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 ................................ 3
  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. Computational models and complexity bounds .................... 7
  1.7. Comparison with power circuits and straight-line programs ....... 8
  1.8. Acknowledgement .................................................... 8

2. Efficient calculation with Ackermann-compressed integers ........ 8
  2.1. Preliminaries ............................................................ 8
  2.2. Examples and general strategy ...................................... 10
  2.3. Our algorithm ....................................................... 16

3. Efficient calculation with ψ-compressed integers .................... 25
  3.1. ψ-functions and ψ-words ............................................. 25

Date: January 30, 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.
1. Introduction

1.1. Ackermann functions and compressed integers. Ackermann functions \( A_0 : \mathbb{Z} \rightarrow \mathbb{Z}, \)
\( A_1 : \mathbb{Z} \rightarrow \mathbb{Z} \) and for \( i \geq 2, A_i : \mathbb{N} \rightarrow \mathbb{N} \) 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 \( A_{i+1}(n+1) = A_i(A_{i+1}(n)) \) 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,
\[
A_3 A_0 A_1^2 A_0 (0) = A_3 A_0 A_1^2 (1) = A_3 A_0 A_1 (2) = A_3 A_0 (4) = A_3 (5) = 2^{65536}
\]
because
\[
A_3 (5) = A_2^2 A_3 (0) = A_2^2 (1) = 2^{2^{2^2}} = 2^{65536}.
\]

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 ineffi-
cient because of the sizes of the integers involved. 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 \( A_0^{\pm 1}, \ldots, A_k^{\pm 1}, \) declares whether or not \( w(0) \) represents an integer, and if so
whether \( w(0) < 0, w(0) = 0 \) or \( w(0) > 0. \)

(The manner in which \( w(0) \) might fail to represent an integer is that as it is evaluated from
right to left, an \( A_i^{\pm 1} \) 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 \( \Gamma \) with a generating set \( A \) can be represented by words—that is, products of elements of \( A \) and their inverses. To work with \( \Gamma \), it is useful to have an algorithm which, on input a word, declares whether that word represents the identity element in \( \Gamma \). After all, if we can recognize when a word represents the identity, then we can recognize when two words represent the same group element, and thereby begin to compute in \( \Gamma \). The issue of whether there is such an algorithm is known as the word problem for \((\Gamma, A)\) 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 \( \Gamma \) has a finite presentation

\[
\langle a_1, \ldots, a_m \mid r_1, \ldots, r_n \rangle.
\]

The Dehn function \( \text{Area} : \mathbb{N} \to \mathbb{N} \) quantifies the difficulty of a direct attack on the word problem: roughly speaking \( \text{Area}(n) \) is the minimal \( N \) such that if a word of length at most \( n \) represents the identity, then it does so ‘as a consequence of’ at most \( N \) defining relations.

Here is some notation that we will use to make this more precise. Associated to a set \( \{a_1, a_2, \ldots\} \) (an alphabet) is the set of inverse letters \( \{a_1^{-1}, a_2^{-1}, \ldots\} \). The inverse map is the involution defined on \( \{a_1^{\pm 1}, a_2^{\pm 1}, \ldots\} \) that maps \( a_i \mapsto a_i^{-1} \) and \( a_i^{-1} \mapsto a_i \) for all \( i \). 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 \( w = x_1 \cdots x_i \mapsto x_i^{-1} \cdots x_1^{-1} = w^{-1} \) when each \( x_i \in \{a_1^{\pm 1}, a_2^{\pm 1}, \ldots\} \).

Words \( u \) and \( v \) are cyclic conjugates when \( u = ab \beta \) and \( v = \beta a \) for some subwords \( \alpha \) and \( \beta \). Freely reducing a word means removing all \( a_j^{\pm 1}a_j^{-1} \) subwords. For \( \Gamma \) presented as above, applying a relation to a word \( w = w(a_1, \ldots, a_m) \) means replacing some subword \( \tau \) with another subword \( \sigma \) 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 \( \Gamma \), \( \text{Area}(w) \) 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 \( \text{Area} : \mathbb{N} \to \mathbb{N} \) is defined by

\[
\text{Area}(n) := \max \{ \text{Area}(w) \mid \text{words } w \text{ with } \ell(w) \leq n \text{ and } w = 1 \text{ in } \Gamma \}.
\]

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 \( \text{Area}(n) \) 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, \( \text{Area}(n) \) is bounded above by a recursive function (in fact \( \text{Area}(n) \) is a recursive function) \([14]\).

There are finitely presented groups for which there are far more efficient algorithms than this brute-force approach. A simple example is

\[
\mathbb{Z}^2 = \langle a, b \mid ab = ba \rangle
\]
There are also ‘naturally arising’ groups which have fast growing Dehn function but an
algorithm 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 $a^\pm 1$ to be cancelled,
and likewise each $b^\pm 1$ together with a $b^\pm 1$. 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 $\text{Area}(n)$ is a fast growing recursive function (so
the ‘brute force’ algorithm succeeds but is extremely inefficient), but there are efficient
ways to solve the word problem. Cohen, Madlener & Otto built extraordinary examples,
in a series of papers [7, 8, 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 $n \mapsto \exp^{O(n)}(n)$ for
some $\ell$. 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 $n \mapsto A_\omega(n)$.

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

There are also ‘naturally arising’ groups which have fast growing Dehn function but an
efficient (that is, polynomial-time) solution to the word problem. A first example is
\[
\langle a, b \mid b^{-1}ab = a^2 \rangle.
\]
Its Dehn function grows exponentially (see, for example, [4]), but the group admits a faithful
matrix representation
\[
a \mapsto \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad b \mapsto \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix},
\]
so it is possible to check efficiently when a word on $a^\pm 1$ and $b^\pm 1$ represents the identity by
multiplying out the corresponding string of matrices.

A celebrated 1-relator group due to Baumslag [1] provides a more dramatic example:
\[
\langle a, b \mid (b^{-1}a^{-1}b) a (b^{-1}ab) = a^2 \rangle.
\]
Platonov [29] proved its Dehn function grows like $n \mapsto \exp_2(\exp_2(\exp_2(1))) \cdots$, 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
\[
\langle a, b, c, d \mid b^{-1}ab = a^2, \ c^{-1}bc = b^2, \ d^{-1}cd = c^2, \ a^{-1}da = d^2 \rangle
\]
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 \(\theta(a_i) = a_ia_{i-1}\) for \(i = 2, \ldots, k\). The family 
\[
G_k := \langle a_1, \ldots, a_k, t \mid t^{-1}a_it = \theta(a_i) \forall i > 1 \rangle,
\]
are called hydra groups. Take HNN-extensions 
\[
\Gamma_k := \langle a_1, \ldots, a_k, t, p \mid t^{-1}a_it = \theta(a_i), [p, a_it] = 1 \forall i > 1 \rangle
\]
of \(G_k\) where the stable letter \(p\) commutes with all elements of the subgroup 
\[
H_k := \langle a_1t, \ldots, a_kt \rangle.
\]

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

**Theorem 2.** For all \(k\), the word problem of \(\Gamma_k\) 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 \(\Gamma_k\) growing like \(n \mapsto A_k(n)\). The massive gap between Dehn function and the time-complexity of the word problem for \(\Gamma_k\) 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
\[
\text{Dist}^{G}_H(n) := \max\{d_T(1, g) \mid g \in H \text{ with } d_S(1, g) \leq n\}.
\]

(Distortion is defined here with respect to specific \(S\) and \(T\), but their choices do not affect the qualitative growth of \(\text{Dist}^{G}_H(n)\).) 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 \(\text{Dist}^{G}_H(n)\), 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 \(\text{Dist}^{G}_H(n)\) 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 \(\text{Dist}^{G_k}_H\) 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 $a_ia_{i-1}$ (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 $a_2a_2a_1$ is annihilated in 5 steps:

$$a_2a_2a_1 \rightarrow a_3a_2a_1 \rightarrow a_2a_1a_1 \rightarrow a_1a_1 \rightarrow a_1 \rightarrow \text{empty word}.$$ 

Define $\mathcal{H}(w)$ to be the number of steps required to reduce a hydra $w$ to the trivial word (so $\mathcal{H}(a_3a_2a_1) = 5$). Then, for $k = 1, 2, \ldots$, define functions $\mathcal{H}_k : \mathbb{N} \rightarrow \mathbb{N}$ by $\mathcal{H}_k(n) = \mathcal{H}(a_t^n)$. 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 $\text{Dist}_{G_k}$ 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 $\{a_1, \ldots, a_t\}^{+1}$ of length $\mathcal{H}_k(n)$ representing $a_t^n\mathcal{H}_k(n)$ in $G_k$ on account of the hydra phenomenon. (For example, $u_{2,3} = (a_2t)^3(a_1t)(a_1t)^3$ equals $a_3^2t^5$ 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 $a_3^2$.) This can be used to show that in $G_k$

$$a_t^n a_2t a_t^{-1} a_2^{-1} a_t^{-n} = u_{k,n} (a_2t) (a_1t) (a_2t)^{-1} u_{k,n}^{-1}.$$ 

The word on the left is a product of length $2n + 4$ of the generators $\{a_1, \ldots, a_n\}^{+1}$ of $G_k$ and that on the right is a product of length $2\mathcal{H}_k(n) + 3$ of the generators $\{a_1, \ldots, a_t\}^{+1}$ 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 $w(A_0, \ldots, A_t)$ 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 $A_iA_{i+1}^{-1}$ (this substitution stems from the recursion defining Ackermann functions), or we delete a subword $A_iA_{i+1}^{-1}$ or $A_{i+1}^{-1}A_i$. The aim of these changes is to eliminate all the letters $A_i^{-1}, \ldots, A_t^{-1}$ in $w$, as these present the greatest obstacle to checking whether such a word represents an integer. Once no $A_i^{-1}, \ldots, A_t^{-1}$ remain in $w'$, when calculating $w'(0)$ letter-by-letter starting from the right, only $A_0^{-1}$ 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 $A_i^{-1}uA_i$ 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 $A_i(0) \in \text{Img} A_i$ to another element of $\text{Img} A_i$ since the gaps between elements of $\text{Img} A_i$ are large.

A number of difficulties arise. For instance, there are exceptional cases when replacing $A_{i+1}$ by $A_iA_{i+1}^{-1}$ 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 \( \psi \)-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 \( t \)-words at the front by shuffling them to the left through the word, applying \( \theta \)-functions as appropriate to the intervening \( a_i \) so that the element of \( G_k \) represented does not change. The result is a word \( t' \nu \) where \( |t'| \leq \ell(w) \) and \( \nu = \nu(a_1, \ldots, a_k) \) has length at most a constant times \( \ell(w)^k \). Then carry the \( t' \) back through \( \nu \) working from left to right, converting (if possible) what lies to the left of the power of \( t \) to a word on the generators \( a_1t, \ldots, a_kt \) 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 \( \psi \)-functions. Very roughly speaking, checking whether this process ever gets stuck (in which case \( w \notin H_k \)) amounts to checking whether an associated \( \psi \)-word is valid. If the end of the word is reached, we then have a word on \( a_1t, \ldots, a_kt \) times some power of \( t \), where the power is represented by a \( \psi \)-word. We then determine whether or not \( w \in H_k \) by checking whether or not that \( \psi \)-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, \( \nu \) is partitioned into subwords which we call pieces. These pieces are determined by the locations of the \( a_k \) and \( a_k^{-1} \) in \( \nu \). Each contains at most one \( a_k \) and at most one \( a_k^{-1} \), and if the \( a_k \) is present in a piece, it is the first letter of that piece, and if the \( a_k^{-1} \) is present, it is the last letter. The power of \( t \) is, in fact, carried through one piece at a time. Whether it can be carried through a piece \( a_k^{\pm 1}ua_k^{\pm 2} \) (here, \( \epsilon_1, \epsilon_2 \in \{0, 1\} \) 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 \( a_k^{\pm 1} \) 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 \( t' \) can pass through a piece \( \pi \)—that is, whether \( t'\pi = \sigma t' \) for some \( \sigma \in H_k \) and some \( s \in \mathbb{Z} \)—and, if it can, how to represent \( s \) by an \( \psi \)-word.

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

1.6. Computational models and complexity bounds. Formally, our complexity bounds depend on the computation model. Following [23], we have in mind Random Access Machines (RAMs), which are conveniently flexible. As explained in [23], a RAM can be polynomially simulated by a multi-tape Turing machine, provided that the integers occupying the registers do not grow above an exponential of the input length (that is, linear bit-length), which is the case in all our algorithms. For instance, in Algorithms 2.1 and 3.1 (Bounds and BoundsII), we work with integers of size at most \( t \), which is the integer initially inputted in binary. All our other algorithms (except in so far as they call Bounds and BoundsII) are manipulations of words and the integers involved have size at most polynomial in the lengths of these words.
We give polynomial upper bounds on the time complexities of our algorithms, but make no claim on their sharpness. Indeed, the first step of our core algorithm (**Member**.) is to shuffle all the \( r^1 \) to the front of the word at the potential expense of polynomially increasing the length of the word. While convenient, this is likely inefficient.

Given this likely inefficiency and the dependence on the computation model, at points in our analysis we are content to give bounds that are crude, but are straightforward to derive and to carry forward to subsequent analysis.

It is interesting to speculate whether our algorithms can be improved or better analyzed to give polynomial complexity bounds of degrees which are independent of the rank \( k \). We believe the complexity bounds for our algorithms as they stand do depend on \( k \). This is because \( \text{Pinch}_k \) makes recursive calls on \( \text{Pinch}_{k-1} \) and so there is a depth-\( k \) recursion (and the lengths of the words fed recursively into \( \text{Pinch}_{k-1} \) may increase with each step of the recursion).

1.7. **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].

1.8. **Acknowledgement.** We are grateful to the referees for careful readings and for thoughtful comments.

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 $\psi_i$ 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.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>⋯</th>
<th>n</th>
<th>⋯</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_0$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>⋯</td>
<td>$n + 1$</td>
<td>⋯</td>
</tr>
<tr>
<td>$A_1$</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>⋯</td>
<td>$2n$</td>
<td>⋯</td>
</tr>
<tr>
<td>$A_2$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>⋯</td>
<td>$2^n$</td>
<td>⋯</td>
</tr>
<tr>
<td>$A_3$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>16</td>
<td>65536</td>
<td>⋯</td>
<td>$2^{2^{\cdot \cdot \cdot n}}$</td>
<td>⋯</td>
</tr>
<tr>
<td>$A_4$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>65536</td>
<td>$2^{2^{\cdot \cdot \cdot n}}$</td>
<td>⋯</td>
<td>65536</td>
<td>⋯</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
</tr>
</tbody>
</table>

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 $A_i(n)$ 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) $A_i^{-1} : \text{Img } A_i \to \mathbb{N}$ 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 sparseness 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 $\text{BasePinch}$). It will tell us that if $u = u(A_1, \ldots, A_{k-1})$ and $uA_k(n) \in \text{Img } A_k$ but $uA_k(n) \neq A_k(n)$, then $\ell(u)$ must be relatively large.

