_{i_y}^{\epsilon_y})$ such that $i_x, \ldots, i_y \leq 2$.

$$\begin{array}{lll}
(i) & \epsilon_p = 1, \ \ \epsilon_q = -1 & z \ = \theta^{e_p}(a_{i_p})u\theta^{e_q}(a_{i_q}^{-1}) \\
 & i_p, i_q \geq 3, \ \ i_{p+1}, \ldots, i_{q-1} \leq 2 & z' = \theta^{e_p-1}(a_{i_p})\text{———}a_{i_q}^{-1} \ \text{ if } e_p > 0 \\
 & e_p, e_q \geq 0 & \quad = a_{i_p}\text{———}a_{i_q}^{-1} \ \text{ for } e_p \geq 0
\end{array}$$

$$\begin{array}{lll}
(ii) & \epsilon_p, \ldots, \epsilon_q = 1 & z \ = \theta^{e_p}(a_{i_p})\cdots\theta^{e_q}(a_{i_q}) \\
 & i_p \geq 3, \ i_q \geq 2 & z' = \theta^{e_p+1}(a_{i_p})\theta^{e_q}(a_{i_q-1}^{-1}) \\
 & i_j = i_{j+1}+1 \text{ for } j = p, \ldots, q-1 & \quad = a_{i_p}\text{———}a_{i_q-1}^{-1} \\
 & e_p < 0 & \\
 & (\text{so } e_{p+1}, \ldots, e_q < 0 \text{ by } (19)) &
\end{array}$$

$$\begin{array}{lll}
(ii^{-1}) & \epsilon_p, \ldots, \epsilon_q = -1 & z \ = \theta^{e_p}(a_{i_p}^{-1})\cdots\theta^{e_q}(a_{i_q}^{-1}) \\
 & i_q \geq 3, \ i_p \geq 2 & z' = \theta^{e_p}(a_{i_p-1})\theta^{e_q+1}(a_{i_q}^{-1}) \\
 & i_j = i_{j-1}+1 \text{ for } j = p+1, \ldots, q & \quad = a_{i_p-1}\text{———}a_{i_q}^{-1} \\
 & e_q < 0 & \\
 & (\text{so } e_p, \ldots, e_{q-1} < 0 \text{ by } (19)) &
\end{array}$$

$$\begin{array}{lll}
(iii) & p < q' \leq q & z \ = \theta^{e_p}(a_{i_p})u\theta^{e_{q'}}(a_{i_{q'}}^{-1})\cdots\theta^{e_q}(a_{i_q}^{-1}) \\
 & \epsilon_p = 1, \ \ \epsilon_{q'}, \ldots, \epsilon_q = -1 & z' = \theta^{e_p-1}(a_{i_p})\text{———}a_{i_q}^{-1} \ \text{ if } e_p > 0 \\
 & i_p, i_{q'}, \ldots, i_q \geq 3, & \quad = a_{i_p}\text{———}a_{i_q}^{-1} \ \text{ for } e_p \geq 0 \\
 & i_{p+1}, \ldots, i_{q'-1} < 3 & \\
 & i_j = i_{j-1}+1 \text{ for } j = q'+1, \ldots, q & \\
 & e_p \geq 0, \ \ e_q < 0 & \\
 & (\text{so } e_{q'}, \ldots, e_{q-1} < 0 \text{ by } (19)) &
\end{array}$$

(iii^{-1})	$p \leq p' < q$ $\epsilon_p, \ldots, \epsilon_{p'} = -1, \;\; \epsilon_q = 1$ $i_p, \ldots, i_{p'}, i_q \geq 3$ $i_j = i_{j+1} + 1$ for $j = p, \ldots, p' - 1$ $e_p < 0, \;\; e_q \geq 0$ (so $e_{p+1}, \ldots, e_{p'} < 0$ by (19))	$z = \theta^{e_p}(a_{i_p}) \cdots \theta^{e_{p'}}(a_{i_{p'}}) u \theta^{e_q}(a_{i_q}^{-1})$ $z' = a_{i_p}$ ——— $a_{i_q}^{-1}$
(iv)	$p \leq p' < q' \leq q$ $\epsilon_p, \ldots, \epsilon_{p'} = 1, \;\; \epsilon_{q'}, \ldots, \epsilon_q = -1$ $i_p, \ldots, i_{p'}, i_{q'}, \ldots, i_q \geq 3$ $i_{p'+1}, \ldots, i_{q'-1} < 3$ $i_j = i_{j+1} + 1$ for $j = p, \ldots, p' - 1$ $i_j = i_{j-1} + 1$ for $j = q' + 1, \ldots, q$ $e_p, e_q < 0$ (so $e_{p+1}, \ldots, e_{p'} < 0$ and $e_{q'}, \ldots, e_{q-1} < 0$ by (19))	$z = \theta^{e_p}(a_{i_p}) \cdots \theta^{e_{p'}}(a_{i_{p'}}) u \theta^{e_{q'}}(a_{i_{q'}}^{-1}) \cdots \theta^{e_q}(a_{i_q}^{-1})$ $z' = a_{i_p}$ ——— $a_{i_q}^{-1}$
(v)	For no $0 \leq p' < q' \leq l + 1$ with $p \leq q' \leq q$ is $\theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{p'}}) \cdots \theta^{e_{q'}}(a_{i_{q'}}^{\epsilon_{q'}})$ one of the above types.	$z = \theta^{e_p}(a_{i_p}^{\epsilon_p}) \cdots \theta^{e_q}(a_{i_q}^{\epsilon_q})$ $z' = \theta^{e_p - 1}(a_{i_p})$——— if $\epsilon_p = 1, i_p \geq 3$ and $e_p > 0$

Proposition 4.7. *In types i, ii$^{\pm 1}$, iii$^{\pm 1}$, iv and v the form of z' is as indicated in Definition 4.6. In type v, no letter of rank 3 or higher in z cancels away on free reduction to z'.*

Proof of Proposition 4.7 in type i. We have

$$z \;=\; \theta^{e_p}(a_{i_p}) u \theta^{e_q}(a_{i_q}^{-1})$$

where $i_p, i_q \geq 3$, and $e_p, e_q \geq 0$, and u is a subword of w of rank at most 2. By definition

$$u \;=\; \theta^{e_{p+1}}(a_{i_{p+1}}^{\epsilon_{p+1}}) \cdots \theta^{e_{q-1}}(a_{i_{q-1}}^{\epsilon_{q-1}}), \tag{20}$$

and by Lemma 4.5, no a_2 and a_2^{-1} can cancel in the process of freely reducing u. We aim to show that the first and last letters of the freely reduced form z' of z are a_{i_p} and $a_{i_q}^{-1}$, respectively, and that if $e_p > 0$, then $\theta^{e_p - 1}(a_{i_p}) a_{i_p - 1}$ is a prefix of z'. We will also show that if $e_q > 0$, then $a_{i_q - 1}^{-1} \theta^{e_q - 1}(a_{i_q}^{-1})$ is a suffix of z'. This is more than claimed in the proposition, but having a conclusion that is 'symmetric' with respect to inverting z' will expedite our proof.

We organize our proof by cases.

1. *Case: u freely equals the empty word.* In this case u is empty else Lemma 4.5 (applied to u rather than to w) would be contradicted. So $z = \theta^{e_p}(a_{i_p}) \theta^{e_q}(a_{i_q}^{-1})$ and by (19), $e_p = e_q$. Now $\theta^{e_p}(a_{i_p})$ contains an a_2 if and only if $i_p - 2 \leq e_p$, and in that event $\theta^{e_p - i_p + 2}(a_2) = a_2 a_1^{e_p - i_p + 2}$ is a suffix of $\theta^{e_p}(a_{i_p})$. Similarly, $\theta^{e_q}(a_{i_q}^{-1})$ contains an a_2^{-1} if and only if $i_q - 2 \leq e_q$, and in that event $\theta^{e_q - i_q + 2}(a_2) = a_1^{-(e_q - i_q + 2)} a_2^{-1}$ is a prefix of $\theta^{e_q}(a_{i_q}^{-1})$. If $i_p - 2 > e_p$, then $i_p > e_p$, and so the final letter of $\theta^{e_p}(a_{i_p})$ is $a_{i_p - e_p}$. Likewise, if $i_q - 2 > e_q$, then $a_{i_q - e_q}^{-1}$ is the first letter of $\theta^{e_q}(a_{i_q}^{-1})$.
   1.1. *Case: cancellation occurs between some letters $a_2^{\pm 1}, \ldots, a_k^{\pm 1}$ when z is freely reduced to z'.* If $i_p - 2 \leq e_p$, then the final a_2 in $\theta^{e_p}(a_{i_p})$ must cancel with the first a_2^{-1} in $\theta^{e_q}(a_{i_q}^{-1})$. So $i_q - 2 \leq e_q$, and the whole suffix $a_2 a_1^{e_p - i_p + 2}$ of $\theta^{e_p}(a_{i_p})$

cancels with the whole prefix $a_1^{-(e_q-i_q+2)}a_2^{-1}$ of $\theta^{e_q}(a_{i_q}^{-1})$. But that implies that $i_p = i_q$ (since $e_p = e_q$), which is a contradiction. If, on the other hand, $i_p - 2 > e_p$, then $i_q - 2 > e_q$, and the last and first letters $a_{i_p-e_p}$ and $a_{i_q-e_q}^{-1}$ of $\theta^{e_p}(a_{i_p})$ and $\theta^{e_q}(a_{i_q}^{-1})$, respectively, must be mutual inverses, and so again we get the contradiction $i_p = i_q$.

1.2. *Case: no cancellation occurs between letters $a_2^{\pm 1}, \ldots, a_k^{\pm 1}$ when z is freely reduced to z'.* If $i_p - 2 > e_p$ or $i_q - 2 > e_q$, then the last letter of $\theta^{e_p}(a_{i_p})$ or the first letter of $\theta^{e_q}(a_{i_q}^{-1})$, respectively, has rank greater than 2 and so is not cancelled away, and therefore $z' = z$. If $i_p - 2 \leq e_p$ and $i_q - 2 \leq e_q$, then there is only cancellation between some of the $a_1^{e_p-i_p+2}$ at the end of $\theta^{e_p}(a_{i_p})$ and some of the $a_1^{-(e_q-i_q+2)}$ at the start of $\theta^{e_q}(a_{i_q}^{-1})$ (but not all as $i_p \neq i_q$). In either event the first and last letters of z' are a_{i_p} and $a_{i_q}^{-1}$, respectively. Moreover, if $e_p > 0$, then $\theta^{e_p-1}(a_{i_p})a_{i_p-1}$ is a prefix of z' as a_{i_p-1} has rank at least 2 and so is not cancelled away. Likewise, if $e_q > 0$, then $a_{i_q-1}^{-1}\theta^{e_q-1}(a_{i_q}^{-1})$ is a suffix of z'.

2. *Case: u does not freely equal the empty word.*

2.1. *Case: no letter $a_3^{\pm 1}, \ldots, a_k^{\pm 1}$ in z is cancelled away when z is freely reduced to give z'.* The first and last letters, a_{i_p} and $a_{i_q}^{-1}$, of z are also the first and last letters of z', because $i_p, i_q \geq 3$. Here is why the prefix $\theta^{e_p-1}(a_{i_p})a_{i_p-1}$ of z survives in z' when $e_p > 0$. If $i_p \geq 4$, then its final letter a_{i_p-1} has rank at least 3 and so is not cancelled away. Suppose then that $i_p = 3$, so that the prefix

$$\theta^{e_p}(a_{i_p}) = \theta^{e_p}(a_3) = \theta^{e_p-1}(a_3)\theta^{e_p-1}(a_2) = \theta^{e_p-1}(a_3)a_2a_1^{e_p-1}.$$

We must show that the a_2 of $\theta^{e_p-1}(a_3)a_2$ is not cancelled away when z is freely reduced to z'. Suppose it is cancelled away. Then u must have a prefix freely equal to $a_1^{-(e_p-1)}a_2^{-1}$ (since no a_2 and a_2^{-1} can cancel when u freely reduces). But u has the form (20), and by a calculation we will see in a more extended form in (28), $a_1^{-e_p+2m_1}a_2^{-1}$ freely equals a prefix of u for some integer m_1. But then $-(e_p - 1) = -e_p + 2m_1$, contradicting m_1 being an integer. Conclude that $\theta^{e_p-1}(a_3)a_2$ is a prefix of z' as required. Likewise, if $e_q > 0$, then $a_{i_q-1}^{-1}\theta^{e_q-1}(a_{i_q}^{-1})$ is a suffix of z'.

2.2. *Case: some letter $a_3^{\pm 1}, \ldots, a_k^{\pm 1}$ in z is cancelled away when z is freely reduced to give z'.* The prefix $\theta^{e_p}(a_{i_p})$ of z is a positive word and the suffix $\theta^{e_q}(a_{i_q}^{-1})$ is a negative word since $e_p, e_q \geq 0$.
There is an a_3 in $\theta^{e_p}(a_{i_p})$ if and only if $e_p - i_p + 3 \geq 0$. Likewise there is an a_3^{-1} in $\theta^{e_q}(a_{i_q}^{-1})$ if and only if $e_q - i_q + 3 \geq 0$.

2.2.1. *Case: $e_p - i_p + 3 < 0$.* The last letter of $\theta^{e_p}(a_{i_p})$ (a positive word) has rank greater than 3 and so must cancel. So $e_q - i_q + 3 < 0$ also, as otherwise $\theta^{e_q}(a_{i_q}^{-1})$ (a negative word) the leftmost letter in $\theta^{e_q}(a_{i_q}^{-1})$ with rank at least 3 would be an a_3^{-1}, which would block any cancelation of other letters $a_3^{\pm 1}, \ldots, a_k^{\pm 1}$ in z. So, in fact, the last letter of $\theta^{e_p}(a_{i_p})$ must cancel with the first letter of $\theta^{e_q}(a_{i_q}^{-1})$, and so u must equal freely the identity, which is a case addressed above.

2.2.2. *Case: $e_q - i_q + 3 < 0$.* Likewise, this reduces to the earlier case.
The remaining possibility is:

2.2.3. *Case: $e_p - i_p + 3 \geq 0$ and $e_q - i_q + 3 \geq 0$.* So $\theta^{e_p}(a_{i_p})$ has suffix

$$\theta^{e_p-i_p+3}(a_3) = a_3a_2\, a_2a_1\, a_2a_1^2\, \cdots a_2a_1^{e_p-i_p+2}$$

and $\theta^{e_q}(a_{i_q}^{-1})$ has prefix

$$\theta^{e_q-i_q+3}(a_3^{-1}) \;=\; a_1^{-(e_q-i_q+2)}a_2^{-1} \,\cdots\, a_1^{-2}a_2^{-1}\, a_1^{-1}a_2^{-1}\, a_2^{-1}a_3^{-1}$$

and the subword

$$\theta^{e_p-i_p+3}(a_3)u\theta^{e_q-i_q+3}(a_3^{-1}) \tag{21}$$

of z freely equals the identity. Now u has rank at most 2, so

$$u \;=\; a_1^{f_1}a_2^{-1}a_1^{f_2}a_2^{-1}\cdots a_1^{f_\lambda}a_2^{-1}a_1^{\xi}a_2a_1^{g_\mu}\cdots a_2a_1^{g_2}a_2a_1^{g_1}$$

for some $\lambda,\mu \geq 0$, some $\xi \in \mathbb{Z}$, some $f_1,\ldots,f_\lambda \leq 0$, and some $g_1,\ldots,g_\mu \geq 0$. And because of cancellations that must occur,

$$\begin{array}{lcllcl} f_1 & = & -(e_p-i_p+2) & \qquad g_1 & = & e_q-i_q+2 \\ f_2 & = & -(e_p-i_p+1) & \qquad g_2 & = & e_q-i_q+1 \\ & \vdots & & & \vdots & \\ f_\lambda & = & -(e_p-i_p+3-\lambda) & \qquad g_\mu & = & e_q-i_q+3-\mu. \end{array}$$

These cancellations reduce $\theta^{e_p-i_p+3}(a_3)u\theta^{-(e_q-i_q+3)}(a_3)$ to

$$a_3a_2\, a_2a_1\, a_2a_1^2\,\cdots a_2a_1^{e_p-i_p+2-\lambda}\; a_1^{\xi}\; a_1^{-(e_q-i_q+2-\mu)}a_2^{-1}\cdots\, a_1^{-2}a_2^{-1}\, a_1^{-1}a_2^{-1}\, a_2^{-1}a_3^{-1}.$$

As this freely equals the identity, the exponent sum of the $a_2^{\pm 1}$ is zero, and so

$$e_p-i_p+3-\lambda \;=\; e_q-i_q+3-\mu. \tag{22}$$

Also, as the $a_1^{\pm 1}$ between the rightmost a_2 and the leftmost a_2^{-1} cancel,

$$e_p-i_p+2+\mu+\xi \;=\; e_q-i_q+2+\lambda. \tag{23}$$

Together (22) and (23) tell us that $\xi = 0$. But then $\lambda = 0$ or $\mu = 0$ because of the hypothesis $a_{i_x}^{\epsilon_x} \neq a_{i_x}^{-\epsilon_{x+1}}$ in the instance of the a_2^{-1} and a_2 (which must be principal letters) in u each side of the a_1^{ξ}.
Suppose $\mu = 0$, which we can do without loss of generality because what we are setting out to prove is symmetric with respect to inverting z and z'. Then

$$u \;=\; a_1^{-(e_p-i_p+2)}a_2^{-1}a_1^{-(e_p-i_p+1)}a_2^{-1}\cdots a_1^{-(e_p-i_p+3-\lambda)}a_2^{-1}. \tag{24}$$

After u has cancelled into $\theta^{e_p}(a_{i_p})$, the word $\theta^{e_p-i_p+3}(a_3)u\theta^{-(e_q-i_q+3)}(a_3)$ becomes

$$a_3a_2\, a_2a_1\, a_2a_1^2\,\cdots a_2a_1^{e_p-i_p+3-\lambda-1}\;\; a_1^{-(e_q-i_q+2)}a_2^{-1}\;\cdots\; a_1^{-2}a_2^{-1}\, a_1^{-1}a_2^{-1}\, a_2^{-1}a_3^{-1} \tag{25}$$

and, as the powers of a_1 and a_1^{-1} must cancel in the middle of this word,

$$e_p-i_p-\lambda \;=\; e_q-i_q. \tag{26}$$

There are no a_2 among the principal letters in u (expressed as (20)), and the a_2^{-1} principal letters are those that occur in (24). The final principal letter $a_{i_{q-1}}^{\epsilon_{q-1}}$ must be a_2^{-1} as that is the final letter in (24). The remaining principal letters are a_1 or a_1^{-1}, and an a_1 principal letter is never adjacent to an a_1^{-1} principal letter. So we can encode the sequence $a_{i_{p+1}}^{\epsilon_{p+1}},\ldots,a_{i_{q-1}}^{\epsilon_{q-1}}$ using integers $m_1,\ldots,m_\lambda \in \mathbb{Z}$, as:

$$\underbrace{a_1^{\mathrm{sign}(m_1)},\ldots,a_1^{\mathrm{sign}(m_1)}}_{|m_1|},a_2^{-1},\underbrace{a_1^{\mathrm{sign}(m_2)},\ldots,a_1^{\mathrm{sign}(m_2)}}_{|m_2|},a_2^{-1},\;\;\ldots,\;\;\underbrace{a_1^{\mathrm{sign}(m_\lambda)},\ldots,a_1^{\mathrm{sign}(m_\lambda)}}_{|m_\lambda|},a_2^{-1}.$$

But (19) and the hypothesis that $\epsilon_p = 1$ allow us to determine $e_{p+1}, \ldots, e_{q-1}$ from e_p and $m_1, \ldots, m_\lambda$, so as to deduce that

$$u = a_1^{m_1}\theta^{e_p-m_1}(a_2^{-1})a_1^{m_2}\theta^{e_p-m_1-m_2+1}(a_2^{-1})\cdots a_1^{m_\lambda}\theta^{e_p-m_1-\cdots-m_\lambda+\lambda-1}(a_2^{-1}) \tag{27}$$

$$= a_1^{-e_p+2m_1}a_2^{-1}a_1^{-1-e_p+m_1+2m_2}a_2^{-1}\cdots a_1^{-\lambda+1-e_p+m_1+\cdots+m_{\lambda-1}+2m_\lambda}a_2^{-1}. \tag{28}$$

Comparing the powers of a_1 here with those in (24), we get:

$$\left\{\begin{array}{rcccccccccccc} -2+i_p & = & & & 2m_1 & & & & & & & & \\ -1+i_p & = & -1 & + & m_1 & + & 2m_2 & & & & & & \\ i_p & = & -2 & + & m_1 & + & m_2 & + & 2m_3 & & & & \\ & \vdots & & & & & & & & & & & \\ \lambda-3+i_p & = & 1-\lambda & + & m_1 & + & m_2 & + & \cdots & + & m_{\lambda-1} & + & 2m_\lambda, \end{array}\right. \tag{29}$$

which simplifies to

$$i_p + 2^{j+1} - 6 = 2^j m_j \qquad \text{for } j = 1, \ldots, \lambda. \tag{30}$$

2.2.3.1. *Case* $\lambda = 0$. This is a case we have previously addressed: u is the empty word.

So we can assume that $\lambda \geq 1$, and then the $j = 1$ instance of (30) tells us that i_p is even, and so

$$i_p \geq 4. \tag{31}$$

2.2.3.2. *Case* $\lambda = 1$. By (26),

$$e_p - i_p - 1 = e_q - i_q. \tag{32}$$

Also

$$z = \theta^{e_p}(a_{i_p})\underbrace{\theta^{e_{p+1}}(a_1^{\text{sign}(m_1)})\cdots\theta^{e_{p+|m_1|}}(a_1^{\text{sign}(m_1)})}_{|m_1|}\theta^{e_p-m_1}(a_2^{-1})\theta^{e_q}(a_{i_q}^{-1})$$

by (27), and so (19) applied to $\theta^{e_p-m_1}(a_2^{-1})$ and $\theta^{e_q}(a_{i_q}^{-1})$ tells us that $e_q = e_p - m_1 + 1$. But $i_p - 2 = 2m_1$ by the $j = 1$ case of (30), and so

$$e_q = e_p - \frac{i_p-2}{2} + 1. \tag{33}$$

By (32) and (33),

$$i_p + 1 = i_q + \frac{i_p-2}{2} - 1,$$

and so

$$i_p + 6 = 2i_q. \tag{34}$$

So (31) implies $i_q \geq 5$. And we can assume that it is not the case that $e_p - i_p + 3 = e_q - i_q + 3 = 0$, else (32) would be contradicted. So $e_p - i_p + 3 > 0$ or $e_q - i_q + 3 > 0$. If $e_p - i_p + 3 > 0$, there are at least two a_3 in $\theta^{e_p}(a_{i_p})$ (because $i_p \geq 4$) and hence at least two a_3^{-1} in $\theta^{e_q}(a_{i_q}^{-1})$. Likewise, if $e_q - i_q + 3 > 0$, then there are at least two a_3^{-1} in $\theta^{e_q}(a_{i_q}^{-1})$ (because $i_q \geq 4$), and so two a_3 in $\theta^{e_p}(a_{i_p})$. In either case, using Lemma 4.4 to identify the relevant suffix of $\theta^{e_p}(a_{i_p})$ and prefix of $\theta^{e_q}(a_{i_q}^{-1})$, there is a subword

$$\theta^{e_p-i_p+2}(a_3)\theta^{e_p-i_p+3}(a_3)u\theta^{e_q-i_q+3}(a_3^{-1})\theta^{e_q-i_q+2}(a_3^{-1}), \tag{35}$$

of z, which contains exactly two a_3 and two a_3^{-1}. If (35) freely reduces to the empty word, then, once the inner a_3 and a_3^{-1} pair have cancelled, it reduces to $\theta^{e_p-i_p+2}(a_3)\theta^{e_q-i_q+2}(a_3^{-1})$, which must therefore also freely reduce to the empty word. But then $e_p - i_p + 2 = e_q - i_q + 2$, also contradicting (26). So (35) must not freely reduce to the empty word, and its first letter (an a_3) and its last letter (an a_3^{-1}) are not cancelled away. If $i_p \neq 4$, then the required conclusions about the prefix and suffix of z' follow because the a_3 and a_3^{-1} bookending (35) do not cancel away and cannot cancel with a prefix $\theta^{e_p-1}(a_{i_p})a_{i_p-1}$ or first letter a_p or suffix $a_{i_q-1}^{-1}\theta^{e_q-1}(a_{i_q}^{-1})$ or final letter a_q^{-1}, because $i_p \geq 5$ and $i_q \geq 5$. If $i_p = 4$, then $i_q = 5$ by (34). And by (32), $e_p = e_q$. Now, by (27), $u = a_1^{m_1}\theta^{e_p-m_1}(a_2^{-1})$.

2.2.3.3. *Case* $\lambda \geq 2$. Then (30) in the case $j = 2$ tells us that $i_p = 4m_2 - 2$, and in particular $i_p \neq 4$ as $m_2 \in \mathbb{Z}$.

At this point we know $i_p \geq 3$ (by hypothesis), is even, and is not 4. So $i_p \geq 6$.

If $e_p - i_p + 3 = 0$, then there is exactly one a_3 in $\theta^{e_p}(a_{i_p})$, specifically its final letter. So the subword $a_3 u \theta^{e_q-i_q+3}(a_3^{-1})$ must freely equal the empty word. But $u = a_1^{-e_p+2m_1}a_2^{-1}a_1^{-1-e_p+m_1+2m_2}a_2^{-1}$ by (28) and $\theta^{e_q-i_q+3}(a_3^{-1})$ is a negative word as $e_q - i_q + 3 \geq 0$, so no cancellation is possible: a contradiction.

So, given that $e_p - i_p + 3 \geq 0$, we deduce that $e_p - i_p + 2 \geq 0$, and so (as $i_p \geq 6$) there are at least two letters a_3 in $\theta^{e_p}(a_{i_p})$. But then, as above, if (35) freely reduces to the empty word, $e_p - i_p + 2 = e_q - i_q + 2$, but then by (23) and that $\mu = \xi = 0$, we find $\lambda = 0$, which is a case we have already addressed. So the first and last letters (a_3 and a_3^{-1}, respectively) of (35) are not cancelled away, and therefore the first and last letters (a_{i_p} and $a_{i_q}^{-1}$, respectively) of z are also those of z', as required. And, as $i_p \geq 6$, if $e_p > 0$, then the prefix $\theta^{e_p}(a_{i_p})$ of z survives into z' as it ends with a letter of rank at least 5 which is not cancelled away. And likewise, if $i_q \geq 5$ and $e_q > 0$, then the suffix $\theta^{e_q}(a_{i_q}^{-1})$ of z survives into z'.

Suppose then that i_q is 3 or 4 and $e_q > 0$.

The exponent sum of the a_2 in z between the rightmost a_3 of $\theta^{e_p}(a_{i_p})$ and the leftmost a_3^{-1} of $\theta^{e_q}(a_{i_q}^{-1})$ is zero, so

$$e_p - i_p + 3 \;=\; e_q - i_q + 3 + \lambda.$$

Applying (19) to the suffix $\theta^{e_p-m_1-\cdots-m_\lambda+\lambda-1}(a_2^{-1})$ of u (expressed as per (27)) and $\theta^{e_q}(a_{i_q}^{-1})$, we get

$$e_q \;=\; e_p - m_1 - \cdots - m_\lambda + \lambda.$$

Adding these two equations together and simplifying yields:

$$-i_p \;=\; -i_q + 2\lambda - m_1 - \cdots - m_\lambda.$$

The final equation of (29) is

$$\lambda - 3 + i_p \;=\; 1 - \lambda + m_1 + m_2 + \cdots + m_{\lambda-1} + 2m_\lambda.$$

Summing the preceding two equations and simplifying gives

$$-4 \;=\; -i_q + m_\lambda.$$

But i_q is 3 or 4, so m_λ is -1 or 0, But, $i_p + 2^{\lambda+1} - 6 \;=\; 2^\lambda m_\lambda$ by (30), which implies that $m_\lambda > 0$ because $i_p \geq 6$ and $\lambda \geq 0$—a contradiction.

□

Proof of Proposition 4.7 in type ii. The result will follow from the type ii^{-1} instance of the proposition, proved below, because z is the inverse of a word of type ii^{-1}. □

Proof of Proposition 4.7 in type ii^{-1}. The hypotheses dictate that in type ii^{-1}, z has the form:

$$z \;=\; \theta^{e_p}(a_{i_p}^{-1})\theta^{e_p+1}(a_{i_p+1}^{-1})\cdots\theta^{e_q}(a_{i_q}^{-1}),$$

where $e_q - e_p = i_q - i_p$. We must show that its freely reduced form is

$$z' \;=\; \theta^{e_p}(a_{i_p-1})\theta^{e_q+1}(a_{i_q}^{-1}).$$

Well,

$$\begin{aligned}\theta^{e_q+1}(a_{i_q}^{-1}) \;&=\; \theta^{e_q}(a_{i_q-1}^{-1})\theta^{e_q}(a_{i_q}^{-1}) \\ &=\; \theta^{e_q-1}(a_{i_q-2}^{-1})\theta^{e_q-1}(a_{i_q-1}^{-1})\theta^{e_q}(a_{i_q}^{-1}) \\ &\;\;\vdots \\ &=\; \theta^{e_p}(a_{i_p-1}^{-1})\theta^{e_p}(a_{i_p}^{-1})\theta^{e_p+1}(a_{i_p+1}^{-1})\cdots\theta^{e_q}(a_{i_q}^{-1}),\end{aligned}$$

and so z' and z are freely equal.