**Lemma 2.1.**

(1) $A_i(1) = 2$ \quad \forall i \geq 0,

(2) $A_i(2) = 4$ \quad \forall i \geq 1,

(3) $A_i(n) \leq A_{i+1}(n)$ \quad \forall i \geq 1; n \geq 0,

(4) $A_i(n) < A_i(n + 1)$ \quad \forall i, n \geq 0,

(5) $n \leq A_i(n)$ \quad \forall i, n \geq 0,

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

(6) $A_i(n) + A_i(m) \leq A_i(n + m)$ \quad \forall i, n, m \geq 1,

(7) $A_i(n) + m \leq A_i(n + m)$ \quad \forall i, n, m \geq 0,

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

**Proof.** Equations (1) and (2) follow from $A_{i+1}(n + 1) = A_iA_{i+1}(n)$ 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 \( k = 2 \) as \( A_k(n) = 2^n \) 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 \( k \geq 2 \). □

When a word \( w = w(A_0, \ldots, A_k) \) is non-empty, we let \( \text{rank}(w) \) denote the maximum \( i \) such that \( A_i \) occurs in \( w \) and \( \eta(w) \) denote the number of \( A_1^{-1}, \ldots, A_k^{-1} \) in \( w \). For example, if \( w = A_4^{-1}A_2A_3^{-1}A_2^{-1}A_2 \), then \( \text{rank}(w) = 4 \) and \( \eta(w) = 2 \).

As we said in Section 1.1, strings of Ackermann functions offer a means of representing integers. For \( x_1, \ldots, x_n \in \{A_0, \ldots, A_k\} \), we say the word \( w = x_nx_{n-1}\cdots x_1 \) is valid if \( x_mx_{m-1}\cdots x_1(0) \) 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, \( w := A_2^{-1}A_1A_0A_0 \) is valid, and \( w(0) = \log_2(2 \cdot 2 \cdot (0 + 1)) = 2 \). But \( A_2^{-1}A_0^{-1}A_1A_0A_0 \) is 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 \( \mathbb{Z} \) (the domain of \( A_1^{-1} \)).

For \( m \in \mathbb{Z} \), the sign of \( m \), denoted \( \text{sgn}(m) \), is \(-, 0, +\) 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 its 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 \( w = A_5^{-6}A_1A_1^{-1}A_3^{-4}A_2A_1A_2A_0 \), which is a word of length 17 with \( \eta(w) = 0 \). We can evaluate directly working from right to left that, if valid, \( w(0) = A_5^{-6}A_1A_0^{-1}A_3^{-1}A_2(12) \). At this point we are reluctant to calculate \( A_3(12) \) as it is enormous, and instead recognize that \( A_3(12) \) is larger than \( \ell(w) = 17 \) (Bounds), which as we will explain in a moment we can do suitably quickly. We then deduce that \( w(0) = 1 \) 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 \( A_i^{-1} \), 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 \( |w_j(0)| > \ell(w) \), 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 \( w(0) - 2 = A_0^{-1}w(0) > 0 \). Here, the size of \( A_2(12) \) still dwarfs \( \ell(A_0^{-1}w) = 19 \), so the computation carried out is essentially the same.

So, how do we determine that \( A_3(12) > 17 \) or, indeed, \( A_3(12) > 19 \) 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 \( A_n(n) \) is less than some \( c \in \mathbb{Z} \). 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.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
<th>( A_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \( A_4(4) = A_3A_3(3) = A_3(16) \), 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 \( A_j(3) \geq A_j(3) \geq 17 \) 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:

\[
\begin{align*}
A_{i+1}^{-1}v &\sim uA_{i+1}A_0^{-1}v & v(0) &> 0 \text{ and } i \geq 1, \\
A_{i+1}^{-1}v &\sim uA_0A_{i+1}^{-1}A_{i+1}^{-1}v & v(0) &> 1 \text{ and } i \geq 1, \\
A_{i+1}^{-1}v &\sim A_iA_{i+1}v & v(0) &\geq 0, i \geq 0.
\end{align*}
\]

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 \( uA_{i+1}^{-1}v \sim uA_0A_{i+1}^{-1}A_{i+1}v \).

Assume \( v \) is valid. If \( v(0) > 0 \), then \( A_{i+1}v \) and \( A_{i+1}A_0^{-1}v \) 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 \( uA_{i+1}^{-1}A_{i+1}v \sim uv \) 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 $w \sim w'$

This lemma only works for suffixes. For example, $A_0A_0^{-1}A_0$ and $A_1^{-1}A_0$ have equivalent prefixes $A_0A_0^{-1}$ and $A_1^{-1}$, but are not equivalent words: the first is valid and the second is not.

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 $A_0$.

   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 $A_{r-1}^{-1}A_0^{-1}$ for this $A_r$, then $A_{r-2}^{-1}\ldots A_0^{-1}$ for the resulting $A_{r-1}$, and so on, as per Lemma 2.4 until we have created an $A_r$ (Reduce).

   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 rank$(u) < r$. (Reduce.)

2. We now invoke a subroutine (Pinch) which will either declare $s$ (and so $w$) invalid, or will convert $s$ to an equivalent word $A_l^0v$ for some $l \in \mathbb{Z}$.

   Suppose first that rank$(u) = r - 1 > 0$. We will explain how to eliminate an $A_{r-1}$ from $u$. On repetition, this will give a word $A_l^0A_r^{-1}uA_rv \sim s$ such that rank$(\tilde{u}) \leq r - 2$. (CutRank.)

   2.1. Find the leftmost $A_{r-1}$ in $s$ and write

   $s = A_r^{-1}uA_{r-1}u'A_rv$

   where rank$(u') < r - 1$ and rank$(u'') \leq r - 1$. Substitute $A_0A_{r-1}^{-1}A_r^{-1}$ for $A_r^{-1}$ as per Lemma 2.4 to give

   $A_0A_{r-1}^{-1}A_r^{-1}uA_{r-1}u'A_rv \sim s$.

   2.2. Apply Pinch$_{r-1}$ to the suffix $A_{r-1}^{-1}uA_{r-1}u'A_rv$ to give an equivalent word $A_l^0u''A_rv$ for some $l' \in \mathbb{Z}$. Thereby get

   $A_0A_{r-1}^{-1}A_l^0u''A_rv \sim s$.

   2.3. Likewise eliminate an $A_{r-1}$ from $u''$ in $A_r^{-1}A_l^0u''A_rv$, and so on, until we arrive at

   $A_m^0A_{r-1}^{-1}\tilde{u}A_rv \sim s$

   such that $m \in \mathbb{Z}$ and 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

   $A_l^0(\tilde{u}A_{r-1})A_r(\tilde{u}A_{r-1}v) \sim s$.

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

   $A_{m+1}^0A_r^{-1}A_l^0A_r^{-1}v \sim s$

   for some $l'' \in \mathbb{Z}$ (FinalPinch$_r$).
3. Eliminate $A_r^{-1}$ and $A_r$ from the suffix $A_r^{-1}A_r^0 A_r A_r^{-1}v$ using a method we will shortly explain via Example 2.7 to give an equivalent suffix $A_r^{m'} A_r^{-1}v$ 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 $A_0 A_r^{m'} A_r^{-1}v$, then $w \sim w'$ and $\eta(w') < \eta(w)$, as required.

4. Repeat steps 1–3 until we have an equivalent word with no $A_i^{-1}, \ldots, A_r^{-1}$.

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

So validity of $v$ and the hypothesis $\nu(0) > 0$ or $\nu(0) > 1$ (and indeed whether $\nu(0) < 0$, whether $\nu(0) = 1$, or whether $\nu(0) \leq 0$, which we will soon also need) can be checked (using Positive) 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 1, $v$ is valid, but $\nu(0) \leq 0$.

- If $i > 0$ and $\nu(0) < 0$, then $A_i A_{i+1} A_{i}^{-1}v$ is invalid, and so $w$ is invalid.
- If $i > 1$ and $\nu(0) = 0$, then $A_i A_{i+1} A_{i}^{-1}v = 1$ and so, instead of making the planned substitution, the suffix $A_{i+1} v$ can be replaced by the equivalent $A_i v$. One might then worry that there will no longer be an $A_{i+1}$ to pair with an $A_i^{-1}$. However, in this special case, if $w$ is not declared invalid as the algorithm progresses, it will produce a word which is equivalent to $w$ and has the suffix $A_i^{-1} A_i^{0} v$, and since $\nu(0) = 0$, this suffix is equivalent to $A_i^{-1} A_i^{0} A_i^{-1} A_{i+1} v$, and our algorithm uses BasePinch to replace this suffix with an equivalent suffix $A_i^{c} v$ for some $c \in \mathbb{Z}$.
- If $i = 1$ and $\nu(0) = 0$, then we have a suffix $A_{2} v$ which we replace by the equivalent $A_{0} A_{1} v$.
- When $i = 0$, no substitution is necessary because $A_i^{-1} u A_i v$ is valid if and only if $u(0)$ is even. If so $u = A_i^{0}$ for some even integer $l$ and $A_i^{-1} u A_i v$ can be replaced by the equivalent $A_{l} / 2 v$.

Suppose, on the other hand, that we are in Case 2, $v$ is valid, but $\nu(0) \leq 1$. The algorithm actually only tries to make substitutions for $A_i^{-1}$ when the input word has suffix $A_i^{-1} u A_{i+1} v_0$ for some subwords $u$ and $v_0$ such that $\eta(u) = \eta(v_0) = 0$ and $\text{rank}(u) < i + 1$ (and $v \equiv u A_{i+1} v_0$). It proceeds as follows:

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

(In Case 2, it is not obvious that outputting $A^{-1} v_0$ 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 $w = A_0 A_2^{-1} A_1 A_0^2 A_2 A_0$. A quick calculation shows $w$ is valid and $w(0) = 4$, but here is how our Ackermann handles it.
1. First aim to eliminate the $A^{-1}_2$ (the subroutine $\textbf{Reduce}$). Look to the right of the $A^{-1}_2$ for the first subsequent letter (if any) of rank at least 2, namely the $A_2$.

2. Try to ‘cancel’ the $A^{-1}_2$ with the $A_2$ ($\textbf{Pinch}_2$) —
   
   2.1. Reduce the rank of the subword $A_1A^{-1}_0$ between $A^{-1}_2$ and $A_2$ as follows ($\textbf{CutRank}_2$).

   2.1.1. Use the technique of Example 2.2 ($\textbf{Positive}$) to check that the suffix $A_1A_0^2A_2A_0$ is valid and $A_1A_0^2A_2A_0(0) > 1$. So, by Lemma 2.4, we can legitimately substitute $A_0A_2^{-1}A^{-1}_1$ for $A^{-1}_2$ to obtain
   
   $A_0A_2^{-1}A^{-1}_1A_1A_0^2A_2A_0 \sim w$.

   2.1.2. Cancel the $A^{-1}_1A_1$ (strictly speaking, this is done by calling $\textbf{CutRank}_2$
   
   on $A_2^{-1}A^{-1}_1A_0A_0^2A_2A_0$) to give
   
   $A_0A_2^{-1}A_0^2A_2A_0 \sim w$.

   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 $\textbf{FinalPinch}$ on the suffix $s := A_2^{-1}A_0^2A_0$ 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
   
   $A_0A_2^{-1}A^{-1}_1A_0^2A_1A_2A_0^{-1}A_0 \sim s$.

   2.2.2. Convert the subword $A_1^{-1}A_0^2A_1$ to a power of $A_0$ (by calling $\textbf{Pinch}_1$ on
   
   $A_1^{-1}A_0^2A_1A_2A_1^{-1}A_0$, which calls $\textbf{BasePinch}$ 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 $A^{-1}_2$ 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 $\textbf{Bounds}(2(f(w)))$). So $A_2^{-1}A_0A_2A_0^{-1}A_0(0) = 1$, and we can conclude that
   
   $s' \sim A_0A_0^{-1}A_0$.

   (Preserving the suffix $A_0^{-1}A_0$ appears unnecessary here, but it reflects
   the recursive design of the algorithm.)

   So

   $w' := A_0A_0^{-1}A_0 \sim w$.

3. Now $\eta(w') = 0$. So evaluate $w'$ from right to left in the manner of Example 2.2 ($\textbf{Positive}$) and declare that $w$ is valid and $w(0) > 0$.

In our next example, the input word has the form $A^{-1}_2uA_rv$ with $\eta(u) = \eta(v) = 0$ and rank($u$) < $r < r'$. As there is no $A_r$ with which we can ‘cancel’ the $A^{-1}_2$, 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 $A^{-1}_2A_0^{-1}A_rv$ (using $\textbf{BasePinch}$) which is crucial for avoiding explicit computation of large numbers.

Example 2.7. Set $w := A_2^{-1}A_0A_0^{-1}$. A moment.

1. Identify the rightmost $A^{-1}_i$ with $i \geq 1$, namely the $A^{-1}_2$. Scanning to the right of
   the $A^{-1}_2$, the first $A_i$ we encounter with $i \geq 2$ is the $A_3$. (Send $w$ to $\textbf{Reduce}$.)
2. Use techniques from Example 2.2 (Positive) to check that $A_0^{100}(0) > 0$. So we can substitute $A_2 A_3 A_0^{-1}$ for $A_3$, as per Lemma 2.4, to obtain
\[ w_0 := A_2^{-1} A_2^{-2} A_2 A_3 A_0^{-1} A_0^{100} \sim w. \]

3. We check we can make substitutions as in Lemma 2.4 for $A_2^{-1}$ and $A_2$ to give
\[ w_1 := (A_0 A_2^{-1} A_1^{-1} A_0^{-2} A_1 A_2 A_0^{-1}) A_3 A_0^{-1} A_0^{100} \sim w. \]

(Run CutRank, on $w_0$ which does nothing as rank($u$) < 1, and then start running FinalPinch($u$).)