When $e_p < 0$ and $i_p - 1 > 1$, Lemma 4.4 tells us that the final letter of $\theta^{e_p}(a_{i_p-1})$ is $a_{i_p-2}^{-1}$. And when $e_q + 1 < 0$ and $i_q > 1$, it tells us that the first letter of $\theta^{e_q+1}(a_{i_q}^{-1})$ is a_{i_q-1}. Our hypotheses include that $e_q < 0$, which implies that $e_p < 0$ as $e_p < e_q$, and that $i_q > 1$, so in all cases except when $i_p = 2$ or $e_q = -1$, we learn that z' is freely reduced as required.

When $i_p = 2$ and $e_q \neq -1$,

$$z' \;=\; a_1\theta^{e_q+1}(a_{i_q}^{-1}),$$

which is freely reduced because the first letter of $\theta^{e_q+1}(a_{i_q}^{-1})$ is $a_{i_q} - 1$. And when $e_q = -1$ and $i_p - 1 \neq 1$,

$$z' \;=\; \theta^{e_p}(a_{i_p-1})a_{i_q}^{-1},$$

which is freely reduced because the last letter of $\theta^{e_p}(a_{i_p-1})$ is a_{i_p-2}. And when $e_q = -1$ and $i_p - 1 = 1$,

$$z' \;=\; a_1a_{i_q}^{-1},$$

which is freely reduced because $i_q \geq 3$.

The first letter of z is a_{i_p-1} by Lemma 4.4 applied to $\theta^{e_p}(a_{i_p-1})$. The final letter of z is $a_{i_q}^{-1}$ because the first letter of $\theta^{e_q+1}(a_{i_q})$ is a_{i_q} by the same lemma. □

Proof of Proposition 4.7 in type iii. We have that

$$z \;=\; \theta^{e_p}(a_{i_p})u\theta^{e_{q'}}(a_{i_{q'}}^{-1})\cdots\theta^{e_q}(a_{i_q}^{-1})$$

where $i_p, i_{q'}, \ldots, i_q \geq 3$, $i_{p+1}, \ldots, i_{q'-1} < 3$, $e_p \geq 0$, $e_q < 0$ (and so $e_{q'}, \ldots, e_{q-1} < 0$ by (19)). Also $i_j = i_{j-1} + 1$ for $j = q' + 1, \ldots, q$, so $i_q = i_{q'} + q - q'$. Like in type i, we must show that the first and last letters of the freely reduced form z' of z are a_{i_p} and $a_{i_q}^{-1}$, respectively, and that if $e_p > 0$, then $\theta^{e_p-1}(a_{i_p})$ is a prefix of z'.

Proposition 4.7 for type ii^{-1}, proved above, applied to the suffix $\theta^{e_{q'}}(a_{i_{q'}}^{-1})\cdots\theta^{e_q}(a_{i_q}^{-1})$, tells us that z freely equals

$$\theta^{e_p}(a_{i_p})\,u\,\theta^{e_{q'}}(a_{i_{q'}-1})\theta^{e_{q'}+q-q'+1}(a_{i_{q'}+q-q'}^{-1}) \tag{36}$$

and that the new suffix $\theta^{e_{q'}}(a_{i_{q'}-1})\theta^{e_{q'}+q-q'+1}(a_{i_{q'}+q-q'}^{-1})$ is reduced.

By hypothesis, $i_{q'} \geq 3$. We again organize our proof by cases.

1. *Case:* $i_{q'} \geq 4$. As the suffix $\theta^{e_{q'}}(a_{i_{q'}-1})\theta^{e'_q+q-q'+1}(a^{-1}_{i_{q'}+q-q'})$ of (36) is freely reduced, its first letter is $a_{i_{q'}-1}$, which has rank at least 3 by hypothesis and so cannot cancel any letter in u, and is positive and so cannot cancel with a letter in $\theta^{e_p}(a_{i_p})$. Therefore letters in u can only cancel with the $\theta^{e_p}(a_{i_p})$ to its left. So the final letter of z' is $a^{-1}_{i_{q'}+q-q'} = a^{-1}_{i_q}$, as required. As $\text{rank}(u) \leq 2$ and $i_p \geq 3$, the first letter a_p of z is also the first letter of z', as required. It remains to show that, assuming $e_p > 0$, the prefix $\theta^{e_p-1}(a_{i_p})$ of z' is also a prefix of z'. If $i_p > 3$, this is immediate because a_{i_p-1} has rank at least 3 and so cannot cancel into u. If $i_p = 3$, then no $a_2^{\pm 1}$ in u cancel with $\theta^{e_p}(a_{i_p})$ for otherwise the first equation of (29) the argument from type i would adapt to this setting to give us the contradiction that i_p is even.
2. *Case:* $i_{q'} = 3$.
   - 2.1. *Case:* $i_q \leq 2$. This does not occur because, by hypothesis, $i_{q'} \geq 3$ and $q - q' \geq 0$.
   - 2.2. *Case:* $i_q \geq 4$. Suppose, for a contradiction, that the first or last letter of z cancels away on free reduction, or that $e_p > 0$ and the prefix $\theta^{e_p-1}(a_{i_p})a_{i_p-1}$ (which is one letter longer than we need) of $\theta^{e_p}(a_{i_p})$ fails to also be a prefix of z'.
     - 2.2.1. *Case:* $e_{q'} + q - q' + 1 = 0$. Here, as $i_{q'} + q - q' = i_q \geq 4$, (36) is
$$\theta^{e_p}(a_{i_p})\, u\, \theta^{e_{q'}}(a_2)a^{-1}_{i_q}.$$
Then $\theta^{e_p}(a_{i_p})$ can contain no a_3 since there is no a_3^{-1} to cancel with. Therefore, $\theta^{e_p}(a_{i_p})$ ends with a letter of rank greater than 3 by Lemma 4.4. For this reason, u cannot cancel to its left, and so $u\theta^{e_{q'}}(a_2)$ freely equals the empty word. By Lemma 4.5, u cannot contain a rank 2 subword that freely equals the empty word, so $u = a_1^{\mu}\theta^{e_{q'-1}}(a_2^{-1})$ for some $\mu \in \mathbb{Z}$. But then by (19) $e_{q'-1} = e_{q'} - 1$, and $u = a_1^{\mu}\theta^{e_{q'}-1}(a_2^{-1})$. Counting the exponent sum of the $a_1^{\pm 1}$ in $u\theta^{e_{q'}}(a_2)$, we find
$$\mu - e_{q'} + 1 + e_{q'} = 0.$$
So $\mu = -1$, and u must be $\theta^{e_{p+1}}(a_1^{-1})\theta^{e_{q'}-1}(a_2^{-1})$. But then applying (19) to $\theta^{e_p}(a_{i_p})\theta^{e_{p+1}}(a_1^{-1})\theta^{e_{q'}-1}(a_2^{-1})$, we find that $e_{q'} - 1 = e_p + 1 \geq 1$, contradicting the fact that $e_{q'} < 0$.
     - 2.2.2. *Case:* $e_{q'} + q - q' + 1 < 0$. Here, (36) is
$$\theta^{e_p}(a_{i_p})\, u\, \theta^{e_{q'}}(a_2)\theta^{e_{q'}+q-q'+1}(a^{-1}_{i_q}).$$
The first letter a_{i_q-1} of the suffix $\theta^{e_{q'}+q-q'+1}(a^{-1}_{i_q})$ has rank at least 3, and must cancel to the left, but has exponent $+1$. Every other letter to the left with exponent -1 has rank at most 2, so this letter cannot be canceled to its left or right. Thus z' must end with $a^{-1}_{i_q}$ and start with a_{i_p}.
If $i_p > 3$ and $e_p > 0$, the letter immediately after the prefix $\theta^{e_p-1}(a_{i_p})$ of z is a_{i_p-1}, which is of rank at least 3, so the prefix $\theta^{e_p-1}(a_{i_p})$ must be preserved because letters of rank 3 or higher cannot cancel as there are no letters of rank 3 or higher between and the first letter a_{i_q-1} (of rank at least 3) of the suffix $\theta^{e_{q'}+q-q'+1}(a^{-1}_{i_q})$.
If $i_p = 3$, it is conceivable that this prefix is partially canceled away by some following subword u of z of rank 2 or less. We will show this leads to a contradiction so does not occur. If any letters in $\theta^{e_p}(a_{i_p})u$ of rank 2 or higher cancel, then $e_p - i_p + 2 \geq 0$ because otherwise $\theta^{e_p}(a_{i_p})$

ends with a letter of rank greater than 3. However, then u must have a prefix that cancels with $\theta^{e_p-i_p+2}(a_2)$ and so is $\theta^{e_{p+1}}(a_1)\cdots\theta^{e_{s-1}}(a_1)\theta^{e_s}(a_2^{-1})$ or $\theta^{e_{p+1}}(a_1^{-1})\cdots\theta^{e_{s-1}}(a_1^{-1})\theta^{e_s}(a_2^{-1})$ for some s. In either case, this simplifies to $a_1^{\mu}\theta^{e_s}(a_2^{-1})$ for some $\mu\in\mathbb{Z}$ and, by (19), $e_p-\mu=e_s$. By summing the exponents of the $a_1^{\pm1}$ in $\theta^{e_p-i_p+2}(a_2)$ and in $a_1^{\mu}\theta^{e_s}(a_2^{-1})$, we find that: $e_p-i_p+2-e_s+\mu=0$. But combined with $e_p-\mu=e_s$, this tells us that $\mu=(i_p-2)/2$, which is not an integer if $i_p=3$. so we have the required contradiction.

2.3. *Case:* $i_q=3$. In this instance, $q=q'$ because $i_{q'}=3$, and so $i_q=3$. So

$$z \;=\; \theta^{e_p}(a_{i_p})u\theta^{e_{q'}}(a_3^{-1}).$$

By Lemma 4.4, there is one a_3^{-1} in $\theta^{e_{q'}}(a_3^{-1})$, specifically its final letter. Suppose this a_3^{-1} cancels with an a_3 (necessarily the rightmost) in $\theta^{e_p}(a_{i_p})$. Then the intervening subword (which has rank at most 2) freely reduces to the empty word.

Now $\theta^{e_p}(a_{i_p})$ contains no a_2^{-1} because $e_p\geq 0$. The same is true of $\theta^{e_{q'}}(a_3^{-1})$ by Lemma 4.4 and the fact that $e_{q'}<0$. So, if u contains an a_2, it must cancel with an a_2^{-1} from u, and so u must contain a subword which starts and ends with principal letters of rank 2 and which freely equals the empty word, violating Lemma 4.5. Conclude that u contains no a_2.

2.3.1. *Case:* $e_p-i_p+2\geq 0$. The rightmost a_3 in $\theta^{e_p}(a_{i_p})$ is the first letter of the suffix $a_3a_2\theta^1(a_2)\cdots\theta^{e_p-i_p+2}(a_2)$, so some prefix of u freely equals the inverse of $a_2\theta^1(a_2)\cdots\theta^{e_p-i_p+2}(a_2)$. This prefix of u must be

$$\theta^{e_{p+1}}(a_{i_{p+1}}^{\epsilon_{p+1}})\cdots\theta^{e_s}(a_{i_s}^{\epsilon_s}) \tag{37}$$

for some s. (The prefix does not end in the midst of some $\theta^{e_s}(a_{i_s}^{\epsilon_s})$, because it must have final letter a_2^{-1}.)

Similarly to (27) and (28) in the type i case, we can use (19) to re-express (37) as

$$\begin{aligned} &a_1^{\nu_{\chi+1}}\theta^{e_s+\nu_1+\cdots+\nu_\chi-\chi}(a_2^{-1})\cdots a_1^{\nu_2}\theta^{e_s+\nu_1-1}(a_2^{-1})a_1^{\nu_1}\theta^{e_s}(a_2^{-1})\\ &\quad=\; a_1^{\nu_{\chi+1}-(e_s+\nu_1+\cdots+\nu_\chi-\chi)}a_2^{-1}\cdots a_1^{\nu_2-(e_s+\nu_1-1)}a_2^{-1}a_1^{\nu_1-e_s}a_2^{-1}\end{aligned}$$

for some s where $\chi:=e_p-i_p+2$ (so that $\chi+1$ is the number of a_2 in $a_2\theta^1(a_2)\cdots\theta^{e_p-i_p+2}(a_2)$) and $\nu_1,\ldots,\nu_\chi\in\mathbb{Z}$ record the number of and exponents of the $a_1^{\pm1}$ between the a_2^{-1}. As this freely equals

$$(a_2\theta^1(a_2)\theta^2(a_2)\cdots\theta^\chi(a_2))^{-1} \;=\; a_1^{-\chi}a_2^{-1}\cdots a_1^{-2}a_2^{-1}a_1^{-1}a_2^{-1}a_2^{-1},$$

we find that

$$\begin{aligned} \nu_1-e_s &= 0\\ \nu_2-(e_s+\nu_1-1) &= -1\\ \vdots\qquad &= \vdots\\ \nu_{\chi+1}-(e_s+\nu_1+\cdots+\nu_\chi-\chi) &= -\chi.\end{aligned}$$

It follows that

$$\nu_{\chi+1} \;=\; 2^\chi e_s-2^{\chi+1}+2. \tag{38}$$

The suffix $\theta^{e_p-i_p+2}(a_2)$ of $\theta^{e_p}(a_{i_p})$ must be the inverse of the prefix $a_1^{\nu_{\chi+1}}\theta^{e_s+\nu_1+\cdots+\nu_\chi-\chi}(a_2^{-1})$ of u, so $\theta^{e_p-i_p+2}(a_2)a_1^{\nu_{\chi+1}}\theta^{e_s+\nu_1+\cdots+\nu_\chi-\chi}(a_2^{-1})$ freely reduces to the empty word. By (19) applied to $\theta^{e_p}(a_{i_p})a_1^{\nu_{\chi+1}}\theta^{e_s+\nu_1+\cdots+\nu_\chi-\chi}(a_2^{-1})$,

$$e_p-\nu_{\chi+1}=e_s+\nu_1+\cdots+\nu_\chi-\chi.$$

By counting the $a_1^{\pm 1}$ in $\theta^{e_p - i_p + 2}(a_2) a_1^{\nu_{\chi+1}} \theta^{e_p - \nu_{\chi+1}}(a_2^{-1})$, which freely reduces to the empty word, we find

$$e_p - i_p + 2 + \nu_{\chi+1} = e_p - \nu_{\chi+1},$$

so that $\nu_{\chi+1} = (i_p - 2)/2$. But then $\nu_{\chi+1} > 0$, since $i_p \geq 3$. Further, we conclude that for u to even cancel an a_2 from $\theta^{e_p}(a_{i_p})$, i_p must be even. So $i_p \geq 4$. Thus after rewriting (38) as

$$e_s = \frac{1}{2^\chi}(\nu_{\chi+1} + 2^{\chi+1} - 2) \tag{39}$$

and using the fact that $\nu_{\chi+1} > 0$ and $\chi \geq 1$, we conclude that $e_s > 0$. The remainder

$$\theta^{e_{s'}}(a_{i_{s'}}^{\epsilon_{s'}}) \cdots \theta^{e_{q'-1}}(a_{i_{q'-1}}^{\epsilon_{q'-1}}), \tag{40}$$

(where $s' = s+1$) of u cancels with all but the a_3^{-1} of

$$\theta^{e_{q'}}(a_3^{-1}) = \theta^{e_{q'}}(a_2)\theta^{e_{q'}+1}(a_2) \cdots \theta^{-1}(a_2)a_3^{-1}. \tag{41}$$

We claim that, similarly to (27), we can rewrite (40) as

$$\begin{aligned} & a_1^{\eta_r}\theta^{e_{q'}+\eta_1+\eta_2+\eta_3+\cdots+\eta_{r-1}-r}(a_2^{-1}) \cdots a_1^{\eta_2}\theta^{e_{q'}+\eta_1-2}(a_2^{-1})a_1^{\eta_1}\theta^{e_{q'}-1}(a_2^{-1}) \\ = \ & a_1^{\eta_r-(e_{q'}+\eta_1+\eta_2+\eta_3+\cdots+\eta_{r-1}-r)}a_2^{-1} \cdots a_1^{\eta_2-(e_{q'}+\eta_1-2)}a_2^{-1}a_1^{\eta_1-(e_{q'}-1)}a_2^{-1} \end{aligned}$$

where r is the number of a_2^{-1} in (40), and $\eta_1, \ldots, \eta_r \in \mathbb{Z}$ record the number of and the signs of the intervening terms $\theta^*(a_1^*)$. There is no power of a_1 at the righthand end because the first letter of (41) is a_2. The iterates of θ are identified by using (19).

Now compare with (41), with which it cancels (to leave only a_3^{-1}), to see that $r = |e_{q'}|$ and

$$\begin{aligned} 0 &= \eta_1 - (e_{q'} - 1) + e_{q'} \\ 0 &= \eta_2 - (e_{q'} + \eta_1 - 2) + e_{q'} + 1 \\ \vdots &= \qquad \vdots \\ 0 &= \eta_r - (e_{q'} + \eta_1 + \eta_2 + \cdots + \eta_{r-1} - r) + e_{q'} + (r-1). \end{aligned}$$

Next we establish by induction that $\eta_i < 0$ and

$$e_{q'} + \eta_1 + \eta_2 + \cdots + \eta_{i-1} - i < 0 \tag{42}$$

for all $1 \leq i \leq r$. For the base case, $e_q - 1 < 0$ because of our hypothesis that $e_q < 0$, and $\eta_1 = -1$ by the first of the above family of equations. For the induction step, suppose $\eta_1, \ldots, \eta_{i-1} < 0$ and $e_{q'} + \eta_1 + \eta_2 + \cdots + \eta_{i-2} - (i-1) < 0$. The family of equations above tells us in particular, that

$$0 = \eta_i - (e_{q'} + \eta_1 + \eta_2 + \cdots + \eta_{i-1} - i) + e_{q'} + (i-1)$$

which rearranges to

$$(\eta_1 + \eta_2 + \cdots + \eta_{i-1}) - 2i + 1 = \eta_i.$$

So, $\eta_i < 0$ because $1 \leq i$ and $\eta_1, \ldots, \eta_{i-1} < 0$. Moreover,

$$e_{q'} + \eta_1 + \eta_2 + \cdots + \eta_{i-1} - i = (e_{q'} + \eta_1 + \eta_2 + \cdots + \eta_{i-2} - (i-1)) + \eta_{i-1} - 1 < 0$$

because $e_{q'} + \eta_1 + \eta_2 + \cdots + \eta_{i-2} - (i-1) < 0$ and $\eta_{i-1} < 0$.

Now

$$e_{s'} = \eta_r + (e_{q'} + \eta_1 + \eta_2 + \cdots + \eta_{r-1} - r) - 1$$

by (19). Conclude that $e_{s'} < 0$.

But

$$u \;=\; \theta^{e_{p+1}}(a_{i_{p+1}}^{\epsilon_{p+1}})\cdots\theta^{e_s}(a_{i_s}^{\epsilon_s})\theta^{e_{s'}}(a_{i_{s'}}^{\epsilon_{s'}})\cdots\theta^{e_{q'-1}}(a_{i_{q'-1}}^{\epsilon_{q'-1}})$$

and by (19), e_s and $e_{s'}$ differ by at most 1. So, as we previously established that $e_s > 0$, we have a contradiction.

We deduce that no a_3 and a_3^{-1} cancel when z freely reduces.

Since no letters of rank 3 can cancel, if $i_p \geq 4$, then z' has a prefix $\theta^{e_p-1}(a_{i_p})$, since cancelling any part of this prefix in $\theta^{e_p}(a_{i_p}) = \theta^{e_p-1}(a_{i_p})\theta^{e_p-1}(a_{i_p-1})$ requires cancellation of a_{i_p-1}. Finally consider the case $i_p = 3$. We showed (immediately above (39)) that if i_p is odd, then no letters of rank 2 can cancel from $\theta^{e_p}(a_{i_p})$. The remainder of the argument is the same as in the case $i_p \geq 4$.

2.3.2. *Case: $e_p - i_p + 2 < 0$.* We have $z = \theta^p(a_{i_p})u\theta^{e_q}(a_3^{-1})$ where $i_q = 3$, $q = q'$, $u = \theta^{e_{p+1}}(a_{i_{p+1}}^{\epsilon_{p+1}})\cdots\theta^{e_{q'-1}}(a_{i_{q'-1}}^{\epsilon_{q'-1}})$, and $\theta^{e_p}(a_{i_p})$ ends with a letter of rank at least 3. Suppose, for a contradiction, some letter of the prefix $\theta^{e_p}(a_{i_p})$ is cancelled when z is freely reduced to z'. No cancellation is possible between $\theta^{e_p}(a_{i_p})$ and u because every letter of $\theta^{e_p}(a_{i_p})$ is rank 3 or higher. By the argument used in Case 2.3.1 to show that $e_{s'} < 0$, we find here that $e_{p+1} < 0$, and by the argument there (immediately after (42)) to show that $\eta_r < 0$, we find here that $\epsilon_{p+1} = -1$. But then by (19), $e_p = e_{p+1}$, and so $e_p < 0$, which contradicts $e_p \geq 0$. So the first letter a_{i_p} of z is also the first letter of z', and the last letter a_3^{-1} of $\theta^{e_{i_{q'}}}(a_3^{-1})$ is also the last letter of z'. Moreover, if $e_p > 0$, then the prefix $\theta^{e_p-1}(a_{i_p})$ of $\theta^{e_p}(a_{i_p})$ is also a prefix of z'. □

Proof of Proposition 4.7 in type iii^{-1}. Inverting a type iii^{-1} word gives a type *iii* word, so we can apply the type *iii* of Proposition 4.7 proved above to get the result (as in this case we are only concerned with the first and last letters and not with a longer prefix). □

Proof of Proposition 4.7 in type iv. We must show that if $i_p, \ldots, i_{p'}, i_{q'}, \ldots, i_q \geq 3$ with $i_j = i_{j+1} + 1$ for $j = p, \ldots, p' - 1$ and $i_j = i_{j-1} + 1$ for $j = q' + 1, \ldots, q$, and $e_p, e_q < 0$, the freely reduced form z' of

$$z \;=\; \theta^{e_p}(a_{i_p})\cdots\theta^{e_{p'}}(a_{i_{p'}})u\theta^{e_{q'}}(a_{i_{q'}}^{-1})\cdots\theta^{e_q}(a_{i_q}^{-1})$$

starts with a_{i_p} and ends with $a_{i_q}^{-1}$.

By Proposition 4.7 in type $ii^{\pm 1}$, proved above, z freely reduces to

$$\theta^{e_p+1}(a_{i_p})\theta^{e_{p'}}(a_{i_{p'}-1}^{-1})u\theta^{e_{q'}}(a_{i_{q'}-1})\theta^{e_q+1}(a_{i_q}^{-1}) \tag{43}$$

where $\theta^{e_p+1}(a_{i_p})\theta^{e_{p'}}(a_{i_{p'}-1}^{-1})$ and $\theta^{e_{q'}}(a_{i_{q'}-1})\theta^{e_q+1}(a_{i_q}^{-1})$ are freely reduced.

We again organize our proof by cases.

1. *Case: $i_p = i_q$.* Suppose, for a contradiction, that z' does not start with a_{i_p} and end with $a_{i_q}^{-1}$. Then the first and last letter must cancel each other since they are the only maximal rank letters (because $i_p > i_{p+1} > \cdots > i_{p'}$ and $i_q > i_{q-1} > \cdots > i_{q'}$). So z freely reduces to the empty word, which we will show is impossible.

   It will be convenient (for Case 1.2.1) to assume $e_p, e_q < -1$, which we can do because applying θ^{-1} to z gives a type *iv* word of the same form which also freely reduces to the empty word.

1.1. *Case: u is the empty word.* This leads to a contradiction because it implies that the last letter $a_{i_{p'}-1}$ of $\theta^{e_p+1}(a_{i_p})\theta^{e_{p'}}(a_{i_{p'}-1}^{-1})$ and the first letter $a_{i_{q'}-1}$ of

$\theta^{e_{q'}}(a_{i_{q'}-1})\theta^{e_q+1}(a_{i_q}^{-1})$ cancel—that is, $i_{p'} = i_{q'}$, so $\theta^{e_{p'}}(a_{i_{p'}})\theta^{e_{q'}}(a_{i_{q'}}^{-1})$ is a subword of z contrary to the definition of z.

1.2. *Case: u is not the empty word.*

1.2.1. *Case: $p \neq p'$ and $q \neq q'$.* In this case, $i_p, i_q \geq 4$ because of our hypotheses on $i_p, \ldots, i_{p'}, i_{q'}, \ldots, i_q$. Since we assumed $e_p, e_q < -1$, the word in (43) has a subword of the form

$$a_{i_p-1}^{-1}\theta^{e_{p'}}(a_{i_{p'}-1}^{-1})u\theta^{e_{q'}}(a_{i_{q'}-1})a_{i_q-1}, \tag{44}$$

and no cancellation is possible with the prefix of z to its the left or the suffix to its right. The maximal rank letters it contains are its first and last letters, so they must cancel, and therefore

$$\theta^{e_{p'}}(a_{i_{p'}-1}^{-1})u\theta^{e_{q'}}(a_{i_{q'}-1}) \tag{45}$$

must freely equal the empty word.

1.2.1.1. *Case: $i_{p'-1} \neq 2$ or $i_{q'-1} \neq 2$.* Then $i_{p'-1} = i_{q'-1}$ because otherwise (45) has a single letter of highest rank which (either the $a_{i_{p'-1}}^{-1}$ or the $a_{i_{q'-1}}$) and hence cannot freely reduce to the empty word. However, then $a_{i_{p'-1}}^{-1}$ and $a_{i_{q'-1}}$ are the letters of highest rank in (45) and so must cancel. Since u is the subword separating them, u must freely reduce to the empty word, which is impossible by Lemma 4.5.

1.2.1.2. *Case: $i_{p'-1} = i_{q'-1} = 2$.* By Lemma 4.5, u cannot have any rank-2 subwords that freely reduce to the empty word. Since (45) freely reduces to the empty word and u contains no rank-2 subwords that freely reduce to the empty word, by (19) u must be

$$\theta^{e_{p'}-1}(a_2)a_1^{\mu}\theta^{e_{q'}-1}(a_2^{-1})$$

for some $\mu \in \mathbb{Z}$. By counting the exponent sum of a_1 in (45):

$$e_{p'} - (e_{p'} - 1) + \mu + (e_{q'} - 1) - e_{q'} \ = \ 0,$$

so that $\mu = 0$, contradicting the fact that u does not have consecutive principal letters a_2 and a_2^{-1} (by definition of z).

1.2.2. *Case: $p = p'$.* In this case, the word (43) which z freely reduces to has the form

$$\theta^{e_p}(a_{i_p})u\theta^{e_{q'}}(a_{i_{q'}-1})\theta^{e_q+1}(a_{i_q}^{-1}).$$

Recall that the suffix $\theta^{e_{q'}}(a_{i_{q'}-1})\theta^{e_q+1}(a_{i_q}^{-1})$ is freely reduced and so its first letter $a_{i_{q'}-1}$ cannot cancel to its right. So it must cancel to its left, and therefore either $i_{q'} = 3$ or it cancels with the terminal $a_{i_p-1}^{-1}$ of $\theta^{e_p}(a_{i_p})$. In the latter case:

$$i_q - 1 \ = \ i_p - 1 \ = \ i_{q'} - 1,$$

so $i_q = i_{q'}$, and so $q = q'$. Therefore it suffices to analyze the following two cases.

1.2.2.1. *Case: $i_{q'} = 3$ and $q \neq q'$.* Since $q \neq q'$, $i_q > 3$. So $i_q > 3$ also as $i_p = i_q$. Hence (43) has a subword

$$a_{i_p-1}^{-1}u\theta^{e_{q'}}(a_2)a_{i_q-1} \tag{46}$$

whose first letter $a_{i_p-1}^{-1}$ cannot cancel to the left and whose last letter a_{i_q-1} cannot cancel to the right. They have rank at least 3, so they must cancel each other. So $u\theta^{e_{q'}}(a_2)$ freely equals the empty word. But u cannot have any rank 2 subwords that freely equal the empty word by Lemma 4.5, so by (19) is

$$a_1^{\mu}\theta^{e_{q'}-1}(a_2^{-1})$$

for some $\mu \in \mathbb{Z}$. So (46) is

$$a_{i_p-1}^{-1}a_1^{\mu}\theta^{e_{q'}-1}(a_2^{-1})\theta^{e_{q'}}(a_2)a_{i_q-1} \;=\; a_{i_p-1}^{-1}a_1^{\mu}\,(a_2a_1^{e_{q'}-1})^{-1}\,a_2a_1^{e_{q'}}\,a_{i_q-1}.$$

By counting the exponent sum of a_1 it contains, we find

$$\mu-(e_{q'}-1)+e_{q'}=0.$$

So $\mu=-1$. Now

$$u \;=\; a_1^{-1}\theta^{e_{q'}-1}(a_2^{-1}) \;=\; \theta^{e}(a_1^{-1})\theta^{e_{q'}-1}(a_2^{-1})$$

for some $e\in\mathbb{Z}$. So $\theta^{e_p}(a_{i_p})\theta^{e}(a_1^{-1})\theta^{e_{q'}-1}(a_2^{-1})$ is a prefix of z and (19) tells us that $e=e_p$ and $e+1=e_{q'}-1$, and so $e_p+2=e_{q'}$. Now, as $u\theta^{e_{q'}}(a_2)$ freely equals the empty word and $p=p'$, (43) freely reduces to

$$\theta^{e_p+1}(a_{i_p})\theta^{e_{p'}}(a_{i_{p'}-1}^{-1})\theta^{e_q+1}(a_{i_q}^{-1}) \;=\; \theta^{e_p}(a_{i_p})\theta^{e_q+1}(a_{i_q}^{-1}).$$

So, as $i_p=i_q>1$, we find $e_p=e_q+1$. But $e_q\geq e_{q'}$, so this contradicts $e_p+2=e_{q'}$.