4. We now want to reduce the rank of the subword between the $A_2^{-1}$ and $A_2$ to zero (Pinch2), and so we (BasePinch) process the suffix
\[ A_1^{-1} A_0^{-1} A_1 A_2 A_0^{-1} A_3 A_0^{-1} A_0^{100} \]

to replace $A_1^{-1} A_0^{-2} A_1$ by $A_0^{-1}$ giving
\[ w_2 := A_0 A_2^{-1} A_0^{-1} A_2 A_0^{-1} A_3 A_0^{-1} A_0^{100} \sim w \]
(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 Pinch2 to end and return to FinalPinch2, which in turn invokes BasePinch). As $A_2$ is the function $\mathbb{N} \to \mathbb{N}$ mapping $n \mapsto 2^n$, if $A_0 A_2^{-1} A_3 A_0^{-1} A_0^{100}(0)$ is in the domain of $A_2^{-1}$ for some $z \in \mathbb{Z} \setminus \{0\}$, then the large gaps between powers of 2 ensure that $2|z| \geq A_0 A_2^{-1} A_3 A_0^{-1} A_0^{100}(0)$. In the case of $w_2$, we have $z = -1$ and so we see that $w_2$ is invalid by checking that $A_2 A_0^{-1} A_3 A_0^{-1} A_0^{100}(0) > 2$. We can do this efficiently in the manner of Example 2.3 by noting that $A_3 A_0^{-1} A_0^{100}(0)$ exceeds the threshold $\ell(A_2 A_0^{-1} A_3 A_0^{-1} A_0^{100}) + 2 = 106$. 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 exception is the subroutine 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 $A_0^{-1} u A_0 A_0'$ (with $\eta(u) = \eta(v) = 0$ and rank($u$) < $r$) by an equivalent suffix of the form $A_0'A_0$ 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 $A_0^{-1} A_0 A_0'$ where $r \geq 2$ and $z \neq 0$, after all the gaps in the range of $A_0$ grow even faster when $r > 2$. In Lemma 2.1 (8), we showed that if $l \in \mathbb{Z}$ is non-zero, then $A_0^{-1} A_0 A_0'$ is valid, then $2|l| \geq A_0 A_0'(0)$. This condition can be efficiently checked if $\eta(v) = 0$. If $2|l| \geq A_0 A_0'(0)$, direct computation of the value of $A_0^{-1} A_0 A_0'(0)$ (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 $w = A_3^{-1} A_0^{-1} A_3 A_0$. 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 on $A_3^{-1} u A_3 v$ where $u = A_3^{-1}$ 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 $A_0^{-1} A_3 A_0$ 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 \texttt{Positive}.)

3. We notice that there are no $A_{-1}^1$ or $A_{-2}^2$ between $A_3^{-1}$ and $A_3$ to remove. (\texttt{Pinch}_3 runs \texttt{CutRank}(w), which does not change $w$.)

4. We seek to substitute $A_0^{-1}A_3A_0^{-1}$ for $A_3^{-1}$ and $A_2A_3A_0^{-1}$ for $A_3$. (\texttt{Pinch}_3 calls \texttt{FinalPinch}_3.) But, by calculating that $A_0^{-1}A_3A_0^{-1}A_3A_0(0) = 0$ (which is done by calling \texttt{Positive}(A_0^{-1}A_3A_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 (\texttt{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 $A_0^{-m}v(0) = 0 = A_0^{-1}(1) = A_0^{-1}v(0)$. It would be simpler to return the empty word, but the recursive structure of \texttt{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. (\texttt{Positive}.)

2.3. **Our algorithm.** We continue to have an integer $k \geq 0$ fixed and work with words on the alphabet $A_{-1}^0,...,A_{-1}^k$. With the exception of our first algorithm, \texttt{Bounds}, the polynomial time bounds we establish in this section all depend on $k$.

![Figure 1](image-url)

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

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

(In fact, we will see that 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. The estimates of the degrees of the polynomial bounds on running time to come later, could be improved by replacing the $O(\ell)$ contributions from Bounds by polylog estimates, but we prefer not to for the sake of simplicity and as those bounds are likely also not sharp in other respects.)

Correctness of Bounds. Bounds generates its list of triples by first listing the at most $\lceil \log_2 \ell \rceil$ triples of the form $(2, n, A_2(n))$ 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 $\ell$ is expressed in binary. It then reads through this list and uses the recurrence relation (and the fact that $A_2(2) = 4$) to list all the $(3, n, A_2(n))$ for which $n \geq 3$ and $A_2(n) \leq \ell$. It then uses those to list the $(4, n, A_2(n))$ 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 $r \leq \lceil \log_2 \ell \rceil$. As $r$ increases, there are fewer $n$ such that $A_r(n) \leq \ell$ and there are none when $r > \log_2 \ell$. 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 $O(\ell)$.

Remarked 2.9. Bounds does not give any $(r, n, A_r(n))$ for which $A_r(n) > \ell$ but $r \leq 1$ or $n < 2$. 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 $n \in \mathbb{Z}$.

Algorithm 2.2 — Positive.
- Input a word $w = x_n x_{n-1} \cdots x_1$ where $x_1, \ldots, x_n \in \{A_0, A_1, \ldots, A_k\}$.
- Return invalid when $w$ is invalid and $\text{sgn}(w(0))$ when $w$ is valid.
- Halt in time $O(\ell(|w|^3))$.

run Bounds($n$)
1. evaluate $x_1(0)$, then $x_2 x_1(0)$, and so on until
2. either $w(0)$ has been evaluated
3. or some $x_i \cdots x_1(0) > n$ (checked by consulting the output of Bounds($n$))
4. or some $x_i \cdots x_1(0) < -n$ (leading to either $w$ being invalid or $w(0) < 0$)
5. or some $x_i \cdots x_1$ is found invalid (that is, $x_i \neq A_0^{-1}, A_1$ and $x_{i-1} \cdots x_1(0) < 0$)

then, respectively, return
6. $\text{sgn}(w(0))$
7. $\text{sgn}(w(0)) = +$
8. if $x_{i+1}, \ldots, x_n \notin \{A_2, \ldots, A_k\}$, then $\text{sgn}(w(0)) = -$, else invalid
9. invalid
Our next subroutine is the rank\((\ord n)\) of the image of \(A\) (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 \(A_0, A_1, \ldots, A_k\) 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, \(\tau n\), is exceeded, then \(w\) must be valid and \(w(0) > 0\), as subsequent letter-by-letter evaluation could never reach a negative value. If \(x_1\ldots x_i(0) < -n\) 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 \(O(n)\) (a crude estimate). 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 \(O(n^2)\) (also crude), so they and the necessary searches of the output of **Bounds** take time \(O(n^3)\).

Our next subroutine is the rank\((\ord n) = 0\) case of **Pinch**, to come.

**Algorithm 2.3 — BasePinch**

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

\[
l := \ell(0) \text{ (so } A_0^{-1} uA_r v \text{ with all } A_0^\pm A_0^{-1} \text{ subwords removed and } A_r^{-1} A_0^{-1} A_r uA_r v \sim w)\]

\[
2. \text{ if } \eta(w) = \text{ invalid, return invalid}\]

3. if \(r \geq 2\) and \(v(0) \neq 0\) (checked using **Positive**), return invalid

4. if \(l = 0\), return \(w' := v\)

5. if \(r = 1\), 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 \(r > 1\)

7. run **Positive**\((A_r^{-1} A_r v)\) to determine if \(A_r^{l} A_r v(0) \leq 0\) (so outside the domain of \(A_r^{-1}\))

8. if so, return invalid

9. run **Positive**\((A_0^{l-2} A_r v)\) to determine whether \(A_r v(0) > 2|l|\)

10. if so, return invalid

11. we now have that \(0 \leq v(0) \leq |l|\) and \(0 < A_r v(0) \leq 2|l|\) and \(A_r v(0) + l \leq 3|l|\)

12. calculate \(v(0)\) by running **Positive**\((A_0^{-1} v)\) for \(i = 0, 1, \ldots, |l|\)

13. run **Bounds**\((3 |l|)\)

14. search the output of **Bounds**\((3 |l|)\) to find \(A_r v(0)\)

15. set \(m := A_r v(0) + l\)

16. search the output of **Bounds**\((3 |l|)\) for \(c\) with \(A_r(c) = m\) (so \(c = A_r^{-1} A_0^{-1} A_r v(0) = w(0))\)

17. if such a \(c\) exists, return \(w' := A_0^{c-v(0)} v\)

18. else 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 \( A'_1(A,v(0)) \) is the parity of \( l \) when \( r = 1 \), and determines the validity of \( w \).

8, 10: We know \( A'_0(A,v) \) and \( A_0^{2x}A,v \) are valid at these points because \( A,v \) is valid.

11: Let \( q \neq v(0) \). For all \( p \neq q \) we have \( |A_r(q) - A_r(p)| \geq \frac{1}{3}A_r(q) \) by Lemma 2.1 (8), and so \( |A_r(q) - A_r(p)| > |l| \). If \( A_r^{-1}A'_0A,v \) is valid, then there exists \( p \in \mathbb{N} \) such that \( A_r(p) = A'_r(A,v(0)) = l + A_r(q) \), 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 \( \text{Img}_A \) contains only positive integers. And, \( A_r(v(0)) \leq 2|l| \) because of lines 10 and 11. It follows that \( v(0) \neq 0 \) because \( 2v(0) = A_1v(0) \leq A_rv(0) \leq 2|l| \). And \( v(0) \geq 0 \) since \( v(0) \) is in the domain of \( A_r \), which is \( \mathbb{N} \) when \( r > 1 \). We have \( A'_iA_rv(0) \leq 3|l| \) here because \( A_rv(0) \leq 2|l| \) and so \( A'_iA_rv(0) \leq l + 2|l| \).

18: If \( m = A_rv(0) + l = A'_0A_rv(0) \) 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: \( A_0^{2x}v(0) = c = A_r^{-1}(l + A_rv(0)) = A_{r-1}A'_0A_rv(0) \).

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 \( A_r(v(0) + l) \geq A_r(v(0)) \). So \( v(0) + l \geq A_r^{-1}(A_rv(0) + l) = c \). 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

\[ c = A_r^{-1}A'_0A_rv(0) \leq A_{r-1}A_rv(0) = v(0) \]

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 \( v(0) + l \leq A_r^{-1}(A_rv(0) + l) = c \). So \( c - v(0) < 0 \) because \( A_r(c) = A_rv(0) + l < A_rv(0) \). 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 \textbf{Positive} on words of length at most \( 3\ell(w) \), and so (by the properties of \textbf{Positive} established), each time it is called, \textbf{Positive} calculates with integers no larger than \( 3\ell(w) \). On input \( 3|l| \leq 3\ell(w) \), \textbf{Bounds} calculates with integers of absolute value at most \( 3\ell(w) \). The only remaining integer manipulations concern \( m, l, 2|l|, A_rv(0), \) all of which have absolute value at most \( 3\ell(w) \).

Finally, that \textbf{BasePinch} halts in time \( O(\ell(w)^3) \) is straightforward given the previously established cubic and linear halting times for \textbf{Positive} and \textbf{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 \textbf{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 \textbf{Bounds} at most once—in that event the input to \textbf{Bounds} is a non-negative integer that is at most \( 3\ell(w) \) and the output of \textbf{Bounds} is searched at most twice and has size \( O((\log_2(\ell(w)))^2) \). \( \Box \)
Algorithm 2.4 — OneToZero.
- Input a valid word \(w = A_r^{-1}uA_r v\) with \(\eta(u) = \eta(v) = 0\), \(u \neq \epsilon\), \(uA_r v(0) = 1\) and \(r \geq 2\).
- Return a word \(A_r^{-v(0)}v \sim w\) of length at most \(\ell(w) - 2\).
- Halt in time \(O(\ell(w)^4)\).

\[
\text{run Positive}(A_r^{-m}v) \text{ for } m = 0, 1, \ldots \text{ until it declares that } A_r^{-m}v = 0
\]
\[
\text{output } A_r^{-m}v
\]

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 \(A_r^{-m}v \sim w\) as required, since \(A_r^{-m}v(0) = 0\).

Since \(\eta(u) = 0\), the only letter \(u\) may contain which decreases the value in the course of evaluating \(uA_r v(0)\) is \(A_r^{-1}\). So, as \(uA_r v(0) = 1\) and \(A_r v(0) \geq v(0) + 1\), there must be at least \(v(0)\) letters \(A_r^{-1}\) in \(u\). So \(\ell(u) \geq v(0)\). So \(\ell(A_r^{-v(0)}v) \leq \ell(w) - 2\), 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 \(A_r^{-v(0)}v\). However, OneToZero is used by Pinch, which we will describe next and whose inductive construction requires the suffix \(v\).

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

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

\[
\text{if } r = 1 \text{ return BasePinch}(w)
\]
\[
\text{run Positive}(v) \text{ to determine whether } v \text{ is invalid or } v(0) < 0
\]
3: \(\text{if } \text{so return invalid}
\]
\[
\text{run Positive}(uA_r v) \text{ to determine whether } uA_r v \text{ is invalid or } uA_r v(0) \leq 0
\]
\(\text{if so, return invalid}
\)
6: \(\text{run CutRank}(w)
\)
\(\text{it either declares } w \text{ invalid, in which case return invalid}
\]
or \(\text{it returns a word } w' := A_r^{l'}v \sim w\) in which case return \(w'
\)
9: \(\text{or it returns a word } w' := A_r^{l'}uA_r v \text{ such that}
\]
\(w' \sim w, \ell(w') \leq \ell(w), \eta(u') = 0, u' \neq \epsilon \text{ and } \text{rank}(u') < r - 1
\)
\(\text{run FinalPinch}(A_r^{-1}uA_r v)
\)
12: \(\text{if it declares } A_r^{-1}uA_r v \text{ invalid, return invalid}
\]
\(\text{else if outputs } A_r^{l'}v \text{ for some } l, \text{ in which case set } w'' := A_r^{l'}v
\)
\(\text{return } w''
\)
Algorithm 2.6 — CutRank, for \( r \geq 2 \).