1.2.2.2. *Case:* $q=q'$. In this instance,

$$z \;=\; \theta^{e_p}(a_{i_p})u\theta^{e_q}(a_{i_q})$$

freely reduces to the identity. Hence $\theta^{\max(-e_p,-e_q)}(z)$ is a type i word which also freely reduces to the identity, which is impossible by the type i case of Proposition 4.7 proved above.

1.2.3. *Case:* $q=q'$. Inverting z returns us to Case 1.2.2 above.

2. *Case:* $i_p>i_q$. By Proposition 4.7 in type $ii^{\pm1}$, w freely reduces to a word of the form:

$$\theta^{e_p+1}(a_{i_p})\theta^{e_{p'}}(a_{i_p-1}^{-1})u\theta^{e_{q'}}(a_{i_{q'}-1})\theta^{e_q+1}(a_{i_q}^{-1}).$$

Observe that a_{i_q} cannot be cancelled because $a_{i_q}^{-1}$ does not appear. To cancel $a_{i_q}^{-1}$, since $i_q\geq 3$ and u is rank 2, $a_{i_q}^{-1}$ must cancel with a letter to the left of u, since it is the only rank i_q letter appearing to the right of u. Also, $a_{i_{p'}-1}^{-1}$, the final letter of $\theta^{e_{p'}}(a_{i_{p'}})$ is an obstruction to cancelling a_{i_q} with any letter from $\theta^{e_p+1}(a_{i_p})$ and $a_{i_{p'}-1}^{-1}$ and has rank at least i_q. Thus the only letters of rank i_p-1 in w come from $\theta^{e_p+1}(a_{i_p})$, so every letter of rank i_p-1 has exponent -1. To cancel $a_{i_q}^{-1}$ with a letter from $\theta^{e_p+1}(a_{i_p})$ requires cancelling the rightmost $a_{i_p-1}^{-1}$ from $\theta^{e_p+1}(a_{i_p})$ which is impossible.

Similarly, if $a_{i_q}^{-1}$ cancels with a letter from $\theta^{e_{p'}}(a_{i_{p'}-1}^{-1})$, the rightmost letter of $\theta^{e_{p'}}(a_{i_{p'}-1}^{-1})$, which is $a_{i_{p'}-1}^{-1}$, must cancel too. By Proposition 4.7 in type $ii^{\pm1}$, $\theta^{e_p+1}(a_{i_p})\theta^{e_p}(a_{i_{p'}-1}^{-1})$ is freely reduced, so its rightmost $a_{i_{p'}-1}^{-1}$ must cancel to the right. However, $a_{i_{p'}-1}^{-1}$ is the highest rank letter in $\theta^{e_p}(a_{i_p-1})^{-1}$, so $e_{p'}-1\geq i_q$. Also $i_{p'}-1\leq i_q$ because $a_{i_{p'}-1}^{-1}$ can only cancel with an $a_{i_{p'}-1}$. We cannot cancel $a_{i_{p'}-1}^{-1}$ from $\theta^{e_{p'}}(a_{i_{p'}-1}^{-1})$ because then $a_{i_q}^{-1}$ would be the only other letter of the same rank. Thus it is impossible to cancel $a_{i_q}^{-1}$.

3. *Case:* $i_p<i_q$. Invert w and apply the argument from Case 2. □

Proof of Proposition 4.7 in type v. We have

$$z \;=\; \theta^{e_p}(a_{i_p}^{\epsilon_p})\cdots\theta^{e_q}(a_{i_q}^{\epsilon_q})$$

and no type *i*–*iv* subword $\hat{z}$ of w overlaps with z. More precisely, there is no $0 \le p' < q' \le l+1$ with $p \le q' \le q$ such that $\theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{p'}})\cdots\theta^{e_{q'}}(a_{i_{q'}}^{\epsilon_{q'}})$ is of type *i*–*iv*. The claim is that free reduction of z to z' removes no letters of rank 3 or higher. Moreover, if $\epsilon_p = 1$, $i_p \ge 3$ and $e_p > 0$, then z' (the reduced form of z) has prefix $\theta^{e_p-1}(a_{i_p})$.

Here is our proof of the first claim. Suppose, for a contradiction, that some letter a_α^ϵ (not necessarily principal) in z with $\alpha \ge 3$ and $\epsilon = \pm 1$ cancels with some $a_\alpha^{-\epsilon}$ to its right when z is freely reduced.

Then z has a subword $a_\alpha^\epsilon v a_\alpha^{-\epsilon}$ which freely equals the empty word. Since $\alpha \ge 3$, we know that a_α comes from some $\theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{p'}})$ where $i_{p'} \ge 3$ while a_α^{-1} comes from some $\theta^{e_{q'}}(a_{i_{q'}}^{\epsilon_{q'}})$ where $i_{q'} \ge 3$. Note that $p' \ne q'$ because otherwise $a_\alpha^\epsilon v a_\alpha^{-\epsilon}$ would be a subword of $\theta^{e_{p'}}(a_{i_{p'}})$, which is freely reduced. We may assume that v contains no letter a_β^δ with $\beta \ge 3$ and $\delta \in \{\pm 1\}$ that cancels to its right with an $a_\beta^{-\delta}$ in v, because otherwise we could replace our original choice of $a_\alpha^\epsilon v a_\alpha^{-\epsilon}$ with a shorter subword $a_\beta^\delta \cdots a_\beta^{-\delta}$. So $\mathrm{rank}(v) \le 2$, and z has a subword

$$\theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{p'}})u\theta^{e_{q'}}(a_{i_{q'}}^{\epsilon_{q'}}) \tag{47}$$

where u is either empty or $\mathrm{rank}(u) \le 2$.

1. *Case:* $\epsilon_{p'} = 1$ *and* $\epsilon_{q'} = -1$. In this case, (47) is type either *i*, or $iii^{\pm 1}$, or *iv* contrary to the hypothesis that z is type *v*.
2. *Case:* $\epsilon_{p'} = 1$ *and* $\epsilon_{q'} = 1$. For $a_\alpha^{-\epsilon}$ is to cancel, the $a_{i_{q'}}$ at the start of $\theta^{e_{q'}}(a_{i_{q'}}^{\epsilon_{q'}})$ must cancel to its left. If $e_p \ge 0$, then $\theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{q'}})$ is a positive word, so the only letters to the left of $a_{i_{q'}}$ with exponent -1 have lower rank, and such cancellation is not possible. If $e_p < 0$, then the last letter of $\theta^{e_{p'}}(a_{i_{p'}})$ is $a_{i_{p'}-1}^{-1}$, so either $i_{p'} - 1 = 2$ or (u is the empty word and $i_{q'} = i_{p'-1}$). In the former case: $\alpha = 3$, but then $a_\alpha^\epsilon v a_\alpha^{-\epsilon}$ cannot freely equal the empty word because $a_\alpha^\epsilon = a_\alpha$ cannot cancel with the first letter $a_{i_{q'}}$ of $\theta^{e_{q'}}(a_{i_{q'}})$. In the latter case: by (19), $e_{q'} = e_{p'} - 1 < 0$, so we have a type *ii* subword contained in z, contrary to the definition of a type *v* subword.
3. *Case:* $\epsilon_{p'} = -1$ *and* $\epsilon_{q'} = -1$. Invert and apply the previous case to obtain a contradiction.
4. *Case:* $\epsilon_{p'} = -1$ *and* $\epsilon_{q'} = 1$. In this case (47) has subword

$$a_{i_{p'}}^{-1} u a_{i_{q'}}$$

where $a_{i_{p'}}^{-1}$ does not cancel to the left and $a_{i_{q'}}$ does not cancel to the right, which makes a contradiction because these letters both have rank higher than 2.

So the first claim is proved.

The second claim—if $\epsilon_p = 1$, $i_p \ge 3$ and $e_p > 0$, then z' has prefix $\theta^{e_p-1}(a_{i_p})$—is proved exactly as per the final paragraph of Case 2.2.2 of our proof above Proposition 4.7 in case *iii*. □

4.4. **The Piece Criterion.** The *Piece Criterion* is the main technical result behind the correctness of our algorithm $\mathtt{Member}_k$. Before we state it, we establish two preliminary propositions. The first is used in the proof of the second, and the second provides a key step of our proof of the Piece Criterion. In both we refer to a *reduced word h on* $(a_1 t)^{\pm 1}, \ldots, (a_k t)^{\pm 1}$, which is to say that h contains no subwords $(a_i t)^{\pm 1}(a_i t)^{\mp 1}$.

Proposition 4.8. *Suppose $u = u(a_1, \ldots, a_{m-1})$ is freely reduced and non-empty, $h = h(a_1t, \ldots, a_kt)$ is freely reduced, $r, s \in \mathbb{Z}$, and $2 \leq m \leq k$. In G_k,*

$$\begin{array}{rcl} (t^r a_m u = ht^s \text{ or } t^r a_m u a_m^{-1} = ht^s) & \Longrightarrow & \text{the first letter of } h \text{ is } (a_m t), \\ (t^r u a_m^{-1} = ht^s \text{ or } t^r a_m u a_m^{-1} = ht^s) & \Longrightarrow & \text{the final letter of } h \text{ is } (a_m t)^{-1}. \end{array}$$

Proof. The second statement follows from the first as can be seen by inverting both sides of the equalities and then rearranging so as to interchange the roles of r and s.

We will prove the first statement in the case $t^r a_m u = ht^s$ only, as the case $t^r a_m u a_m^{-1} = ht^s$ can be proved in essentially the same way.

So assume $t^r a_m u = ht^s$, and so $a_m u = t^{-r} h t^s$, in G_k. Consider carrying all the $t^{\pm 1}$ in $t^{-r} h t^s$ from left to right through the word, with the effect of applying $\theta^{\mp 1}$ to the intervening letters $a_i^{\pm 1}$, and then freely reducing, so as to arrive at $a_m u$.

We will first argue that h contains no $(a_{m+1}t)^{\pm 1}, \ldots, (a_k t)^{\pm 1}$. Suppose otherwise. Let i be maximal such that h contains an $(a_i t)^{\pm 1}$. As carrying all the $t^{\pm 1}$ to the right and cancelling gives $a_m u$, there must be an $(a_i t)^{\mp 1}$ in h so that there is an $a_i^{\mp 1}$ to cancel with the $a_i^{\pm 1}$ in our $(a_i t)^{\pm 1}$—this is because applying $\theta^{\pm 1}$ to $a_1^{\pm 1}, \ldots, a_i^{\pm 1}$, neither creates nor destroys any $a_i^{\pm 1}$. But then if h' is the subword of h that has first and last (or last and first) letters these $(a_i t)^{\pm 1}$ and $(a_i t)^{\mp 1}$, then $t^{r'} h' = t^{s'}$ for some $r', s' \in \mathbb{Z}$. That then implies that $h' \in \langle t \rangle$. But $H_k \cap \langle t \rangle = \{1\}$ by Lemma 6.1 of [12], so $h = 1$ in G_k. But $H_k = F(a_1 t, \ldots, a_k t)$ by Proposition 4.1 of [12], and so our assumption that h is freely reduced is contradicted.

Next notice that there must be an $(a_m t)$ in h because $a_m u$ contains an a_m and applying $\theta^{\pm 1}$ to $a_1^{\pm 1}, \ldots, a_m^{\pm 1}$ neither creates nor destroys any $a_m^{\pm 1}$. Suppose, for a contradiction, that the first $(a_m t)$ in h is not at the front. Express h as $\alpha (a_m t) \beta$ where $\alpha = \alpha(a_1 t, \ldots, a_{m-1} t)$ is non-empty.

We claim that the a_m of the first $(a_m t)$ in h must cancel with some subsequent a_m^{-1}. Suppose otherwise. We have that

$$t^{-r} h t^s \;=\; t^{-r} \alpha (a_m t) \beta t^s \;=\; v t^j (a_m t) \beta t^s$$

for some $v = v(a_1, \ldots, a_{m-1})$ and some $j \in \mathbb{Z}$. But then $v = 1$ as the first a_m serves as a barrier to cancelling away v when the remaining $t^{\pm 1}$ are carried to the right: applying $\theta^{\pm 1}$ to a_m only produces new letters $a_1^{\pm 1}, \ldots, a_{m-1}^{\pm 1}$ (see Lemma 7.1 in [12]) to its right, and (by assumption) it is not cancelled away by a subsequent a_m^{-1}. But then $\alpha \in \langle t \rangle$, leading to a contradiction as before.

Now, if a_m of the first $(a_m t)$ in h cancel with some subsequent a_m^{-1}, by the same argument as earlier, the subword bookended by that $(a_m t)$ and $(a_m t)^{-1}$ must freely reduce to the empty word, contradicting the assumption that h is freely reduced. □

To follow the details of the following proof it will help to have a copy of Definition 4.6 and Proposition 4.7 to hand.

Proposition 4.9. *Suppose $u = u(a_1, \ldots, a_{m-1})$ is freely reduced, $h = h(a_1 t, \ldots, a_k t)$ is freely reduced, $r, s \in \mathbb{Z}$, $3 \leq m \leq k$, and $t^r a_m u = ht^s$ or $t^r a_m u a_m^{-1} = ht^s$ in G_k. If $r > 0$, then $\theta^{r-1}(a_m)$ is a prefix of $a_m u$.*

Proof. We will prove the case where $t^r a_m u a_m^{-1} = ht^s$ in G_k. The proof for the case $t^r a_m u = ht^s$ is the same.

Proposition 4.8 tells us that the first and last letters of h are $(a_m t)$ and $(a_m t)^{-1}$, respectively. Express h as $(a_{i_0}t)^{\epsilon_0}\cdots(a_{i_{j+1}}t)^{\epsilon_{j+1}}$ where $\epsilon_0 = 1$, and $\epsilon_1,\ldots,\epsilon_j = \pm1$, and $\epsilon_{j+1} = -1$, and $i_0 = i_{j+1} = m$, and $i_1,\ldots,i_j \in \{1,\ldots,m-1\}$.