1. Input a word \( w = A_r^{-1} u A_r v \) with \( \eta(u) = \eta(v) = 0 \) and \( \text{rank}(u) \leq r - 1 \).
2. Either declare \( u \) invalid, or return \( w' = A_0^{-1} u A_0 v \) where \( \ell(w') \leq \ell(w) - 2 \), or return \( w' = A_0^{-1} A_r^{-1} u A_r v \) with \( \text{rank}(u') \leq r - 2 \), \( \eta(u') = 0 \), and \( \ell(w') \leq \ell(w) \).
3. Halt in time \( O(\ell(w)^{4r(r-1)}) \).

\[
i := 0 \text{ and re-express } w \text{ as } A_r^{-1} u A_r v
\]

\[
\text{if } v(0) < 0 \text{ (checked using Positive), return invalid}
\]

6. \( \text{while } \text{rank}(u) = r - 1 \) do

\[
\text{run Positive}(A_0^{-1} u A_0 v) \text{ to test whether } u A_r v(0) = 1
\]

\[
\text{if so return the output } w' := A_r^{-1} v = \text{OneToZero}(w)
\]

9. \( \text{express } u \text{ as } u' A_{r-1} u'' \) where \( \text{rank}(u') < r - 1 \) (i.e. locate the leftmost \( A_{r-1} \) in \( u \))

\[
\text{increment } i \text{ by 1}
\]

\[
w := A_r^{-1} u A_r^{-1} u' A_{r-1} u'' A_r v \text{ (i.e. substitute } A_0 A_r^{-1} A_{r-1} \text{ for } A_r^{-1} \text{ in } w)
\]

12. \( \text{run Pinch}_{r-1}(A_r^{-1} u A_{r-1} u' A_r v)
\]

\[
\text{if it returns invalid, return invalid}
\]

\[
\text{else let } w_0 := A_r^{-1} u'' A_r v \text{ be the (valid) word returned}
\]

\[
w := A_r^{-1} w_0
\]

\[
u := A_0^{-1} u'' \text{ so that } w = A_0^{-1} u A_r v
\]

end while

18. return \( w \)

Correctness of \( \text{Pinch}_{r-1} \) implies the correctness of \( \text{CutRank} \), for all \( r \geq 2 \). The idea of \( \text{CutRank} \), 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 \( \text{OneToZero} \).

8: If \( u A_r v(0) \leq 0 \), 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 \( u A_r v(0) \leq 1 \). 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 \( \text{Pinch}_{r-1} \) (assuming it does not return invalid) 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 \( A_r^{-1} u' A_{r-1} u'' A_r v \) by an equivalent word produced by \( \text{Pinch}_{r-1} \).

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, \( \text{Positive} \) (twice), \( \text{OneToZero} \) and \( \text{Pinch}_{r-1} \) 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 \( O(\ell(w)^{4r(r-2)}) \), \( \text{CutRank} \), halts in \( O(\ell(w)^{4r(r-1)}) \) time.

Correctness of \( \text{Pinch}_{r-1} \) implies correctness of \( \text{FinalPinch} \), for \( r \geq 2 \).

2: If \( u A_r v(0) < 1 \), then it is outside the domain of \( A_r^{-1} \).

4: \( u A_r v \) 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, for \( r \geq 2 \).
\( \hat{\omega} \) Input a word \( w = A_{r-1}^{-1}uA_{r}v \) with \( \eta(u) = \eta(v) = 0, u \neq \epsilon \) and \( \text{rank}(u) < r - 1 \).
\( \hat{\omega} \) Either declare \( w \) invalid or return a word \( A_{0}^{\prime}v \sim w \) of length at most \( \ell(w) - 2 \).
\( \hat{\omega} \) Halt in \( O((w)^{\frac{1}{r}+\epsilon}) \) time.

run Positive\((A_{0}^{\prime}uA_{r}v)\) to decide among the following cases

- if \( A_{0}^{\prime}uA_{r}v \) is invalid or \( uA_{r}v(0) < 1 \), return invalid
- if \( uA_{r}v(0) = 1 \), return OneToZero\((w)\)

we now have that \( uA_{r}v \) is valid and \( uA_{r}v(0) > 1 \)

6: run Positive\((v)\) to determine whether \( v(0) < 0, v(0) = 0, v(0) > 0 \)

if \( v(0) < 0 \), return invalid

9: if \( v(0) = 0 \)

- if \( r = 2 \), run BasePinch\((A_{r-1}^{-1}uA_{r}v)\)

12: if it returns invalid, return invalid.

else return its result \( A_{0}^{\prime}v \), which will satisfy \( \ell(A_{0}^{\prime}v) \leq \ell(w) - 2 \)

if \( r > 2 \), run Pinch\(_{r-1}(A_{r-1}^{-1}uA_{r-1}v)\)

15: if it returns invalid, return invalid

else it returns \( A_{0}^{\prime}v \) for some \( l \) \( \leq \ell(u) \)

if \( l \leq 0 \), return invalid

18: run BasePinch\((A_{r}^{-1}A_{0}^{\prime}A_{r}v)\)

if it returns invalid, return invalid

else it returns \( A_{0}^{\prime}v \) for some \( |l| \leq |l| - 1 | = l - 1 \),

in which case return \( A_{0}^{r+1}v \)

if \( v(0) > 0 \)

24: run Pinch\(_{r-1}(A_{r-1}^{-1}uA_{r-1}A_{r}^{-1}v)\)

if it returns invalid, return invalid

else it returns \( A_{0}^{\prime}A_{r}v \) for some \( |l| \leq \ell(w) \)

27: run BasePinch\((A_{r}^{-1}A_{0}^{\prime}A_{r}A_{0}^{-1}v)\)

if it returns invalid, return invalid

else it returns \( A_{0}^{\prime}v \) for some \( |l| \leq |l| \),

in which case return \( A_{0}^{r+1}v \)

11: If \( r = 2 \), the rank of \( u \) is zero, so BasePinch applies.

13: \( \ell(A_{0}^{\prime}v) \leq \ell(w) - 2 \) by properties of BasePinch.

16: \( w \sim A_{0}A_{r}^{-1}A_{r-1}^{-1}uA_{r-1}v \) 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 \( uA_{r}v(0) > 1 \). So if \( A_{0}^{-1}uA_{r-1}v \) is invalid, then so is \( w \). And if Pinch\(_{r-1} \) gives us that \( A_{r}^{-1}uA_{r-1}v \sim A_{0}^{\prime}v \), then \( w \sim A_{0}A_{r}^{-1}A_{0}^{\prime}v \).

17: If \( l \leq 0 \), then \( w \) is invalid because \( A_{0}^{\prime}v(0) \leq 0 \) and lies outside of the domain of \( A_{r}^{-1} \) (since \( r \geq 2 \)).

19: Next, working from \( w \sim A_{0}A_{r}^{-1}A_{0}^{\prime}v \) established in our comment above on line 16,
we get that \( w \sim A_{0}A_{r}^{-1}A_{0}^{\prime}A_{r}v \) because \( A_{0}^{-1}A_{r}v \sim v \), given that \( r \geq 2 \) and \( v(0) = 0 \). So, if BasePinch tells us that \( A_{r}^{-1}A_{0}^{\prime}v \) is invalid, then so is \( w \).

20: \( |l - 1| = l - 1 \) here because \( l \) > 0 here.

21: Similarly, if \( A_{0}^{-1}A_{r}v \sim A_{0}^{\prime}v \), then \( w \sim A_{0}^{r+1}v \). 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 \( \ell(A_{0}^{r+1}v) \leq \ell(w) - 2 \), as required.
That Pinch word with a suitable correctness of above on lines 11 and 14.

Correctness of above on lines 11 and 14.

by either using

Correctness of above on lines 11 and 14.

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

4: \( A_0^{-1}A_0(0) = 0 \) (since \( r \geq 2 \)), so \( w_2A^{-1}A_r \sim w_2. \)

6: \( A_0^f \sim A_0^f w_2A_0^{-1}A_r \) 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 \( w_2A_1 \sim w_2. \)

10: \( A_0^f \sim A_0^f w_2A_1 \sim A_1^f w_2 \) 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.
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 $O(\ell(w)^{k+2})$ time.

express $w$ as $w_1A_r^{-1}w_2$ where $r \geq 1$ and $\eta(w_2) = 0$
(i.e. locate rightmost $A_r^{-1}, A_r^{-2}, \ldots, A_r^{-k}$ in $w$)
3:

15: The length of $w''$ is at most $\ell(w) - \ell(w_1) - 2 \leq \ell(w) - \ell(w_1) + 2k$ by properties of Pinch.
16: If $w_3(0) < 0$, then $w$ is invalid because $s \geq 2$
17: In this case $A_r^{-1}w_3A_0A_w4 \sim A_r^{-1}w_3A_1w_4$ since $A_0A_r(0) = A_r(0)$. As required, if $w'' \neq \text{invalid}$, it has length at most $\ell(A_r^{-1}uA_0w) = \ell(w) - \ell(w_1) + 1 < \ell(w) - \ell(w_1) + 2k$ and contains no $A_r^{-1}, A_r^{-2}, \ldots, A_r^{-k}$ by the properties established for Pinch.
18: Similarly, in this case $A_r^{-1}w_3A_0A_w4 \sim A_r^{-1}w_3A_1w_4$ since $A_r(0) = A_r(0)$, and the output has the required properties.
19: If $w_3(0) > 0$, then $A_r^{-1}w_3A_0w_4$ and $A_r^{-1}w_3A_{r-1}A_rA_0^{-1}w_4$ are equivalent by Lemma 2.4.

As $u(0) - 1 \geq 0$, and so is in the domain of $A_r$, the word $A_rA_0^{-1}v$ is valid. And, as $A_rA_0^{-1}v(0) = A_r(u(0) - 1) > 0$, we may replace the $A_{r-1}$ by $A_{r-2}A_{r-1}A_0^{-1}$ to get another equivalent word. Indeed, we may repeat this process $s - r \leq k$ times, to yield an equivalent word

$$A_r^{-1}w_3A_1A_rA_0^{-1}A_{r+1}A_0^{-1} \cdots A_rA_0^{-1}w_4$$
of length $\ell(w) - \ell(w_1) + 2(s - r)$ where $2(s - r) \leq 2k$ because $s, r \leq k$. Applying Pinch, then returns (if valid) an equivalent word

$$w'' = A_r^{-1}A_rA_0^{-1}A_{r+2}A_0^{-1} \cdots A_rA_0^{-1}w_4$$
where $\ell(w'') \leq \ell(w) - \ell(w_1) + 2(s - r) - 2 \leq \ell(w) - \ell(w_1) + 2k$.
20: If the suffix $A_r^{-1}w_3A_0w_4$ of $w$ is invalid, then $w$ is invalid.
21: By the above $\ell(w'') \leq \ell(w) - \ell(w_1) + 2k$, we have that $w' \sim A_r^{-1}w_3A_1w_4 = A_r^{-1}w_2$ and $\eta(w'') = 0$. It follows that $w \sim w_1w''$ and $\ell(w_1w'') = \ell(w_1) + \ell(w'') \leq \ell(w_1) + \ell(w) - \ell(w_1) + 2k \leq \ell(w) + 2k$, as required. Also, again evidently, $\eta(w'') = \eta(w) - 1$. 

if $w'' = \text{invalid}$, return invalid
21: else return $w' := w_1w''$
**Reduce** halts in \(O(\ell(w)^{3+4(k-1)})\) time since **Pinch**, 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 \(\text{sgn}(w)\).

**Algorithm 2.9 — Ackermann.**

\[
\begin{align*}
&\text{Input a word } w. \\
&\text{Return whether } w \text{ is valid and if it is, return } \text{sgn}(w(0)). \\
&\text{Halt in } O(\ell(w)^{4+k}) \text{ time.}
\end{align*}
\]

\[
\begin{align*}
\text{if } \eta(w) > 0, \text{ run } \text{Reduce} \text{ successively until} & \\
\text{it either returns that } w \text{ is invalid,} & \\
or it returns some } w' \sim w \text{ with } \eta(w') = 0 & \\
\text{return the result of } \text{Positive}(w')
\end{align*}
\]

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 \(O(\ell(w)^{3+4(k-1)})\) time in the lengths of their inputs, **Ackermann** halts in \(O(\ell(w)^{4+k})\) time.

3. **Efficient calculation with \(\psi\)-compressed integers**

3.1. **\(\psi\)-functions and \(\psi\)-words.** Similarly to Ackermann functions in Section 2.1, we define **\(\psi\)-functions** by

\[
\begin{align*}
\psi_1 &: \mathbb{Z} \to \mathbb{Z} \quad n \mapsto n - 1 \\
\psi_2 &: \mathbb{Z} \to \mathbb{Z} \quad n \mapsto 2n - 1 \\
\psi_i &: -\mathbb{N} \to -\mathbb{N} \\
\psi_i(0) &= -1 \quad \forall i \geq 1 \\
\psi_{i+1}(n) &= \psi_i(\psi_{i+1}(n) - 1) \quad \forall n \in -\mathbb{N}, \forall i \geq 2.
\end{align*}
\]

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:

\[
\begin{array}{ccccccc}
\cdots & n & \ldots & -4 & -3 & -2 & -1 & 0 \\
\cdots & n - 1 & \ldots & -5 & -4 & -3 & -2 & -1 & \psi_1 \\
\cdots & 2n - 1 & \ldots & -9 & -7 & -5 & -3 & -1 & \psi_2 \\
\cdots & 2 - 3 \cdot 2^n & \ldots & -46 & -22 & -10 & -4 & -1 & \psi_3 \\
\cdots & \vdots & \vdots & -1 - 3 \cdot 2^5 & -95 & -5 & -1 & \psi_4 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \psi_i \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \psi_{i+1} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}
\]

The following proposition explains why we defined \(\psi\)-functions with the given domains. It details the key property of \(\psi\)-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 $t^i a_i \in H_k t^r$ in $G_k$ if and only if $r$ is in the domain of $\psi_i$ 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 $t^{-1} a_1 t = a_1$,

$$t^r a_1 = a_1 t \quad t^{-1} \in H_k t^{-1} = H_k t^{\psi_i(r)}$$

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,

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

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

$$t^r a_i = a_i = a_2 t \quad t^{-1} \in H_k t^{-1} = H_k t^{\psi_i(0)},$$

since $a_2 t \in H_k$ and $\psi_i(0) = -1$. Finally, induction gives us that

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

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

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

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 $\psi_i$. Now, given that $r$ is in the domain of $\psi_i$, we have that $t^r a_i \in H_k t^{\psi_i(r)}$ from the calculations earlier in our proof, and so $H_k t^{\psi_i(r)} = H_k t^r$, but this implies that $s = \psi_i(r)$ by Lemma 6.1 in [12].

For example, painful calculation can show that

$$t^{r-2} a_3 a_1 = (a_3 t)(a_2 t)(a_1 t)(a_1 t)^{r-1} \in H_k t^{r-1},$$

but Proposition 3.1 immediately gives:

$$t^{r-2} a_3 a_1 \in H_k t^{\psi_i(-2)} = H_k t^{r-1}.$$  

The following criterion for 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 $t$ can be derived from Proposition 3.1.

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

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

The connection between $\psi$-functions and hydra groups is also apparent in that they relate to the functions $\phi_i$ 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 $\psi$-functions.
Lemma 3.3.

\begin{align*}
(9) & \quad \psi_3(n) = 2n - 1 & \forall n \leq 0, \\
(10) & \quad \psi_3(n) = 2 - 3 \cdot 2^{-n} & \forall n \leq 0, \\
(11) & \quad \psi_i(-1) = -i - 1 & \forall i \geq 1, \\
(12) & \quad \psi_i(n) \geq \psi_{i+1}(n) & \forall i \geq 1, n \leq 0, \\
(13) & \quad \psi_i(n) > \psi_i(n - 1) & \forall i \geq 1, n \leq 0, \\
(14) & \quad n > \psi_i(n) & \forall i \geq 1, n \leq 0, \\
(15) & \quad \psi_i(m) + \psi_i(n) \geq \psi_i(m + n) & \forall n, m \leq -2, i \geq 2, \\
(16) & \quad |\psi_i(m) - \psi_i(n)| \geq \frac{1}{2} |\psi_i(n)| & \forall i \geq 3, m \neq n.
\end{align*}

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 \(|n| > |m|\)),

\[
|\psi_3(m) - \psi_3(n)| = |3 \cdot 2^{-m} - 3 \cdot 2^{-n}| \geq |3 \cdot 2^{-m} - 3 \cdot 2^{-n-1}| = \frac{1}{2} \cdot 3 \cdot 2^{-n}
\]

\[
\geq \frac{1}{2} \cdot 3 \cdot 2^{-n} - 1 = \frac{1}{2}(3 \cdot 2^{-n} - 2) = \frac{1}{2} |\psi_3(n)|,
\]

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

\[
|\psi_3(m) - \psi_3(n)| = |\psi_3(n) - \psi_3(m)| \geq \frac{1}{2} |\psi_3(m)| \geq \frac{1}{2} |\psi_3(n)|,
\]

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. \(\Box\)

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

\[
\begin{align*}
\psi_1^{-1} & : \mathbb{Z} \to \mathbb{Z} \quad n \mapsto n + 1, \\
\psi_2^{-1} & : 2\mathbb{Z} + 1 \to \mathbb{Z} \quad n \mapsto (n + 1)/2, \\
\psi_3^{-1} & : \text{Img} \psi_3 \to \mathbb{N} \quad n \mapsto \psi_3^{-1}(n).
\end{align*}
\]

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

\[
\eta(f) := \#\{ i \mid 1 \leq i \leq n, f_i = \psi_j^{-1} \text{ for some } j \geq 2 \}.
\]

If \(f_{j-1} \cdots f_1(0)\) is in the domain of \(f_i\) for all \(2 \leq j \leq n\), then \(f\) is valid and represents the integer \(f(0)\). When \(f\) is non-empty, \(\text{rank}(f)\) denotes the highest \(i\) such that \(\psi_i^{-1}\) is a letter of \(f\). As in Section 2.1 we say two words \(w(\psi_0, \ldots, \psi_k)\) and \(w'(\psi_0, \ldots, \psi_k)\) are equivalent and write \(w \sim w'\) if either both \(w\) and \(w'\) are invalid, or are both valid and \(w(0) = w'(0)\).

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

\[
r^{-3} a_2^{-1} a_1 \in H_2^{\rho(\psi_3^{(-3)})}
\]

if \(-3 \in \text{Img} \psi_2\) and \(\psi_3^{(-3)}\) is in the domain of \(\psi_1\)—in other words, if \(\psi_1 \psi_2^{-1} \psi_3^{(-3)}\) is valid.

In fact these provisos are met: \(\psi_2^{(-3)} = -1\) and \(\psi_1^{-1} = -2\), so \(r^{-3} a_2^{-1} a_1 \in H_2^{\rho^2}\).

And, given that \(H_1^{\rho^3} = H_1\) if and only if \(r = 0\) by Lemma 6.1 in [12], determining whether \(r^{-3} a_2^{-1} a_1 \in H_2\) amounts to determining whether \(\psi_1 \psi_2^{-1} \psi_3^{(0)} = 0\). (In fact it equals \(2\), as we just saw, so \(r^{-3} a_2^{-1} a_1 \not\in H_2\).) This suggests that efficiently testing validity of \(\psi\)-words and when valid, determining whether a \(\psi\)-word represents zero, will be a step towards a polynomial time algorithm solving the membership problem for \(H_k\) in \(G_k\). (Had \(\psi_1 \psi_2^{-1} \psi_3^{(0)}\) been invalid, we could not have immediately concluded that that \(r^{-3} a_2^{-1} a_1 \not\in H_2\) or indeed
3.2. An example. Let

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

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

1. First we locate the rightmost \(\psi_i^{-1}\) in \(f\) with \(i \geq 2\), namely the \(\psi_2^{-1}\), and look to 'cancel' it with the first \(\psi_2\) to its right. In short, this is possible because

\[ (2x-1) - 2 - 1 = x - 1, \]

allowing us to replace \(\psi_2^{-1}\psi_2\) with \(\psi_1\) 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 \(\psi_i^{-1}\) with \(i \geq 2\), namely the \(\psi_3^{-1}\) and we look to 'cancel' it with the \(\psi_3\) to its right. To this end we first reduce the rank of the subword between the \(\psi_3^{-1}\) and \(\psi_3\) (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 \(\psi_3^{-1}\psi_2^{-1}\psi_1^{-1}\) for \(\psi_3^{-1}\) is legitimate by Lemma 3.5 and

\[ \psi_1\psi_2\psi_3(\psi_2\psi_3)^2\psi_1\psi_1^{-1} \sim f. \]

By Lemma 3.5, cancelation of the \(\psi_4^{-1}\) with \(\psi_1, \psi_2^{-1}\) with \(\psi_2\), and then \(\psi_3^{-1}\) with \(\psi_3\) then gives

\[ \psi_4(\psi_2\psi_3)^2\psi_1\psi_1^{-1} \sim f. \]

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 \(\psi_k\).

**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 \textbf{Bounds}: replace $A_i$ by $\psi_{i+1}$, calculate values of $\psi_i(n)$ for $n \leq -2$, and use the recursive relation for $\psi_i$-functions. The correctness argument for \textbf{BoundsII} is virtually identical to that for \textbf{Bounds}.

Similarly to \textbf{Ackermann, Psi} works right-to-left through a $\psi$-word eliminating letters $\psi_{r-1}$ 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 $\psi$-word. Once all have been eliminated, giving a $\psi$-word $f$ with $\eta(f) = 0$, a subroutine \textbf{PositiveII} determines the validity of $f$.

\begin{algorithm}
\caption{\textbf{PositiveII}}
\begin{itemize}
\item Input a $\psi$-word $f$ with $\eta(f) = 0$.
\item Either return that $f$ is invalid, or that $f$ is valid and declare whether $f(0) > 0$, $f(0) = 0$, or $f(0) < 0$.
\item Halt in time $O(\ell(f)^3)$.
\end{itemize}
\end{algorithm}

\textbf{PositiveII} can be constructed analogously to \textbf{Positive} with the following changes:

1. The role of $\psi_i$ corresponds to the role of $A_{i-1}$.
2. Unlike Ackermann functions, $\psi_i : -\mathbb{N} \rightarrow -\mathbb{N}$, so appropriate signs and inequalities need to be altered.
3. We still evaluate letter-by-letter. However, in place of using \textbf{Bounds} to check whether an evaluation by $A_i$ is above some (positive) threshold, we use \textbf{BoundsII} to check that $\psi_k$ 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 \textbf{PositiveII} can be justified similarly to \textbf{Positive}.

Next \textbf{BasePinchII} processes words of the form $\psi_k^{-1}\psi_1\psi_l\psi_v$. We make one major change: we have a stricter bound that \textbf{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 \textbf{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 \textbf{PinchII} from growing too large.
Algorithm 3.3 — BasePinchII

○ Input a word \( f = \psi_{r}^{-1} w_{r}, v \) with \( r \geq 2, \) \( \text{rank}(v) \leq 1, v \) a \( \psi \)-word, and \( \eta(v) = 0. \)

○ Either return invalid when \( f \) is invalid or return a word \( f' = \psi_{r}^{-1} v \sim f \) 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 \( O(\ell(f)^{3}). \)

\[
l := -u(0) \text{ (so } \psi_{1} \text{ is } u \text{ with all } \psi_{1}^{+1} \psi_{1}^{+1} \text{ subwords removed and } \psi_{r}^{-1} \psi_{r}^{-1} \psi_{r}^{-1} v \sim f)
\]

3: if PositiveII(\( \psi_{r}, v \)) = invalid, return invalid

4: if \( l = 0, \) return \( f' := v \)

5: if \( r = 2, \) return \( f' := \psi_{1}^{1/2} v \) or invalid depending on whether \( l \) is even or odd

6: we now have \( l \neq 0 \) and \( r > 1 \)

7: run PositiveII(\( \psi_{r}, v \)) to determine if \( \psi_{r} v(0) \geq 0 \) (so outside the domain of \( \psi_{r}^{-1} \))

8: if so, return invalid

9: run PositiveII(\( \psi_{r}^{-1} \psi_{r} v \)) to determine whether \( \psi_{r} v(0) < 2l \)

10: if so, return invalid

11: we now have that \( 0 > v(0) \geq -|l| \) and \( 0 > \psi_{r} v(0) \geq -2|l| \) and \( \psi_{r} v(0) - l \geq -3|l| \)

12: calculate \( v(0) \) by running PositiveII(\( \psi_{r}^{-1} v \)) for \( r = 0, 1, \ldots, |l| \)

13: run BoundsII(3 \( |l| \))

14: search the output of BoundsII(3 \( |l| \)) to find \( \psi_{r} v(0) \)

15: set \( m := \psi_{r} v(0) - l \)

16: search the output of BoundsII(3 \( |l| \)) for \( c \) with \( \psi_{r}(c) = m \) (so \( c = \psi_{r}^{-1} \psi_{r}^{1/2} v, v(0) = w(0) \))

17: if such a \( c \) exists, return \( f' := \psi_{1}^{v(0)-1} v \)

else return invalid

The main differences between BasePinchII and BasePinch are that subroutines by their \( \psi \)-versions, indices are shifted by one, signs and inequalities are adjusted to reflect that \( \psi_{r} : [N] \rightarrow [N] \) (for \( r \geq 3 \)) and that \( \psi_{r}(n) = n - 1 \) (in contrast to \( A_{0}(n) = n + 1 \)), and 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:

\[ |A_{r}(m) - A_{r}(p)| \geq \frac{1}{2} A_{r}(m) \]

which holds for all \( r \geq 2 \) and \( m \neq p. \)

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 \( \psi_{r}^{-1} \) and \( \psi_{r}, \) reducing length by 2.

For \( l \neq 0 \) and \( r = 2, \) 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 \( w_{r} v(0). \) For validity, we need \( w_{r} v(0) \) to be odd, and this is sufficient since \( \psi_{2}^{-1}(n) = (n + 1)/2. \) When \( l \) is even, return the equivalent word \( f' := \psi_{2}^{1/2} v. \) Otherwise \( f \) is invalid. The restrictions on the length of \( l \) follow directly from the fact that \( |l|/2 \leq |l| - 1 \) if \( l = 0. \)

For the case \( l \neq 0 \) and \( r > 2, \) consider the following: we claim that

\[ |\psi_{r}(n) - \psi_{r}(n - 1)| \geq |\psi_{3}(0) - \psi_{3}(-1)| = 3. \]
For $r \geq 3$, assume the result holds for all ranks less than $r$. We have:

$$|\psi_r(n) - \psi_r(n - 1)| = |\psi_{r-1}(\psi_r(n)) - \psi_{r-1}(\psi_r(n - 1))| \\
\geq |\psi_{r-1}(\psi_r(n)) - \psi_{r-1}(\psi_r(n - 1))| \geq |\psi_3(0) - \psi_3(-1)|$$

where the final two inequalities follow from the fact that $\psi_{r-1}$ is non-decreasing and the inductive hypothesis, respectively.

By extending this argument inductively and using that $\psi_r$ is non-decreasing:

$$|\psi_r(n) - \psi_r(n + m)| \geq 3m.$$ 

So, for $r > 3$ and $l \neq 0$ where $f' = \psi_0^{r-1}(\psi_0(0))$, 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 $\ell(f')$. \hfill $\square$

**OneToZeroII** is essentially the same as **OneToZero** with $A_0$ replaced by $\psi_1$.

---

**Algorithm 3.4 — OneToZeroII.**

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

---

**Proof that $\ell(f') \leq \ell(f) - 3$ in **OneToZeroII**.** Now $\nu(0) \leq 0$ since $\nu(0)$ is in the domain of $\psi_r$ and $r \geq 3$. Consider first the case $\nu(0) \leq -1$. First observe that $\psi_r(x) \leq x - 3$ when $x \leq -1$ and $r \geq 3$. Since $\eta(u) = 0$, $\psi_1^{-1}$ is the only letter it can contain which decreases the absolute value as $f(0)$ is evaluated. So, given that $uw\psi_rv(0) = -1$, $u$ must contain $\psi_1^{-1}$ at least $|\nu(0) - 3| - 1 = |\nu(0)| + 2$ times. So $\ell(u) \geq |\nu(0)| + 2$ and therefore

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

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

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

**PinchII**, is an analogue to **Pinch**. 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, for $r \geq 2$.**

- Input a word $f = \psi_r^{-1}uw\psi_rv$ with $r \geq 2$, $\text{rank}(u) \leq r - 1$, $v$ a $\psi$-word, and $\eta(v) = 0$.
- Either return that $f$ is invalid, or return a word $f' = \psi_1^rv$ equivalent to $f$ such that $\ell(f') \leq \ell(f) - 2$ if $u$ is empty and otherwise, $\ell(f') \leq \ell(f) - 3$.
- Halt in $O(\ell(f)^{2(r-1)})$ time.

---

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

1. We replace $A_r$ by $\psi_{r+1}$ for $r \geq 0$.
2. We replace all called subroutines by their $\psi$-word versions.
Before discussing the correctness of PinchII, we construct and analyze its subroutines CutRankII and FinalPinchII.

Algorithm 3.6 — CutRankII, for $r \geq 2$.
- Input a $\psi$-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 return $f' := \psi_r^{-1}v \sim f$, or return $f' := \psi_r^{-1}u'\psi_r v \sim f$ where rank($u'$) \leq r - 2 and $u' \neq e$. In all cases $\ell(f') \leq \ell(f)$ and if $f' := \psi_r^{-1}v$, then $\ell(f') \leq \ell(f) - 3$.
- Halt in $O((\ell(f))^{4(r-1)})$ time.