If we shuffle all the $t^{\pm1}$ in $t^{-r}ht^s$ to the right, then the power of t emerging on the right cancels away since $t^{-r}ht^s$ equals $a_m u a_m^{-1}$ and $u = u(a_1,\ldots,a_{m-1})$ in G_k, and we get

$$\pi \ := \ a_m u a_m^{-1} \ = \ \theta^{e_0}(a_{i_0}^{\epsilon_0})\cdots\theta^{e_j}(a_{i_j}^{\epsilon_j})\theta^{e_{j+1}}(a_{i_{j+1}}^{\epsilon_{j+1}})$$

where e_l is, for $0 \le l \le j+1$, the exponent sum of the $t^{\pm1}$ in h that precede a_{i_l} in $t^{-r}ht^s a_m$ (which includes the t^{-1} of $(a_{i_l}t)^{\epsilon_l}$ if $\epsilon_l = -1$):

$$e_l \ = \ \begin{cases} r + \epsilon_1 + \cdots + \epsilon_{l-1} & \text{if } \ \epsilon_l = 1 \\ r + 1 + \epsilon_1 + \cdots + \epsilon_{l-1} & \text{if } \ \epsilon_l = -1. \end{cases}$$

Also $a_{i_x}^{\epsilon_x} \neq a_{i_{x+1}}^{-\epsilon_{x+1}}$ for $x = 0,\ldots,j$ because h is freely reduced as a word on $(a_1t)^{\pm1},\ldots,(a_kt)^{\pm1}$. So, π is of the form in which it appears in Definition 4.6.

We will work right to left through z choosing subwords z_1, z_2, ... until we have π expressed as a concatenation $z_l\cdots z_2z_1$. Define $\pi_1 := \pi$ and define z_1 to be the maximal length suffix of π_1 of one of the five types of Definition 4.6. (Such a suffix exists if π_1 is non-empty, as there must be a type v suffix if no other type.) Let π_2 be π_1 with the suffix z_1 removed, and then define z_2 to be the maximal length suffix of π_2 of one of the five types of Definition 4.6. Continue likewise until z is exhausted and we have $\pi = z_l\cdots z_2z_1$.

Let $\pi', z_1',\ldots,z_l'$ denote the freely reduced forms of $\pi, z_1,\ldots,z_l$, respectively. We will use Proposition 4.7 to argue that $\pi' = z_l'\cdots z_2'z_1'$. In other words, when freely reducing π, all cancellation is within the z_i—none occurs between a z_{i+1} and the neighboring z_i.

Given how Proposition 4.7 identifies the first and last letters of each z_i' when of type *i*–*iv*, and given that $a_{i_x}^{\epsilon_x} \neq a_{i_{x+1}}^{-\epsilon_{x+1}}$ for $x = 0,\ldots,j$, cancellation between z_{i+1}' and z_i' is ruled out except in these four situations:

- z_i is of type ii^{-1},
- z_{i+1} is of type *ii*,
- z_i is of type *v*,
- z_{i+1} is of type *v*.

We will explain why these too do not give rise to cancellation. Express z_{i+1} and z_i as:

$$z_{i+1} \ = \ \theta^{e_p}(a_{i_p}^{\epsilon_p})\cdots\theta^{e_q}(a_{i_q}^{\epsilon_q}) \qquad \text{and} \qquad z_i \ = \ \theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{p'}})\cdots\theta^{e_{q'}}(a_{i_{q'}}^{\epsilon_{q'}}).$$

(So $p' = q+1$.)

Case: z_{i+1} not type v, z_i type ii^{-1}. The first letter of z_i' is $a_{i_{p'}-1}$ by Proposition 4.7 in type ii^{-1}. If z_{i+1} is of type *ii*, then the final letter of z_{i+1}' is $a_{i_{p'-1}-1}^{-1}$ (remember $p'-1 = q$) which cannot cancel with the $a_{i_{p'}-1}$ at the start of z_i' since $a_{i_{p'-1}}^{\epsilon_{p'-1}}$ and $a_{i_{p'}}^{\epsilon_{p'}}$ are not mutual inverses and $\epsilon_{p'-1} = 1$ and $\epsilon_{p'} = -1$. If z_{i+1} is of type *i*, ii^{-1}, $iii^{\pm1}$, or *iv*, then the final letter of z_{i+1} is $a_{i_{p'-1}}^{-1}$ which cannot be $a_{i_{p'}-1}^{-1}$ as that would contradict the maximality of z_i: prepending $\theta^{e_p'}(a_{i_{p'-1}}^{\epsilon_{p'-1}})$ to z_i would give a longer type ii^{-1} word.

Case: z_{i+1} type ii, z_i not type v. Similarly, there can be no cancellation between z_{i+1}' and z_i'. In the cases where z_i is of type *i*, *ii*, $iii^{\pm1}$, or iv appending $\theta^{e_{p+1}}(a_{i_{p+1}}^{\epsilon_{p+1}})$ to z_{i+1} would give a longer type *ii* word, contradicting the definition of z_i as a type *v* word.

Case: z_{i+1} not type ii, z_i type v. Then z_{i+1} cannot be of type *v*, else $z_{i+1}z_i$ would be of type *v* contrary to maximality of z_i. So z_{i+1} is of type *i*, ii^{-1}, $iii^{\pm1}$ or *iv*, and therefore $i_q \ge 3$ and

$\epsilon_q = -1$, and by Proposition 4.7, the final letter of z'_{i+1} is $a_{i_q}^{-1}$. So if there is cancellation between z'_{i+1} and z'_i, then the first letter of z'_i must be a_{i_q}. But then, there is a subword

$$\pi'' \ := \ \theta^{e_q}(a_{i_q}^{-1})\theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{p'}})\cdots\theta^{e_m}(a_{i_m}^{\epsilon_m})$$

of $z_{i+1}z_i$ such that the $a_{i_q}^{-1}$ in $\theta^{e_q}(a_{i_q}^{-1})$ cancels with some a_{i_q} in $\theta^{e_m}(a_{i_m}^{\epsilon_m})$ on free reduction and $i_{p'}, \cdots, i_{m-1} \leq 2$—otherwise there would be some intervening letter of rank at least 3 which would have to cancel away on freely reducing this subword and hence on freely reducing z_i, contrary to Proposition 4.7 in type *v*.

Suppose $\epsilon_m = 1$. Then $\theta^{e_m}(a_{i_m}^{\epsilon_m})$ is a_{i_m} times a word on lower rank letters. So, as the $a_{i_q}^{-1}$ in $\theta^{e_q}(a_{i_q}^{-1})$ cancels away when π'' is freely reduced, $a_{i_m}^{\epsilon_m} = a_{i_q}$. But then the intervening subword $\theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{p'}})\cdots\theta^{e_{m-1}}(a_{i_{m-1}}^{\epsilon_{m-1}})$ has rank at most 2 and freely reduces to the empty word, and so is empty by Lemma 4.5. So $p' = m$ and, as $a_{i_m}^{\epsilon_m} = a_{i_q}$, that contradicts the $x = q$ instance of $a_{i_x}^{\epsilon_x} \neq a_{i_{x+1}}^{-\epsilon_{x+1}}$.

Suppose, on the other hand, $\epsilon_m = -1$. If $e_m \geq 0$, then $\theta^{e_m}(a_{i_m}^{-1})$ contains no positive letters and so cannot supply a letter to cancel with $a_{i_q}^{-1}$. If $e_m < 0$ and $i_m = 3$, then the only letter in $\theta^{e_m}(a_{i_m}^{-1})$ of rank at least three is a single a_3^{-1}, and that cannot cancel with $a_{i_q}^{-1}$. If $e_m < 0$ and $i_m > 3$, then the first letter of $\theta^{e_m}(a_{i_m}^{\epsilon_m})$ is a_{i_m-1} (Lemma 4.4) and this could only cancel with the $a_{i_q}^{-1}$ were the intervening subword $\theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{p'}})\cdots\theta^{e_{m-1}}(a_{i_{m-1}}^{\epsilon_{m-1}})$ empty (as before) and $p' = m = q+1$, but in that case z_i has prefix $\theta^{e_{p'}}(a_{i_{p'}}^{\epsilon_{p'}}) = \theta^{e_{q+1}}(a_{i_q+1}^{-1})$, violating the definition of a type *v* subword because $\theta^{e_q}(a_{i_q}^{-1})\theta^{e_{q+1}}(a_{i_q+1}^{-1})$ is type ii^{-1}.

Case: z_{i+1} type v, z_i not type ii^{-1}. As in the previous case, z_i cannot be of type *v*, so z_i is type *i*, *ii*, *iii*$^{\pm 1}$ or *iv* and $i_{q+1} \geq 3$. The same arguments as the previous case apply to tell us that cancellation is impossible. The final case concludes with the maximality of the type *i*, *ii*$^{-1}$, *iii*$^{\pm 1}$ or *iv* word z_i being contradicted.

Case: z_{i+1} type v, z_i type ii^{-1}. We have that

$$z_{i+1} \ = \ \theta^{e_p}(a_{i_p}^{\epsilon_p})\cdots\theta^{e_q}(a_{i_q}^{\epsilon_q}) \qquad \text{and} \qquad z'_i \ = \ \theta^{e_{p'}}(a_{i_{p'}-1})\theta^{e_{q'}+1}(a_{i_{q'}}^{-1})$$

by definition and by Proposition 4.7 in type *i*, respectively, and $e_{q'} < 0$, $i_{q'} \geq 3$, and $i_{p'} \geq 2$. Moreover, the first letter of z'_i is $a_{i_{p'}-1}$ by Proposition 4.7 in type *ii*. Suppose $i_{p'}$ is 2 or 3. Then z_{i+1} has suffix $\theta^{e_q}(a_{i_q}^{\epsilon_q}) = \theta^{e_q}(a_{i_{p'}-1}^{-1})$ or something of rank at most 2 which could be prepended to z_i contradicting its maximality. Suppose, on the other hand, $i_{p'} > 3$. If there is cancellation between z'_{i+1} and z'_i, then a letter of rank at least 3 in z_{i+1} cancels with the first letter $a_{i_{p'}-1}$ of z'_i. As in the preceding cases, conclude that $a_{i_q}^{\epsilon_q}$ must cancel with the first letter of z'_i, so $i_q = i_{p-1}$ and $\epsilon_q = -1$, contradicting maximality of z_i.

Case: z_{i+1} type ii, z_i type v. This case is essentially the same as the preceding one. Follow the steps from the previous case, except instead of appealing to maximality of z_{i+1}^{-1}, observe that the last letter of z_{i+1} and z_i form a type *ii* subword which is forbidden by the definition of a type *v* subword.

Having established that there is no cancellation between z'_{i+1} and z'_i for $i = 1, \ldots, l-1$, all that remains is to argue that $a_m z'_l$ has prefix $\theta^{r-1}(a_m)$, for it will then follow that $a_m\pi'$ has the same prefix.

But z_l is type *i*, *iii* or *v* because $e_0 = r > 0$. It has prefix $\theta^{e_0}(a_{i_0}^{\epsilon_0}) = \theta^r(a_m)$ and $r > 0$, so as $i_0 = m \geq 3$, Proposition 4.7 in types *i*, *iii* and *v*, tells us that $\theta^{r-1}(a_m)$ is a prefix of z'_l, and hence of $\pi = a_m u$. □

We are now ready for the Piece Criterion. It concerns only the case where the rank (denoted by m) is at least 3. In the cases $m = 1$ and $m = 2$ our algorithms are straightforward and the Piece Criterion is not required to prove correctness.

Proposition 4.10 (The Piece Criterion)**.** *Suppose $m \geq 3$ and $r \in \mathbb{Z}$, and suppose $\pi = a_m^{\epsilon_1} u a_m^{-\epsilon_2}$ is a freely reduced word such that $u = u(a_1, \ldots, a_{m-1})$ and $\epsilon_1, \epsilon_2 \in \{0, 1\}$. Define*

$$x_l := a_m^{-1}\theta^l(a_m) \quad \text{for } l \in \mathbb{Z},$$

$$x := \begin{cases} x_r & \text{if } r > 0 \text{ and } \epsilon_1 = 1 \\ \text{empty word} & \text{otherwise,} \end{cases}$$

$$\delta := \begin{cases} r & \text{if } \epsilon_1 = 0 \\ \psi_m(r) & \text{if } \epsilon_1 = 1 \text{ and } r \leq 0 \\ r - 1 & \text{if } \epsilon_1 = 1 \text{ and } r > 0. \end{cases}$$

Suppose $s \in \mathbb{Z}$. Let π' be the freely reduced form of $x^{-\epsilon_1} u a_m^{-\epsilon_2}$. Consider the following conditions.

(*i*) $\epsilon_1 = 0$.
(*ii*) $\epsilon_1 = 1$ *and* $r \leq 0$.
(*iii*) $\epsilon_1 = 1$, $r > 0$ *and* $\theta^{r-1}(a_m)$ *is a prefix of* π.

(*a*) $\epsilon_2 = 0$ *and* $t^\delta x^{-\epsilon_1} u \in H_k t^s$.
(*b*) $\epsilon_2 = 1$, $s \leq 0$ *and* $t^\delta x^{-\epsilon_1} u \in H_k t^{\psi_m(s)}$.
(*c*) $\epsilon_2 = 1$, $s > 0$ *and* $t^\delta x^{-\epsilon_1} u x_s \in H_k t^{s-1}$ *and* $\theta^{s-1}(a_m^{-1})$ *is a suffix of* π.

We have $t^r\pi \in H_k t^s$ if and only if ((i, ii or iii) and $t^\delta \pi' \in H_k t^s$). Moreover, $t^\delta \pi' \in H_k t^s$ if and only if (a, b or c).

Proof. Suppose $s \in \mathbb{Z}$. First suppose that $t^r \pi \in Ht^s$. Then (*i*, *ii* or *iii*) holds because if $\epsilon_1 = 1$ and $r > 0$, then $\theta^{r-1}(a_m)$ is a prefix of π by Proposition 4.9. So $t^\delta x^{-\epsilon_1} u a_m^{-\epsilon_2} \in H_k t^s$ for the same $s \in \mathbb{Z}$.

Next we will prove that $t^r \pi \in Ht^s$ is equivalent to $t^\delta \pi' \in H_k t^s$ under the assumption that (*i*, *ii* or *iii*) holds.

Under *i*, $\epsilon_1 = 0$, x is the empty word, and $\delta = r$. So $t^\delta \pi' = t^\delta x^{-\epsilon_1} u a_m^{-\epsilon_2} = t^r u a_m^{-\epsilon_2} = t^r \pi$ and the equivalence is immediate.