The construction of CutRankII is the same as CutRank, except that:

1. We replace $A_r$ by $\psi_{r+1}$ for $r > 0$, $A_0$ by $\psi_1^{-1}$. We replace all called subroutines by their $\psi$-word versions.
2. In line 5, check whether $w_0\psi_r v(0) = -1$. If so, run and return the result of OneToZeroII($w$).
3. In line 11, instead of the substitution $A_r = A_{r-1}A_rA_0^{-1}$ which encodes the defining recursion relation for Ackermann functions, use Lemma 3.5 and make the substitution $\psi_r^{-1} = \psi_r\psi_r^{-1}\psi_r^{-1}\psi_r^{-1}$ to convert $w$ to $\psi_r\psi_r^{-1}\psi_r^{-1}\psi_r^{-1}\psi_r^{-1}u'\psi_r^{-1}v$ where $\eta(u) = \eta(u') = 0$ and $u'$ has rank strictly less than $r - 1$.

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

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

CutRankII always returns $f' = f$ if $u$ has rank less than $r - 1$, so assume $\psi_{r-1}$ appears in $u$ so rank($u$) = $r - 1$. If $w_0\psi_r v(0) = -1$, then CutRankII calls and returns the output $\psi_1^{-1}v \sim w$ of OneToZeroII. The bound on the length of the output follows from the specifications of OneToZeroII. If $w_0\psi_r v(0) \neq -1$, then, as we show for CutRank, after each iteration of the loop, there is no increase in length. Indeed, express $f$ as $\psi_{r-1}u'\psi_{r-1}u''\psi_r v$ where $\eta(u') = \eta(u'') = 0$, rank($u'$) < $k - 1$ and rank($u''$) \leq $k - 1$. Substituting $\psi_r\psi_r^{-1}\psi_r^{-1}\psi_r^{-1}$ for $\psi_r^{-1}$ adds 3 letters. There is at least one letter between $\psi_r^{-1}$ and $\psi_{r-1}$, so applying PinchII$_{r-1}$ then decreases length by at least 3. Hence when CutRankII iterates the while loop without calling OneToZeroII, length does not increase. On the other hand, when an iteration of the while loop calls OneToZeroII, CutRankII halts and returns invalid or an $f' \sim f$, where $\ell(f) \leq \ell(f')$; the argument is the same as in the commentary for Line 6 in the proof of correctness of CutRank.

To adapt FinalPinchII, to give FinalPinchII:

1. In lines 1–3, check whether $w_0\psi_r v(0) \geq 0$ or $w_0\psi_r v(0) = -1$. In the former case, $\psi_r^{-1}w_0\psi_r v(0)$ is invalid, so $w$ is as well. In the latter case, run and return the result of OneToZeroII($f$).
2. In line 24, use Lemma 3.5 instead of Lemma 2.4 to make the analogous substitutions, $\psi_r^{-1} = \psi_r\psi_r^{-1}\psi_r^{-1}\psi_r^{-1}$ and $\psi_r = \psi_r\psi_r^{-1}\psi_r^{-1}$.
Algorithm 3.7 — FinalPinchII, for \( r \geq 2 \).

- Input a word of the form \( f := \psi_r^{-1}u\psi_r v \) with \( \eta(u) = \eta(v) = 0, u \neq \epsilon \) and \( \text{rank}(u') < r - 1 \).
- Either return \text{invalid} or return an equivalent word \( \psi^1v \) of length at most \( \ell(f) - 3 \).
- Halt in \( O(\ell(f)^{3(r-1)}) \) time.

Correctness of FinalPinchII, assuming correctness of PinchII,−1. Consider the special cases:

- \( u\psi_r v(0) = -1 \): FinalPinchII calls OneToZeroII and returns the output \( \psi^0v \sim f \). The bounds on the length of the output are established by the specifications of OneToZeroII.
- \( v(0) = 0 \): substituting \( \psi_1\psi_r^{-1}\psi_{r-1}^{-1}\psi_r^{-1} \) for \( \psi_r^{-1} \) adds 3 letters. Substituting for \( \psi_r \) by \( \psi_{r-1} \) results in no increase in length in this case. As in CutRankII, \( r \), the substitution for \( \psi_{r-1}^{-1} \) ensures that there is at least one letter between \( \psi_1^{-1} \) and \( \psi_{r-1} \), so if the subroutine PinchII,−1 (if \( r-1 > 2 \)) or BasePinchII (if \( r-1 = 2 \)) returns an equivalent word, that word is at least 3 letters shorter than the word that subroutine was run on (by the specifications of PinchII,−1 or BasePinchII).
- \( u\psi_r v(0) < -1 \) and \( v(0) < 0 \): substituting \( \psi_1\psi_{r-1}\psi_r\psi_{r-1}^{-1} \) and \( \psi_1\psi_r^{-1}\psi_{r-1}^{-1}\psi_1^{-1} \) for \( \psi_r \) and \( \psi_r^{-1} \), respectively, adds 6 letters. Applying PinchII,−1 to

\[
\psi_r^{-1}\psi_{r-1}^{-1}u\psi_1\psi_{r-1}\psi_r\psi_1^{-1}v,
\]

whose length is at most \( \ell(f) + 6 \). There are non-trivial letters between \( \psi_r^{-1}v \) and \( \psi_{r-1}v \). So the equivalent word returned by PinchII,−1 is at least three letters shorter. Therefore, the result is of the form

\[
\psi_1\psi_r^{-1}\psi_1^{r-1}v
\]

for some \( l \in \mathbb{Z} \) and has length at most \( \ell(f) + 3 \). If \( l = 0 \), running BasePinchII triggers a trivial case where \( f' = v \) is returned and \( \ell(v) \leq \ell(f) - 3 \) since \( u \) is non-empty. Otherwise, applying BasePinchII to \( \psi_r^{-1}\psi_1\psi_r\psi_1^{-1}v \), if an equivalent word of the form \( \psi_r^{r-1}v \) is returned, its length is 4 letters shorter than the input to BasePinchII. Hence we have a word equivalent to \( f \) of the form

\[
\psi_1\psi_r^{-1}v
\]

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

\[
\psi_r^{r-1}v
\]

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

Correctness of PinchII, assuming the correctness of PinchII,−1. Correctness can be proved by mimicking our proof of correctness for Pinch. The main difference is the bound on the length of any returned word. When \( r = 2 \), the bound on \( \ell(f') \) comes directly from the bound for BasePinchII.

Suppose \( r \geq 3 \). If \( w \) is a word and CutRankII,\( w \) returns a word \( w' \), then \( \ell(w') \leq \ell(w) \).

When \( u \) is the empty word and \( f = \psi_r^{-1}u\psi_r v \), PinchII,\( f \) returns \( w' := v \) which has length \( \ell(w') = \ell(f) - 2 \). On the other hand, if \( u \neq \epsilon \), then CutRankII,\( f \) returns a word \( \psi_r^{-1}u\psi_r v - f \) which has length at most \( \ell(f) \) and where \( u' \) is non-empty.

Finally, if FinalPinchII,\((\psi_r^{-1}u\psi_r v)\) returns an equivalent word, the output has length at most \( \ell(f) - 3 \) by the specifications of FinalPinchII. \( \square \)
Correctness and construction of ReduceII are nearly immediate by following those of Reduce, replacing $A_i$ by $\psi_{i+1}$ and changing the subroutines to the $\psi$-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 $\psi$-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 $O(\ell(f)^{3+\ell(k-1)})$ time.

Finally, $\Psi$ can be constructed similarly to Ackermann by replacing all $A_i$ by $\psi_{i+1}$ 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 $\psi$-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 $O(\ell(f)^{k+1})$ 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 Member, which, given a word $w = w(a_1, \ldots, w_k, t)$ on the generators of the hydra group

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

where $\theta(a_i) = a_{i-1}$ for all $i > 1$ and $\theta(a_1) = a_1$, will tell us whether or not $w$ represents an element of $H_k = \langle a_1t, \ldots, a kt \rangle$.

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

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

and that $r'v$ can be produced in time $O(\ell(w)^2)$.

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 

\[ \theta^k(a_k) = \theta^{k-1}(\theta(a_k)) = \theta^{k-1}(a_k)\theta^{k-1}(a_{k-1}), \]

we have

\[
\begin{align*}
f(n, k) &= f(n-1, k) + f(n-1, k-1) \\
&= f(0, k) + f(0, k-1) + \cdots + f(n-1, k-1) \\
&\leq 1 + t^{k-2} + \cdots + n^{k-2} \\
&\leq (n+1)^{k-1}
\end{align*}
\]

where the first inequality uses \( f(0, k) = \ell(\theta^k(a_k)) = \ell(a_k) = 1 \) 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 \( \theta^{-(n)}(a_k) = a_k \theta^{-(n)}(a_{k-1}) \) because \( \theta(a_k) = a_k a_{k-1} \). So, for all \( n \in \mathbb{Z} \)

\[ \theta^{-(n)}(a_k) = \theta^{-(n-1)}(a_k) = \theta^{-(n-1)}(a_k) \theta^{-(n-1)}(a_{k-1}) = \theta^{-(n-1)}(a_k) \theta^{-(n-1)}(a_{k-1}) \]

and therefore

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

So for all \( n > 0 \)

\[
\begin{align*}
g(n, k) &\leq g(n-1, k) + g(n, k-1) \\
&\leq g(0, k) + g(1, k-1) + \cdots + g(n, k-1) \\
&\leq 1 + t^{k-2} + \cdots + (n+1)^{k-2} \\
&\leq (n+1)^{k-1}
\end{align*}
\]

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

Next \textbf{Member} calls the subroutine \textbf{Push}, 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 \( a_1, \ldots, a_{lt} \). The powers of \( t \) that occur as this proceeds are recorded by \( \psi \)-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 \( a_{lt}^{\pm 1} \) at a time. (There are words representing elements of \( H_L \) for which that is impossible.) Instead, we first consider the locations of the \( a_{lt}^{\pm 1} \), then the \( a_{lt-1}^{\pm 1} \), and so on. Following [12], we define the \textit{rank-} \( k \) \textit{decomposition of} \( v \text{ into pieces} \) as the (unique) way of expressing \( v \) as a concatenation \( \pi_1 \cdots \pi_p \) of the minimal number of subwords (‘pieces’) \( \pi_i \) of the form \( a_{lt}^{\epsilon_1} a_{lt-k}^{\epsilon_2} \) where \( \arg\min_{u \leq k-1} \epsilon_1, \epsilon_2 \in \{0, 1\} \). For example, the rank-5 decomposition of

\[ a_5 a_3 a_2 a_1 a_5^{-1} a_1 a_5^{-1} \]

is

\[ (a_5 a_3 a_2 a_1 a_5^{-1})(a_2)(a_5 a_1 a_5^{-1})(a_1 a_5^{-1}). \]

We use pieces because \( t^r v \in H_L t^r \) 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 \( H_L \). More precisely, \( t^r v \in H_L t^r \) if and only if there exists a sequence \( r = r_0, \ldots, r_p = s \) such that \( t^r \pi_{r+1} \in H_L t^{r+1} \) (Lemma 6.2 of [12]).

Let \( f_0 := \psi_{\ell}^{\ell}, \) so \( f_0(0) = r \). Then, for each successive \( \ell \), we determine, using the subroutine \textbf{Piece}, whether or not there exists \( r_1 \in \mathbb{Z} \) (unique if it exists) such that

\[ f_0(\ell)^{\ell+10} \pi_1 \in H_L t^r \]

and if so, it gives a \( \psi \)-word \( f_1 \) such that \( f_1(0) = r_1 \). \textbf{Piece} expresses \( \pi_i \) as \( a_{lt}^{\epsilon_1} a_{lt-k}^{\epsilon_2} \) 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.’ **Piece** has two subroutines. The first, **Front**, reduces the problem of whether \( r_i \) exists to determining whether, for a certain \( \psi \)-word \( f_{i-1}' \) and a certain rank-\( k \) piece \( \pi' \) which does not have \( a_k \) as its first letter, there exists \( r'_i \in \mathbb{Z} \) such that \( f_{i-1}' \pi' \in H_{k-1}r'_i. \) Then the second, **Back**, makes a similar reduction to a situation when there is no \( a_k' \) at the end. It then inductively calls **Push** on the modified piece (which is now a word of rank less than \( k \)) to find a \( \psi \)-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 **PSI**) an emerging \( \psi \)-word not being valid, or noticing that \( \pi \) fails to have a suffix or prefix of a particular form.

This inductive construction has base cases **Push** and **Piece**, which use elementary direct manipulations.

If \( r_1, \ldots, r_p \) all exist, then **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 \( w = a_3^4a_2a_1a_3^3a_2^2a_1^2a_3 \). As we saw in Section 1.4, \( w = u_{5,4}(a_2t)(a_1t)(a_2t)^{-1}u_{4,3}^{-1} \) in \( G_1 \) which has length \( 2|H_0(4)| + 3 = 2^{47} \cdot 3 - 1 \) as a word on the generators \( a_1t, a_2t, a_3t \) of \( H_3 \). Here is how our algorithm **Member** 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 \( a_i t = t \theta(a_i) \) in \( G_3 \) for all \( i \) to shuffle the \( t \) to the front. This produces
   \[
   v = \theta(a_3)^4 \theta(a_2) a_2^3 a_1 = (a_3a_2)^4 a_2^2 a_1^2 a_3.
   \]

2. Define \( f_0 := \psi_1^{-1} \), to express the power \( f_0(0) = 1 \) of \( t \) here.

3. The rank-3 decomposition of \( v \) into pieces is:
   \[
   v = (a_3a_2)(a_3a_2)(a_3a_2)(a_3a_2)(a_3a_2)^{-1}(a_3^{-1})(a_3^{-1})(a_3^{-1}).
   \]

   Accordingly, define
   \[
   \pi_1 := \pi_2 := \pi_3 := a_3a_2, \quad \pi_4 := a_3a_2^2, \quad \pi_5 := \pi_6 := \pi_7 := a_3^{-1}.
   \]

   The subroutine **Push** now aims to find \( \psi \)-words \( f_1, \ldots, f_7 \) such that \( f^{(0)} \pi_i \in H_3f^{(0)} \) for \( i = 1, \ldots, 7 \), by ‘pushing the power of \( t \) through successive pieces.’