Under *ii*, $\epsilon_1 = 1$, $r \leq 0$, x is the empty word, and $\delta = \psi_m(r)$. So $t^\delta \pi' = t^\delta x^{-\epsilon_1} u a_m^{-\epsilon_2} = t^{\psi_m(r)} u a_m^{-\epsilon_2}$, giving the third of the following equivalences. The first equivalence holds simply because $\pi = a_m u a_m^{-\epsilon_2}$. For the second, r is in the domain of ψ_m because $r \leq 0$, so $t^r a_m \in H_k t^{\psi_m(r)}$ by Proposition 3.1, and so $t^{\psi_m(r)} a_m^{-1} t^{-r} \in H_k$.

$$\begin{aligned} & t^r \pi \in H_k t^s \\ & \quad \Leftrightarrow \; t^r a_m u a_m^{-\epsilon_2} \in H_k t^s \\ & \quad \Leftrightarrow \; t^{\psi_m(r)} u a_m^{-\epsilon_2} \in H_k t^s \\ & \quad \Leftrightarrow \; t^\delta \pi' \in H_k t^s. \end{aligned}$$

Under *iii*, $\epsilon_1 = 1$, $r > 0$, $x = x_r$, and $\delta = r - 1$. Observe that

$$t^\delta \pi' = t^{r-1} x_r^{-1} u a_m^{-\epsilon_2} \in H_k t^s \;\Leftrightarrow\; t^r \pi = t^r a_m u a_m^{-\epsilon_2} \in H_k t^s$$

because $t^{r-1} x_r^{-1} a_m^{-1} t^{-r} = t^{r-1}\theta^r(a_m^{-1}) t^{-r} = (a_m t)^{-1} \in H_k$.

So, assuming (*i*, *ii* or *iii*) holds, $t^r \pi \in H_k t^s$ if and only if $t^\delta \pi' \in H_k t^s$, as required.

Next we will prove that $t^\delta \pi' \in H_k t^s$ if and only if (*a*, *b* or *c*) holds.

Suppose $\epsilon_2 = 0$. Then $t^\delta \pi' = t^\delta x^{-\epsilon_1} u a_m^{-\epsilon_2} = t^\delta x^{-\epsilon_1} u$ and so $t^\delta \pi' \in H_k t^s$ is the same as Condition *a*.

Suppose, on the other hand, that $\epsilon_2 = 1$. Suppose further that $s \leq 0$. Proposition 3.1 tells us that $t^s a_m \in H_k t^{\psi_m(s)}$ since $s \leq 0$ and so is in the domain of ψ_m. So $t^\delta \pi' = t^\delta x^{-\epsilon_1} u a_m^{-1} \in H_k t^s$ if and only if $t^\delta x^{-\epsilon_1} u \in H_k t^{\psi_m(s)}$. So $t^\delta \pi' \in H_k t^s$ is equivalent to Condition *b*.

Finally, observe that

$$\begin{aligned} & t^\delta \pi' = t^\delta x^{-\epsilon_1} u a_m^{-1} \in H_k t^s \\ \Leftrightarrow\ & t^\delta x^{-\epsilon_1} u a_m^{-1} t^{-s} \in H_k \\ \Leftrightarrow\ & t^\delta x^{-\epsilon_1} u a_m^{-1} t^{-s} (t^s a_m x_s t^{-(s-1)}) \in H_k \\ \Leftrightarrow\ & t^\delta x^{-\epsilon_1} u x_s \in H_k t^{s-1} \end{aligned}$$

because $t^s a_m x_s t^{-(s-1)} = a_m t \in \mathcal{H}_k$. Suppose now that $s > 0$. The part of Condition *c* concerning the suffix of π follows from Proposition 4.9 (applied to h^{-1}). So $t^\delta \pi' \in H_k t^s$ is equivalent to Condition *c*.

We conclude that $t^r \pi \in H t^s$ implies (*i*, *ii*, or *iii*) and (*a*, *b*, or *c*). □

4.5. **Our algorithm in detail.** Here we construct $\mathtt{Member}_k$, where k is, as usual, any integer greater than or equal to 1, and is kept fixed. $\mathtt{Member}_k$ inputs a word $w = w(a_1, \ldots, a_k, t)$ and declares whether or not w represents an element of H_k.

Most of the workings of $\mathtt{Member}_k$ are contained in a subroutine $\mathtt{Push}_k$, which inputs a valid ψ-word f and a reduced word $v = v(a_1, \ldots, a_k)$, and declares whether or not $t^{f(0)} v \in H_k t^s$ for some $s \in \mathbb{Z}$ and, if so, returns a ψ-word f' with $s = f'(0)$. (If such an s exists, it is unique by Lemma 6.1 in [12].) The key subroutine for $\mathtt{Push}_k$ when $k \geq 2$ is $\mathtt{Piece}_k$ which handles the special case in which w is a rank-m piece. $\mathtt{Piece}_k$ calls a subroutine $\mathtt{Back}_k$, which in turn calls a subroutine $\mathtt{Push}_{k-1}$. So the construction of these three families of subroutines is inductive.

Additionally, subroutines $\mathtt{Prefix}_m$, and $\mathtt{Front}_m$ (where $3 \leq m \leq k$) are used. These do not require an inductive construction, so we will give them first. The designs of $\mathtt{Prefix}_m$, $\mathtt{Front}_m$ (and also $\mathtt{Back}_m$) are motivated by the Piece Criterion (Proposition 4.10).

Algorithm 4.1 — $\mathtt{Prefix}_m$, $m \geq 3$.
- Input a rank-m piece $\pi = a_m u a_m^{-\epsilon_2}$ (so, $u = u(a_1, \ldots, a_{m-1})$ is reduced and $\epsilon_2 \in \{0, 1\}$).
- Return the largest integer $i > 0$ (if any) such that $\theta^{i-1}(a_m)$ is a prefix of π.
- Halt in time $O(\ell(\pi)^2)$.

construct $\theta^{i-1}(a_m)$ for $i = 1, 2, \ldots$ until $\ell(\theta^{i-1}(a_m)) > \ell(\pi)$, and **compare** to π
return the maximum i encountered (if any) such that $\theta^{i-1}(a_m)$ is a prefix of π

Correctness of $\mathtt{Prefix}_m$. As $\ell(\theta^{i-1}(a_m)) \geq i$ for $i = 1, 2, \ldots$, the algorithm returns the appropriate i in time $O(\ell(\pi)^2)$. □

$\mathtt{Front}_m$ takes a rank-m piece π and ψ-word f and reduces the task of determining whether $t^{f(0)} \pi \in H t^s$ to performing a similar determination: specifically whether $t^{f'(0)} \pi' \in H t^s$ where $f'(0) = \delta$ and π' and δ are as per the Piece Criterion. This will represent progress because π' is a piece of rank-m that does not begin with a_m, and because we are able to give good bounds on $\ell(\pi')$ and $\ell(f')$.

Algorithm 4.2 — $\texttt{Front}_m$, $m \geq 3$.

∘ Input a rank-m piece $\pi = a_m^{\epsilon_1} u a_m^{-\epsilon_2}$ with $\epsilon_1, \epsilon_2 \in \{0, 1\}$, and a valid ψ-word $f = f(\psi_1, \ldots, \psi_k)$. Let $r := f(0)$.

∘ Declare whether or not (*i*, *ii* or *iii*) of the Piece Criterion holds. If so, output π' of the Criterion and a valid ψ-word $f' = f'(\psi_1, \ldots, \psi_k)$ such that $f'(0)$ equals δ of the Criterion. These satisfy $\ell(\pi') \leq \ell(\pi)$ and $\ell(f') \leq \ell(f) + 1$, and $t^r \pi \in H_k t^s$ if and only if $t^{f'(0)} \pi' \in H_k t^s$.

∘ Halt in time $O((\ell(w) + \ell(f))^{k+4})$.

if $\epsilon_1 = 0$ (so *i* holds), **output** $\pi' := u a_m^{-\epsilon_2}$ and $f' := f$, and **halt**
run $\texttt{Psi}(f)$ to determine whether or not $r \leq 0$
3: **if** $\epsilon_1 = 1$ and $r \leq 0$ (so *ii* holds), **output** $\pi' := u a_m^{-\epsilon_2}$ and $f' := \psi_m f$, and **halt**

we now have that $\epsilon_1 = 1$ and $r > 0$ (so *i* and *ii* both fail, and it remains to test *iii*)
6: **run** $\texttt{Prefix}_m$ on π
  if it fails to return an i **declare** that *i*, *ii* and *iii* all fail and **halt**
  else it returns some some i
9: **run** $\texttt{Psi}$ on input $\psi_1^i f$ to check whether $i < r$
  if $i < r$, **then declare** that *i*, *ii* and *iii* all fail
  else *iii* holds, so **return** the reduced form π' of $\theta^r(a_m^{-1})\pi$ and $f' := \psi_1 f$

Correctness of $\texttt{Front}_m$.

2: In was established in Section 3.3 that $\texttt{Psi}$ on input f halts in time $O(\ell(f)^{k+4})$.
5: Whether *iii* holds depends on whether $\theta^{r-1}(a_m)$ is a prefix of π, so that is what the remainder of the algorithm examines.
6: $\texttt{Prefix}_m$ halts in time $O(\ell(\pi)^2)$.
9: At this point we know that $\theta^{i-1}(a_m)$ is a prefix of π, and so $i \leq \ell(\pi)$. Therefore, $\ell(\psi_1^i f) \leq \ell(\pi) + \ell(f)$, and so, by the bounds established in Section 3.3, $\texttt{Psi}$ halts in time $O((\ell(\pi) + \ell(f))^{k+4})$.
11: For all $0 \leq p \leq q$, $\theta^p(a_m)$ is a prefix of $\theta^q(a_m)$: after all, for $q \geq 0$, $\theta^{q+1}(a_m) = \theta^q(a_m)\theta^q(a_{m-1})$. So, given that we know at this point that $\theta^{i-1}(a_m)$ is a prefix of π and $r \leq i$, it is the case that $\theta^{r-1}(a_m)$ is also a prefix of π. Note that $\theta^r(a_m^{-1})\pi$ is $\theta^{r-1}(a_{m-1}^{-1}) u a_m^{-\epsilon_2}$ of the Criterion when *iii* holds.

In lines 1, 3 and 11, the claimed bound $\ell(f') \leq \ell(f) + 1$ is immediate, as is $\ell(\pi') \leq \ell(\pi)$ in lines 1 and 3. In line 11, π' is the reduced form of $\theta^r(a_m^{-1})\pi$ and $\theta^{r-1}(a_m)$ is a prefix of π. Now $\theta^r(a_m^{-1}) = \theta^{r-1}(a_{m-1}^{-1})\theta^{r-1}(a_m^{-1})$ and the length of $\theta^{r-1}(a_m^{-1})$ is at least half that of $\theta^r(a_m^{-1})$ (as $r > 0$), and the last letter of $\theta^{r-1}(a_{m-1}^{-1})$ is a_{m-1}^{-1}. So all of the prefix $\theta^{r-1}(a_m)$ of π is cancelled away when $\theta^r(a_m^{-1})\pi$ is freely reduced to give π', and $\ell(\pi') \leq \ell(\pi)$, as claimed.

The algorithm halts in time $O((\ell(\pi) + \ell(f))^{k+4})$ by our comments on lines 5, 6 and 9 and the fact that $\theta^r(a_m^{-1})\pi$ in the final line has length at most $3\ell(\pi)$: after all, $\theta^r(a_m^{-1}) = \theta^{r-1}(a_{m-1}^{-1})\theta^{r-1}(a_m^{-1})$ and $\ell(\theta^{r-1}(a_{m-1}^{-1}))$ is at most $\ell(\theta^{r-1}(a_m^{-1}))$, and $\theta^{r-1}(a_m^{-1})$ is the inverse of a prefix of π. □

Next we construct $\texttt{Back}_m$, $\texttt{Piece}_m$ and $\texttt{Push}_m$.

For a rank-m piece π which does not start with the letter a_m, $\texttt{Back}_m$ determines whether $t^{f(0)}\pi \in Ht^s$ for some $s \in \mathbb{Z}$, and if so it outputs a ψ-word f' with $f'(0) = s$. Initially, it works similarly to $\texttt{Front}_m$ in that it reduces its task to performing a similar determination without the final letter a_m^{-1}. But then it calls $\texttt{Push}_{m-1}$ to find out whether the s exists, and, if so, to output a ψ-word f' with $f'(0) = s$. A crucial feature of this algorithm is that the lengths of the input data to $\texttt{Push}_{m-1}$ (specifically u' and f) is carefully bounded in terms of the length of the inputs to $\texttt{Back}_m$, and so does not blow up course of the induction.

Algorithm 4.3 — $\mathtt{Back}_m$, $m \geq 3$.

∘ Input a rank-m piece $\pi = ua_m^{-\epsilon_2}$ (so $u = u(a_1, \ldots, a_{m-1})$ is reduced and $\epsilon_2 \in \{0, 1\}$) and a valid ψ-word $f = f(\psi_1, \ldots, \psi_k)$. 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 $\mathrm{rank}(f') \leq \max\{\mathrm{rank}(f), m\}$.

∘ Halt in time $O((\ell(\pi) + \ell(f))^{2m+k})$.

---

run $\mathtt{Push}_{m-1}(u, f)$ to test whether or not $t^r u \in \bigcup_{s\in\mathbb{Z}} H_k t^s$
if it is, let g be the valid ψ-word it outputs such that $t^r u \in H_k t^{g(0)}$
3:
if $\epsilon_2 = 0$,
  if $t^r u \in H_k t^{g(0)}$ (so, (a) of the Criterion holds with $s = g(0)$), **return** $f' := g$
6: **else declare** $t^r\pi \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$
  halt

9: we now have that $\epsilon_2 = 1$
run $\mathtt{Psi}(\psi_m^{-1}g)$ to check validity of $\psi_m^{-1}g$ (so whether $g(0) \in \mathrm{Img}\,\psi_m$)
and, if so, to check $\psi_m^{-1}g(0) \leq 0$ (so, whether (*b*) of the Criterion holds with $s = \psi_m^{-1}g(0)$)
12: **if** so, **halt** and **return** $f' := \psi_m^{-1}g$

run $\mathtt{Prefix}_m(\pi^{-1})$ to determine the maximum i (if any) such that $a_{m-1}^{-1}\theta^{i-1}(a_m^{-1})$ is a suffix of π
15: if there is no such i **halt** and **declare** $t^r\pi \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$
for $s = 1$ to i **do**
  run $\mathtt{Push}_{m-1}(u', f)$ where u' is the freely reduced word representing $ua_m^{-1}\theta^s(a_m)$
18: **if** it outputs a ψ-word h, **run** $\mathtt{Psi}(\psi_1^{s-1}h)$ to check if $h(0) = s - 1$
    if so **halt** and **return** $f' := \psi_1 h$
end for
21:
declare that $t^{f(0)}w \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$

For $m \geq 3$, correctness of $\mathtt{Push}_{m-1}$ *(as specified below) implies correctness of* $\mathtt{Back}_m$. The idea is to employ the Piece Criterion in the instance when $\epsilon_1 = 0$, and therefore $\delta = r$, $\pi' = \pi$ and Condition i holds. In this circumstance, the Criterion tells us that $t^r\pi \in H_k t^s$ (that is, $t^\delta\pi' \in H_k t^s$) if and only if (*a*, *b* or *c*) holds.

2: Referring to the specifications of $\mathtt{Push}_{m-1}$, we see that $\ell(g) \leq \ell(u) + \ell(f)$ and $\mathrm{rank}(g) \leq \max\{\mathrm{rank}(f), m\}$.