4. First the subroutine **Piece** is called to try to pass \( f^{(0)} = t \) through \( \pi_1 \). Since the power of \( t \) is positive, the subroutine **Front** calls a further subroutine **Prefix** to find the longest prefix (if one exists) of \( \pi_1 \) of the form \( \theta^{-1}(a_3)a_2 \) for some \( i \geq 1 \). **Prefix** does so by generating \( \theta^i(a_3)a_2, \theta(a_3)a_2, \) and so on, and comparing, until the length of \( \pi_1 \) is exceeded. In this instance **Prefix** returns \( i = 1 \). It follows from the Piece Criterion that \( f^{(0)} \pi_1 = a_3t \in H_3f^{(0)} = H_3f^{(0)} \psi_1 \). Accordingly define \( f_1 := \psi_1 \psi_1^{-1} \).

5. **Piece**, next looks to pass \( f^{(0)} = t^0 = 1 \) through \( \pi_2 \). **Front** uses **PSI** to check that \( f_1(0) = 0 \leq 0 \). By the Piece Criterion, it then follows by Proposition 3.1 (since the power of \( t \) is non-positive) from the fact that there are no inverse letters in \( \pi_2 \) that \( a_3a_2 \in H_3f^{(0)} \). So define \( f_2 := \psi_2 \psi_3 \psi_1 \psi_1^{-1} \).

6. Next **Piece** tries to pass \( f^{(0)} \) 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 **Piece** tries to pass \( f^{(0)} \) through \( \pi_4 \).

   7.1. **Front** uses **PSI** to check that \( f_3(0) \leq 0 \). It follows that \( f^{(0)} \pi_4 \in H_3f^{(0)} \) and the problem is reduced (by the Piece Criterion) to finding an \( s \in \mathbb{Z} \) (if one exists) such that
   \[
   f^{(0)}(0)a_2^2a_1a_3^{-1} \in H_3f^{(0)}.
   \]
This will represent progress as (unlike \( \pi_d \)) \( a_2^3a_1^2a_2^{-1}a_3^{-1} \) is a piece without an \( a_0 \) at the front.

7.2. Then the subroutine \textbf{Back}, recursively calls \textbf{Piece} to find the \( s \in \mathbb{Z} \) (if there is one) such that \( t^{\theta(t)}a_2^3a_1^2a_2^{-1} \in H_{t^{\theta(t)}} \). It returns \( \psi_1^{-1}(\theta_1)\psi_2^3\psi_3f_3 \). (We omit the steps \textbf{Piece} goes through.) \textbf{Back}, then uses \textbf{Psi} to test whether \( f_4 := \psi_1^{-1}(\theta_1)\psi_2^3\psi_3f_3 \) is valid, which it is: we examined it in Section 3.2. Also \textbf{Psi} declares that \( f_4(0) \leq 0 \) It follows (using the Piece Criterion) that \( t^{\theta(t)}_{\pi_4} \in H_{t^{\theta(t)}} \).

8. Next \textbf{Piece}, tries to pass \( t^{\theta(t)} \) through \( \pi_3 \). This is done by \textbf{Back}. By the Piece Criterion, it suffices to check that \( f_5 := \psi_3^{-1}f_4 \) is valid, which is done using \textbf{Psi}.

9. \textbf{Piece}, likewise passes \( t^{\theta(t)} \) through \( \pi_6 \) giving \( f_6 := \psi_3^{-2}f_4 \), and then \( t^{\theta(t)} \) through \( \pi_7 \) giving \( f_7 := \psi_3^{-3}f_4 \).

10. Finally, let \( g := f_7 \). We have that \( w = tv \in H_{t^{\theta(t)}} \). So use \textbf{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 \textbf{Member}, handling such a situation.

\textbf{Example 4.3.} Let \( w = a_3a_2a_1t^2a_2^{-1}a_3^2a_1^{-1}a_3t^{-1}a_3^{-1} \). We will show how \textbf{Member}, discovers that \( w \in H_3 \).

1. Shuffle the \( t^1 \) in \( w \) to the front, applying \( \theta^1 \) to letters they pass, so as to convert \( w \) to the word \( t^2v \) representing the same element of \( G_3 \), where \( v = a_3a_2^3a_1^2a_2^{-1}a_3^2a_1^{-1} \).

   Let \( f = \psi_3^{3} \) so that \( f(0) = 2 \) records the power of \( t \).

2. Express \( v \) as its rank-3 decomposition into pieces: \( v = \pi_1\pi_2 \) where

   \[
   \pi_1 := a_3a_2^3a_1^2a_2^{-1}a_3^{-1}, \quad \pi_2 := a_1^2a_3^{-1}.
   \]

   Set \( f_0 := f \). \textbf{Push}_3 now looks for valid \( \psi \)-words \( f_1 \) and \( f_2 \) such that \( t^{\theta(t)}\pi_1 \in H_{t^{\theta(t)}} \) and \( t^{\theta(t)}\pi_2 \in H_{t^{\theta(t)}} \) by twice calling its subroutine \textbf{Piece}.

3. \textbf{Piece} calls \textbf{Front} to 'try to move \( t^{\theta(t)} \) past \( \pi_1 \). As \( a_3 \) is the first letter of \( \pi_1 \), \textbf{Front} calls \textbf{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 \( \pi_1 \) has a prefix \( \theta'(a_3) \) where \( i = 2 = 1 = 1 \).

   The idea is that if \( r > 0 \), then \( t^r\theta'(a_3) = a_3t^r \), and the only situation in which a \( t^2 \) can pass through the rank-3 piece \( \pi_1 \) is when \( \pi_1 \) has a prefix which is 'approximately' \( \theta'(a_3) = a_3a_2^2a_1 \)—more precisely, a prefix of \( \pi_1 \) is the prefix \( \theta(a_3) \) of \( t^2 \).

   The subroutine \textbf{Prefix} looks for this prefix by generating \( \theta'(a_3)a_3^2 = a_3a_2^2 \), then \( \theta'(a_3)a_3 = a_3a_2^3 \), then \( \theta(a_3)a_2 = a_3a_2^2a_3a_2 \), and so on, until the length of \( \pi \) is exceeded, and comparing with the start of \( \pi_1 \). Here, \( a_3a_2^2 \) and \( a_3a_2^3 \) are prefixes of \( \pi_1 \), but \( a_3a_2^2a_3a_2 \) is not, and \textbf{Prefix} returns \( i = 2 \).

4. Call \textbf{Psi} to check that \( i \) is at least \( f_0(0) = 2 \).

5. Intuitively speaking, as this prefix \( a_3a_2^3 \) is 'approximately' \( \theta'(a_3) \), the length of the 'correction' \( a_1a_1^{-1} \) that has to be made for the discrepancy between \( \theta'(a_3) \) and the prefix \( a_3a_2^3 \) is minimal compared to the length of the prefix that the power of \( t \) advances past. In this instance:

   \[
   t^2\pi_1 = t^2a_3a_2^3a_1^2a_2^{-1}a_3^{-1} = t^2\theta'(a_3)a_1a_1^{-1}a_1a_2^{-1}a_3^{-1} = (a_3t)a_1a_2^{-1}a_3^{-1},
   \]

   and so we 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 \( \psi_1f_0 \), as \( \psi_1f_0(0) = 1 \).

6. Next \textbf{Piece} calls \textbf{Back} on input \( a_1a_2^{-1}a_3^{-1} \) and \( \psi_1f \) 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 \textbf{Push}_3, which calls \textbf{Piece} to attempt to push \( t \) through \( a_1a_2^{-1} \). \textbf{Piece} calls \( \Psi \) to find out
whether $\psi^{-1}_2 \psi_1 f$ is valid. It is not, and it follows from the Piece Criterion that there is no $s \leq 0$ such that $ta_1a_2^{-1}a_3^{-1} \in H_t t'$.

8. So, instead **Piece**, searches for an $s > 0$ such that $ta_1a_2^{-1}a_3^{-1} \in H_t t'$ or, equivalently, $ta_2a_3^{-1} \in H_t t$.

9. We check for $s = 1, 2, \ldots$ whether we can move $t'$ 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, $ta_3a_2 = t\theta(a_3) = (a_1t) \in H_3$, the problem reduces to determining whether $t' \in H_t t$ or, equivalently, $ta_1 \in H_3 t'$. This shown to be the case by **Push**, which finds that $ta_1 = (a_1t) \in H_3$ and returns $\psi_1 f$, which satisfies $\psi_1 f(0) = 0$, to indicate the coset $H_3 t'$ of $H_3$. Finally, **Back**, checks that $H_3 t' = H_3 t' \psi_1 f$ by calling $\psi_1 f$, and returns $f_1 := \psi^{-1}_1 \psi_1 f$ (which satisfies $f_1(0) = 1$) to indicate that $\pi_1 \in H_3 t' f(0)$.

(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**, because the prefixes we check for must be no longer than that word.)

10. We now seek to pass $t' f(0)$ through $\pi_2$ by another call on **Piece**. Recall $\pi_2 = a_2^2a_1^{-1}$ and $f_1 := \psi^{-1}_1 \psi_1 f$, and $f_1(0) = 1$.

11. **Piece** first calls **Front** but the first letter of $\pi_2$ is not $a_3$, so **Front** does nothing.

12. **Piece** then calls **Back**. It first looks for $s \leq 0$ such that $t' f(0) \pi_2 \in H_t t'$, which it succeeds in finding as follows.

12.1. **Push** tries to pass $t' f(0)$ through $a_3^2$, which is elementary since $a_1$ commutes with $t$: $ta_3^{-1} = (a_1t)(a_1t)^{-1}$ and so **Push** returns $\psi_t f_1$, representing $\psi_t f_1(0) = -1$.

12.2. Call $\psi_1 f$ to check that $\psi_t f_1$ is valid. Then to pass $s$ through $a_3^{-1}$, call $\psi_1 f$ to check that $\psi_t f_1(0) = 1$. Return $f_2 := \psi^{-1}_t \psi_t f_1$ to indicate that $t' f(0) \pi_2 \in H_t t' f_2(0)$.

13. **Member** checks that $f_2(0) = 0$ and declares that $w \in H_3$.

These examples illustrate the tests **Member** 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 $t' v$ is in $H_t t'$ for some $s \in \mathbb{Z}$. A result we call the ‘Piece Criterion’ is at the heart of that and presentation of 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 $\theta(a_m^l)$, for $m \geq 1$ and $r \in \mathbb{Z}$, to refer to the freely reduced word on $a_1, \ldots, a_m$ equal to $\theta(a_m^l)$. The following lemma will be useful for calculating with iterations of $\theta$.

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

\[
\theta^r(a_m) = a_m \theta^r(a_{m-1}) \theta^r(a_{m-2}) \cdots \theta^r(a_{m-1})
\]

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

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

\[
\theta^r(a_m) = a_m \theta^{-1}(a_{m-1}) \theta^{-2}(a_{m-2}) \cdots \theta^{-1}(a_{m-1})
\]

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

\[
\theta^i(a_2) = a_2\theta^{-1}(a_1^i)\theta^{-2}(a_1^1)\cdots\theta^{-m}(a_1^1) = a_2a_1^i,
\]

and the result holds. For \( m > 2 \), the induction hypothesis tells us that the first letter of each subword \( \theta^{-i}(a_1^{m-1}) \) is \( a_1^{m-2} \) and the final letter is \( a_1^{m-1} \), 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 \( a_1^{m-1} \). □

The remainder of this section concerns words \( w \) expressed as

\[
w = \theta^{e_0}(a_0)\theta^{e_1}(a_1^i)\cdots\theta^{e_l}(a_l^{m-1})
\]

where \( e_x \in \{\pm 1\} \) for \( x = 0,\ldots,l+1 \), and \( a_i^j \neq a_{i+1}^{j+1} \) and

\[
e_{x+1} = \begin{cases} 
 e_x & \text{if } e_x = -e_{x+1} \\
 e_x - 1 & \text{if } e_x = e_{x+1} = 1 \\
 e_x + 1 & \text{if } e_x = e_{x+1} = -1
\end{cases}
\]

for \( x = 0,\ldots,l \). We refer to the \( a_0^{e_0},\ldots,a_l^{m-1} \) in the subwords \( \theta^{e_0}(a_0), \theta^{e_1}(a_1^i), \ldots, \theta^{e_l}(a_l^{m-1}) \) of \( w \) as the principal letters of \( w \).

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

**Proof.** The point of the hypotheses is that \( w \) is the word obtained by shuffling all \( t^{x+1} \) rightwards in

\[
\left\{ \begin{array}{ll}
 t^{-e_0}(a_0) t^{e_0} \cdots (a_{i+1}^j) t^{e_{i+1}} & \text{if } e_0 = 1 \\
 t^{-e_0-1}(a_0) t^{e_0} \cdots (a_{i+1}^j) t^{e_{i+1}} & \text{if } e_0 = -1,
\end{array} \right.
\]

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

Now \( (a_0 t^{e_0}) \cdots (a_{i+1}^j) t^{e_{i+1}} = 1 \) 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 \( a_1 t, \ldots, a_{l+1} t \) (Proposition 4.1 in [12]). □

The following definition and Proposition 4.7 concerning it are for analyzing free reduction of \( w \). We will use 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 letters. The other idea concerns where new letters appear when \( \theta^{x+1} \) is applied to some \( a_i^{x+1} \). It is evident from the definition of \( \theta \) that when \( i \geq 0 \), the lower rank letters produced by applying \( \theta \) to \( a_0 \) or \( a_i^{x+1} \) appear to the right of \( a_n \) and to the left of \( a_1^{x+1} \). The same is true when \( i < 0 \) — see Lemma 7.1 of [12].
Definition 4.6. We will define various types a subword

\[ z = \theta^x(a_i^1) \cdots \theta^x(a_i^k) \]

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 iv, and prefixes and suffixes it has in cases i–iv. When it appears below, \( u \) denotes a (possibly empty) subword \( \theta^x(a_i^1) \cdots \theta^x(a_i^k) \) such that \( i_1, \ldots, i_k \leq 2 \).