4–6: $\mathtt{Push}_{m-1}$ in line lines 1–2 tests whether or not $t^\delta x^{-\epsilon_1}u$ (that is, $t^r u$) is in $\bigcup_{s\in\mathbb{Z}} H_k t^s$ and, if so, it identifies the s such that $t^\delta x^{-\epsilon_1}u \in H_k t^s$. 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. (This instance of the Criterion has no real content because $t^\delta x^{-\epsilon_1}u = t^r\pi$. The other two instances that follow are more substantial but will follow the same pattern of reasoning.) By our comment on line 2, $\ell(f') \leq \ell(f) + \ell(u) = \ell(f) + \ell(\pi)$, and $\mathrm{rank}(f') \leq \max\{\mathrm{rank}(f), m\}$, as required.

10–12: Again, we refer back to lines 1–2 for whether or not $t^\delta x^{-\epsilon_1}u$ (that is, $t^r u$) is in $\bigcup_{s_0\in\mathbb{Z}} H_k t^{s_0}$. Assuming that it is, in fact, it is in $H_k t^{g(0)}$, and then Condition *b*, 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 $t^r\pi \in \bigcup_{s\in\mathbb{Z}} H_k t^s$, 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 $\mathrm{rank}(f') \leq \max\{\mathrm{rank}(f), m\}$, as required.

14–20: The aim here is to determine whether Condition *c* 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 14 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 *c* fails, and, as we know at this point that Conditions *a* and *b* also fail, we declare in line 15 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 16–20 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 $ua_m^{-1}\theta^s(a_m)$, 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, $\mathbf{Psi}$ 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 $\pi = ua_m^{-1}$ has suffix $\theta^{s-1}(a_m^{-1})$, when we form u' by freely reducing $ua_m^{-1}\theta^s(a_m)$, 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 *a*, *b* and *c* 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 $\mathbf{Push}_{m-1}$, $\mathbf{Psi}$, 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 $\mathbf{Back}_m$. □

The correctness of $\mathbf{Piece}_2$. By applying Proposition 3.1 repeatedly, we see that $t^{f(0)}\pi \in H_k t^s$ if and only if $t^{\psi_1^l\psi_2^{\epsilon_1} f(0)} a_2^{-\epsilon_2} \in H_k t^s$, since $\psi_1^l\psi_2^{\epsilon_1} f$ is valid as the domains of ψ_1 and ψ_2 are $\mathbb{Z}$. So, by Corollary 3.2, $t^{f(0)}\pi \in H_k t^s$ if and only if $g = \psi_2^{-1}\psi_1^l\psi_2^{\epsilon_1} f$ is valid and $s = \psi_2^{-1}\psi_1^l\psi_2^{\epsilon_1} f(0)$.

It halts in time $O(\ell(w) + \ell(f)^6)$ because $\mathbf{Psi}$ halts in time $O(\ell(f)^6)$ on input $\psi_2^{-1} f$ by the bounds established in Section 3.3, given that f is of rank 2. □

For $k \geq m \geq 3$*, correctness of* $\mathbf{Back}_m$ *implies correctness of* $\mathbf{Piece}_m$. It follows from the specifications of $\mathbf{Front}_m$ and $\mathbf{Back}_m$, that they combine in the manner of $\mathbf{Piece}_m$ to declare whether or not $t^{f(0)}\pi \in \bigcup_{s\in\mathbb{Z}} H_k t^s$, and if it is to return a g with the claimed properties.

Using that $\ell(\pi') \leq \ell(\pi)$ and $\ell(f') \leq \ell(f) + 1$, we can add the halting-time estimates for $\mathbf{Front}_m$ and $\mathbf{Back}_m$, to deduce that $\mathbf{Piece}_m$ halts in time

$$O((\ell(w) + \ell(f))^{\max\{k+4, 2m+k\}}) \ = \ O((\ell(w) + \ell(f))^{2m+k}).$$ □

Algorithm 4.4 — $\texttt{Piece}_m$, $k \geq m \geq 2$.
- Input a rank-m piece π and a valid ψ-word $f = f(\psi_1, \ldots, \psi_k)$.
- Declare whether or not $t^{f(0)}\pi \in \bigcup_{s \in \mathbb{Z}} H_k t^s$ and, if it is, return a valid ψ-word g such that $t^{f(0)}\pi \in H_k t^{g(0)}$, $\text{rank}(g) \leq \max\{m, \text{rank}(f)\}$, and $\ell(g) \leq \ell(f) + 2(m-1)\ell(\pi) + 2$.
- Halt in time $O((\ell(\pi) + \ell(f))^{2m+k})$.

if $m = 2$
  π is $a_2^{\epsilon_1} a_1^l a_2^{-\epsilon_2}$ for some $l \in \mathbb{Z}$ and some $\epsilon_1, \epsilon_2 \in \{0, 1\}$
  set $g = \psi_2^{-\epsilon_2} \psi_1^l \psi_2^{\epsilon_1} f$
  run $\texttt{Psi}(g)$
  if it declares that g is invalid, **then declare** that $t^{f(0)}\pi \notin \bigcup_{s \in \mathbb{Z}} H_k t^s$
  else return g
  halt

if $m > 2$
  run $\texttt{Front}_m(\pi, f)$
  if it declares that *i*, *ii* and *iii* of the Piece Criterion all fail
    declare that $t^{f(0)}\pi \notin \bigcup_{s \in \mathbb{Z}} H_k t^s$ and **halt**
  else run $\texttt{Back}_m$ on the output (π', f') of $\texttt{Front}_m$ and **return** the result

Algorithm 4.5 — $\texttt{Push}_m$, $k \geq m \geq 1$.
- Input a reduced word $v = v(a_1, \ldots, a_m)$ and a valid ψ-word $f = f(\psi_1, \ldots, \psi_k)$.
- Declare whether or not $t^{f(0)}v \in \bigcup_{s \in \mathbb{Z}} H_k t^s$. If it is, return a valid ψ-word g with $\ell(g) \leq \ell(f) + 2m\ell(v)$, $\text{rank}(g) \leq \max\{m, \text{rank}(f)\}$ and $t^{f(0)}v \in H_k t^{g(0)}$.
- Halt time $O((\ell(v) + \ell(f))^{2m+k+1})$.

if $m = 1$ (and so $v = a_1^l$ for some $l \in \mathbb{Z}$)
  declare yes, **output** $g := \psi_1^l f$ and **halt**
3:
if $m > 1$
  let $\pi_1 \cdots \pi_p$ be the rank-m decomposition of v into pieces as per Section 4.1
6: **set** $f_0 := f$
  for $i = 1$ to p
    run $\texttt{Piece}_m(\pi_i, f_{i-1})$
9: **if** it declares $t^{f_{i-1}(0)}\pi_i \notin \bigcup_{s \in \mathbb{Z}} H_k t^s$, **declare** $t^{f(0)}w \notin \bigcup_{s \in \mathbb{Z}} H_k t^s$ and **halt**
    else set f_i to be its output
  end for
12: **return** $g := f_p$

The correctness of $\texttt{Push}_1$. The case $m = 1$ is handled in lines 1–2. The point is that in G_k we have $t^{f(0)}a_1^l = (a_1 t)^l t^{f(0)-l} \in H_k t^{g(0)}$ since $g(0) = \psi_1^l f = f(0) - l$. That it halts within the time bound is clear. □

For $k \geq m \geq 2$, *correctness of* $\texttt{Piece}_m$ *implies correctness of* $\texttt{Push}_m$. This algorithm runs in accordance with Lemma 6.2 of [12] as we described in Section 4.1.

By the specifications of $\texttt{Piece}_m$, after the ith iteration of the **for** loop,

$$\ell(f_i) \;\leq\; \ell(f) + \sum_{j=1}^{i}(2(m-1)\ell(\pi_j) + 2) \;\leq\; \ell(f) + 2(m-1)\ell(v) + 2i \;\leq\; \ell(f) + 2m\ell(v),$$

as $i \leq \ell(v)$, and $\text{rank}(f_i) \leq \max\{m, \text{rank}(f)\}$. In particular, $\text{rank}(g) \leq \max\{m, \text{rank}(f)\}$, as claimed.

$\mathtt{Piece}_m(\pi_i, f_{i-1})$ halts in time $O((\ell(\pi_i) + \ell(f_{i-1}))^{2m+k})$ and $p \leq \ell(\pi)$, so for $1 \leq i \leq p$,

$$\ell(\pi_i) + \ell(f_{i-1}) \;\leq\; \ell(\pi_i) + \ell(\pi_1) + \cdots + \ell(\pi_{i-1}) + \ell(f) + i - 1 \;=\; O((\ell(v) + \ell(f))).$$

So $\mathtt{Push}_m$ halts in time $O((\ell(v) + \ell(f))^{2m+k+1})$. □

Correctness of $\mathtt{Piece}_m$ *for* $2 \leq m \leq k$, *of* $\mathtt{Push}_m$ *for* $1 \leq m \leq k$, *and of* $\mathtt{Back}_m$ *for* $3 \leq m \leq k$. We established the correctness of $\mathtt{Push}_1$ and $\mathtt{Piece}_2$ individually. The implications proved above give the correctness of the others by induction in the order:

$$\mathtt{Piece}_2 \implies \mathtt{Push}_2 \implies \mathtt{Back}_3 \implies \mathtt{Piece}_3 \implies \mathtt{Push}_3 \implies \mathtt{Back}_4 \implies \cdots. \quad \square$$

Finally, we are ready for:

Algorithm 4.6 — $\mathtt{Member}_k$, $k \geq 1$.
- ∘ Input a word $w = w(a_1, \ldots, a_k, t)$.
- ∘ Declare whether or not $w \in H_k$.
- ∘ Halt in time $O(\ell(w)^{3k^2+k})$.

convert w to normal form $t^r v$ where $v = v(a_1, \ldots, a_k)$ is reduced, $r \in \mathbb{Z}$, and $t^r v = w$ in G_k, as described at the start of Section 4.1
set $f = \psi_1^{-r}$
run $\mathtt{Push}_k(v, f)$
if it outputs a (necessarily valid) ψ-word g
  then run $\mathtt{Psi}(g)$ to test whether $g(0) = 0$
    if so, **declare** $w \in H_k$ and **halt**
declare $w \notin H_k$

Correctness of $\mathtt{Member}_k$. The process set out at the start of Section 4.1 produces $t^r v$ in time $O(\ell(w)^k)$. Moreover, $\ell(f) = |r| \leq \ell(w)$ and $\ell(v) \leq \ell(w)(\ell(w) + 1)^{k-1}$.

The algorithm calls $\mathtt{Push}_k(v, f)$, which halts in time

$$O((\ell(v) + \ell(f))^{2k+k+1}) \;=\; O((\ell(w)^k + \ell(w))^{2k+k+1}) \;=\; O(\ell(w)^{3k^2+k}).$$

It either declares that $t^r v \notin \bigcup_{s \in \mathbb{Z}} H_k t^s$, and so $w \notin H_k$, or it returns a valid ψ-word g such that $w \in H_k t^{g(0)}$ and $\ell(g) \leq \ell(f) + 2k\ell(v) \leq \ell(w) + 2k\ell(w)(\ell(w) + 1)^{k-1} = O(\ell(w)^k)$. But then $w \in H_k$ if and only if $g(0) = 0$ (by Lemma 6.1 of [12]), which is precisely what the algorithm uses $\mathtt{Psi}(g)$ to check. This call on $\mathtt{Psi}$ halts in time $O((\ell(w)^k)^{k+4}) = O(\ell(w)^{k^2+4k})$ when $k > 1$ and in time $O(\ell(w))$ when $k = 1$. So, as $\max\left\{k^2 + 4k, 3k^2 + k\right\} = 3k^2 + k$ for all $k > 1$, $\mathtt{Member}_k$ halts in time $O(\ell(w)^{3k^2+k})$, as required. □

5. Conclusion

The construction and analysis of $\mathtt{Member}_k$ in the last section solves the membership problem for H_k in G_k in polynomial time, indeed in $O(n^{3k^2+k})$ time, where n is the length of the input word, and so proves Theorem 3.

Here is why a polynomial time (indeed $O(n^{3k^2+k+2})$ time) solution to the word problem for Γ_k follows, giving Theorem 2.

Suppose we have a word $x = x(a_1, \ldots, a_k, p, t)$ of length n on the generators of Γ_k. Recall that Γ_k is the HNN-extension of G_k with stable letter p commuting with all elements of H_k. Britton's Lemma (see, for example, [6, 25, 34]) tells us that if $x = 1$ in Γ_k, then it has a subword $p^{\pm 1} w p^{\mp 1}$ such that $w = w(a_1, \ldots, a_k, t)$ and represents an element of H_k.

There are fewer than n subwords $p^{\pm 1} w p^{\mp 1}$ in x such that $w = w(a_1, \ldots, a_k, t)$. As discussed above, $\mathtt{Member}_k$ checks whether such a $w \in H_k$ in time in $O(n^{3k^2+k})$. If none represents an element of H_k, we conclude that $x \neq 1$ in G_k. If, for some such subword $p^{\pm 1} w p^{\mp 1}$, we find $w \in H_k$, then we can remove the $p^{\pm 1}$ and $p^{\mp 1}$ to give a word of length $n-2$ representing the same element of G_k.

This repeats at most $n/2$ times until we have either determined that $x \neq 1$ in Γ_k, or no $p^{\pm 1}$ remain. In the latter case, we then have a word on $a_1^{\pm 1}, \ldots, a_k^{\pm 1}, t^{\pm 1}$ of length at most n, which represents an element of the subgroup G_k. But G_k is automatic (Theorem 1.3 of [12]) and so there is an algorithm solving its word problem in $O(n^2)$ time (Theorem 2.3.10 of [13]).

In all, we have called $\mathtt{Member}_k$ at most $n^2/2$ times and an algorithm solving the word problem in G_k once, in every case with input of length at most n. It follows that the whole process can be completed in time $O(n^{3k^2+k+2})$.

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. arXiv:1103.1232.

[12] W. Dison and T. R. Riley. Hydra groups. *Comment. Math. Helv.*, 88(3):507–540, 2013.

[13] D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson, and W. P. Thurston. *Word Processing in Groups*. Jones and Bartlett, 1992.

[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.

[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.

[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.

[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.

[23] M. Lohrey. *The compressed word problem for groups*. Springer Briefs in Mathematics. Springer, New York, 2014.

[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] R. C. Lyndon and P. E. Schupp. *Combinatorial Group Theory*. Springer-Verlag, 1977.

[26] 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.

[27] 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.

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

[29] 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).

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

[31] Mark Sapir. Asymptotic invariants, complexity of groups and related problems. *Bull. Math. Sci.*, 1(2):277–364, 2011.

[32] S. Schleimer. Polynomial-time word problems. *Comment. Math. Helv.*, 83(4):741–765, 2008.

[33] P. Schupp. personal communication.

[34] J. Stillwell. *Classical Topology and Combinatorial Group Theory*. Graduate Texts in Mathematics. Springer-Verlag, second edition, 1993.