\((i)\) \( e_p = 1, e_q = -1 \)
\( i_p, l_q \geq 3, i_{p+1}, \ldots, l_{q-1} \leq 2 \)
\( e_p, e_q \geq 0 \)
\( z = \theta^x(a_i^1)u \theta^x(a_i^1) \)
\( z' = \theta^{x-1}(a_i^1) \theta^{-1}(a_i^1) \quad \text{if } e_p > 0 \)
\( = a_{i_1} \cdots a_{i_k} \quad \text{for } e_p \geq 0 \)

\((ii)\) \( e_p, \ldots, e_q = 1 \)
\( i_p \geq 3, l_q \geq 2 \)
\( i_j = i_{j+1} + 1 \) for \( j = p, \ldots, q - 1 \)
\( e_p < 0 \)
\( \text{(so } e_{p+1}, \ldots, e_q < 0 \text{ by (19))} \)
\( z = \theta^x(a_i^1) \cdots \theta^x(a_i^k) \)
\( z' = \theta^{x+1}(a_i^1) \theta^x(a_i^1) \)
\( = a_{i_1} \cdots a_{i_k} \quad \text{for } e_p \geq 0 \)

\((ii')\) \( e_p, \ldots, e_q = -1 \)
\( i_p \geq 3, i_q \geq 2 \)
\( i_j = i_{j+1} + 1 \) for \( j = p + 1, \ldots, q \)
\( e_q < 0 \)
\( \text{(so } e_p, \ldots, e_{q-1} < 0 \text{ by (19))} \)
\( z = \theta^x(a_i^1) \cdots \theta^x(a_i^k) \)
\( z' = \theta^{x+1}(a_i^1) \theta^x(a_i^1) \)
\( = a_{i_1} \cdots a_{i_k} \quad \text{for } e_p \geq 0 \)

\((iii)\) \( p < q' \leq q \)
\( e_p = 1, e_q', \ldots, e_q = -1 \)
\( i_p, l_{q'}, \ldots, l_q \geq 3, \)
\( i_{p+1}, \ldots, l_{q'-1} < 3 \)
\( i_j = i_{j+1} + 1 \) for \( j = q' + 1, \ldots, q \)
\( e_p \geq 0, e_q < 0 \)
\( \text{(so } e_p', \ldots, e_{q-1} < 0 \text{ by (19))} \)
\( z = \theta^x(a_i^1) \cdots \theta^x(a_i^p) \theta^x(a_i^q') \)
\( z' = a_{i_1} \cdots a_{i_k} \quad \text{for } e_p \geq 0 \)

\((iii')\) \( p \leq p' < q \)
\( e_p, \ldots, e_{p'} = -1, e_q = 1 \)
\( i_p, \ldots, i_{p'}, i_q \geq 3 \)
\( i_j = i_{j+1} + 1 \) for \( j = p, \ldots, p' - 1 \)
\( e_p < 0, e_q \geq 0 \)
\( \text{(so } e_{p+1}, \ldots, e_{p'} < 0 \text{ by (19))} \)
\( z = \theta^x(a_i^1) \cdots \theta^x(a_i^p) \theta^x(a_i^q') \)
\( z' = a_{i_1} \cdots a_{i_k} \quad \text{for } e_p \geq 0 \)

\((iv)\) \( p \leq p' < q' \leq q \)
\( e_p, \ldots, e_{p'} = 1, e_q', \ldots, e_q = -1 \)
\( i_p, \ldots, i_{p'}, i_q', \ldots, i_q \geq 3 \)
\( i_{p'+1}, \ldots, i_{q'-1} < 3 \)
\( i_j = i_{j+1} + 1 \) for \( j = p, \ldots, p' - 1 \)
\( i_j = i_{j+1} + 1 \) for \( j = q' + 1, \ldots, q \)
\( e_p, e_q < 0 \)
\( \text{(so } e_{p+1}, \ldots, e_{p'} < 0 \text{ and } e_{q'}, \ldots, e_{q-1} < 0 \text{ by (19))} \)
(v) For no $0 \leq p' < q' \leq l + 1$ with $p \leq q' \leq q$ and by Lemma 4.5, no $e$ is $\theta^r(a_q) \cdot \theta^s(a_q^2)$
one of the above types.

**Proposition 4.7.** In types i, ii$^a$, iii$^a$, 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

$$z = \theta^r(a_{i_p}) u \theta^s(a_{i_q}^{-1})$$

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

$$u = \theta^r(a_{i_p}) \cdot \theta^s(a_{i_q}^{-1})$$

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 $\theta^r(e_p) a_{i_q}^{-1}$ is a prefix of $z'$. We will also show that if $e_q > 0$, then $a_{i_q}^{-1} \theta^s(e_q)$ 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 elseLemma 4.5 (applied to $u$ rather than to $w$) would be contradicted. So $z = \theta^r(a_{i_p}) \theta^s(a_{i_q}^{-1})$ and by (19), $e_p = e_q$. Now $\theta^r(a_{i_p})$ contains an $a_2$ if and only if $i_p - 2 \leq e_p$, and in that event $\theta^r a_{i_p} \theta^s(a_{i_q}^{-1}) = a_2 a_1^{e_p} a_{i_q}^{-1}$ is a suffix of $\theta^r(a_{i_p})$. Similarly, $\theta^s(a_{i_q}^{-1})$ contains an $a_2^{-1}$ if and only if $i_q - 2 \leq e_q$, and in that event $\theta^s a_{i_q}^{-1} \theta^r(a_{i_p}^{-1}) = a_1^{e_q} a_{i_p}^{-1}$ is a prefix of $\theta^s(a_{i_q}^{-1})$. If $i_p - 2 > e_p$, then $i_q > e_q$, and so the final letter of $\theta^r(a_{i_p})$ is $a_{i_q}^{-1}$. Likewise, if $i_q - 2 > e_q$, then $a_{i_q}^{-1}$ is the first letter of $\theta^s(a_{i_q}^{-1})$.

1.1. **Case: cancellation occurs between some letters $a_{i_1}^{\pm 1}, \ldots, a_{i_k}^{\pm 1}$ when $z$ is freely reduced to $z'$.** If $i_p - 2 \leq e_p$, then the final $a_2$ in $\theta^r(a_{i_p})$ must cancel with the first $a_2^{-1}$ in $\theta^s(a_{i_q}^{-1})$. So $i_q - 2 \leq e_q$, and the whole suffix $a_2 a_{i_p}^{-1} a_{i_q}^{-1}$ of $\theta^r(a_{i_p})$ cancels with the whole prefix $a_1^{e_p} a_{i_q}^{-1}$ of $\theta^s(a_{i_q}^{-1})$. 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 $a_{i_q}^{-1}$ and $a_{i_q}^{-1}$ of $\theta^r(a_{i_q})$ and $\theta^s(a_{i_q}^{-1})$, respectively, must be mutual inverses, and so again we get the contradiction $i_p = i_q$.

1.2. **Case: no cancellation occurs between letters $a_{i_1}^{\pm 1}, \ldots, a_{i_k}^{\pm 1}$ when $z$ is freely reduced to $z'$.** If $i_p - 2 > e_p$ or $i_q - 2 > e_q$, then the last letter of $\theta^r(a_{i_p})$ or the first letter of $\theta^s(a_{i_q}^{-1})$, respectively, has rank greater than 2 and so is not cancelled away, and therefore $z' = z$. If $i_p - 2 \leq e_p$, $i_q - 2 \leq e_q$, then there is no cancellation between some of the $a_1^{e_p-i_p+1}$ at the end of $\theta^r(a_{i_p})$ and some of the $a_1^{e_q-i_q+2}$ at the start of $\theta^s(a_{i_q}^{-1})$ (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 $\theta^r(a_{i_p}) a_{i_q}^{-1}$ is a prefix of $z'$ as $a_{i_q}^{-1}$ has rank at least 2 and so is not cancelled away—that is, $a_{i_q}^{-1}$ stands as a barrier to any cancellation with later letters in $z'$, and so $\theta^s(a_{i_q}^{-1})$ is a prefix of $z'$ as claimed. Likewise, if $e_q > 0$, then $a_{i_q}^{-1} \theta^s(a_{i_q}^{-1})$ (and therefore $a_{i_q}^{-1}$) is a suffix of $z'$. 
2. Case: $u$ does not freely equal the empty word.
   2.1. Case: no letter $a_k^{i_1}, \ldots, a_k^{i_k}$ in $z$ is cancelled away when $z$ is freely reduced to give $z'$. The first and last letters, $a_{i_0}$ and $a_{i_0}^{-1}$, of $z$ are also the first and last letters of $z'$, because $i_p, i_q \geq 3$. Here is why the prefix $\theta^{r-1}(a_{i_p})a_{i_p-1}$ 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
   \[
   \theta^{r}(a_{i_p}) = \theta^{r}(a_3) = \theta^{r-1}(a_3)\theta^{r-1}(a_2) = \theta^{r-1}(a_3)a_2a_1^{i_p-1}.
   \]
   We must show that the $a_2$ of $\theta^{r-1}(a_3)a_2$ 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 $a_1^{(e_p - 1)(i_q + 2m)}a_{i_q}^{-1}$ (since no $a_2$ and $a_{i_q}^{-1}$ 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), $a_1^{(e_p + 2m)}a_{i_q}^{-1}$ freely equals a prefix of $u$ for some integer $m$. But then $-(e_p - 1) = -e_q + 2m$, contradicting $m$ being an integer. Conclude that $\theta^{r-1}(a_3)a_2$ (and therefore $\theta^{r-1}(a_{i_q})$) is a prefix of $z'$ as required. Likewise, if $e_q > 0$, then $a_{i_q-1}^{(e_q - 1)(i_q)}$ (and therefore $a_{i_q}^{-1}$) is a suffix of $z'$.
   2.2. Case: some letter $a_k^{i_1}, \ldots, a_k^{i_k}$ in $z$ is cancelled away when $z$ is freely reduced to give $z'$. The prefix $\theta^{r}(a_{i_p})$ of $z$ is a positive word and the suffix $\theta^{r}(a_{i_q}^{-1})$ is a negative word since $e_p, e_q \geq 0$. There is an $a_3$ in $\theta^{r}(a_{i_q})$ if and only if $e_p - i_p + 3 \geq 0$. Likewise there is an $a_{i_q}^{-1}$ in $\theta^{r}(a_{i_q}^{-1})$ 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 $\theta^{r}(a_{i_p})$ (a positive word) has rank greater than 3 and so must cancel. So $e_p - i_p + 3 < 0$ also, as otherwise $\theta^{r}(a_{i_q}^{-1})$ (a negative word) the leftmost letter in $\theta^{r}(a_{i_q}^{-1})$ with rank at least 3 would be an $a_{i_q}^{-1}$, which would block any cancelation of other letters $a_1^{i_1}, \ldots, a_k^{i_k}$ in $z$. So, in fact, the last letter of $\theta^{r}(a_{i_q})$ must cancel with the first letter of $\theta^{r}(a_{i_q}^{-1})$, 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 $\theta^{r}(a_{i_q})$ has suffix
   \[
   \theta^{r-1}(a_3) = a_2a_2a_1a_2a_1 \cdots a_2a_1^{e_q - i_q + 2}
   \]
   and $\theta^{r}(a_{i_q}^{-1})$ has prefix
   \[
   \theta^{r-1}(a_{i_q}^{-1}) = a_1^{-1}a_2^{-1}a_1^{-1}a_2^{-1}a_1^{-1}a_2^{-1}a_1^{-1}a_3^{-1}
   \]
   and the subword
   \[
   \theta^{r-1}(a_3)\theta^{r-1}(a_{i_q}^{-1})
   \]
   of $z$ freely equals the identity. Now $u$ has rank at most 2, so
   \[
   u = a_1^{f_1}a_2^{-1}a_1^{f_2}a_2^{-1} \cdots a_1^{f_{i_q-1}}a_2^{-1}a_1^{f_{i_q-1}}a_2^{-1}a_1^{f_{i_q-1}}a_2^{-1}a_1^{f_{i_q-1}}a_2^{-1}
   \]
   for some $\lambda, \mu \geq 0$, some $e, f_1, \ldots, f_{i_q} \leq 0$, and some $g_1, \ldots, g_{i_q} \geq 0$. The word $u$ has this form because $u$ is freely reduced and therefore any $a_3^{-1}$ in $u$ must cancel with an $a_2$ to the left which is not in $u$ and similarly every $a_2$ in $u$ must cancel with an $a_3^{-1}$ to the right.
which is not in \( u \). And because of cancellations that must occur,

\[
f_1 = -(e_p - i_p + 2) \quad g_1 = e_q - i_q + 2 \\
f_2 = -(e_p - i_p + 1) \quad g_2 = e_q - i_q + 1 \\
\vdots \\
f_{i} = -(e_p - i_p + 3 - \lambda) \quad g_{\mu} = e_q - i_q + 3 - \mu.
\]

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

\[
a_3a_2a_1a_2a_1^2 \cdots a_2a_1^{e_p-i_p+2} \bar{\xi}_i a_1^{-1}a_2^{-1}a_2^{-1} \cdots a_1^{-2}a_2^{-1}a_2^{-1}a_3^{-1}.
\]

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

\[
e_p - i_p + 3 - \lambda = e_q - i_q + 3 - \mu.
\]

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

\[
e_p - i_p + 2 + \mu + \xi = e_q - i_q + 2 + \lambda.
\]

Together (22) and (23) tell us that \( \xi = 0 \). But then \( \lambda = 0 \) or \( \mu = 0 \) because of the hypothesis \( a_1^{\xi} = a_2^{\xi} \) in the instance of the \( a_2^{\pm 1} \) and \( a_2 \) (which must be principal letters) in \( u \) each side of the \( \bar{a}_i \).

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

\[
u = a_1^{-(e_p-i_p+2)}a_2^{-1}a_1^{-(e_p-i_p+1)}a_2^{-1} \cdots a_1^{-(e_p-i_p+3)}a_2^{-1}.
\]

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

\[
a_3a_2a_1a_2a_1^2 \cdots a_2a_1^{e_p-i_p+3-\lambda-1} \bar{\xi}_i a_1^{-1}a_2^{-1}a_2^{-1} \cdots a_1^{-2}a_2^{-1}a_1^{-1}a_2^{-1}a_2^{-1}a_3^{-1}
\]

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

\[
e_p - i_p - \lambda = e_q - i_q.
\]

There are no \( a_2 \) among the principal letters in \( u \) (expressed as (20)), and the \( a_1^{\pm 1} \) principal letters are those that occur in (24). The final principal letter \( a_1^{\pm 1} \) 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 \( a_1^{m_1}, \ldots, a_1^{m_1} \) using integers \( m_1, \ldots, m_\lambda \in \mathbb{Z} \), as:

\[
\underbrace{a_1^{\text{sign}(m_1)}, \ldots, a_1^{\text{sign}(m_1)} , a_2^{-1} , a_1^{\text{sign}(m_2)}, \ldots, a_1^{\text{sign}(m_2)} , a_2^{-1} , \ldots, a_1^{\text{sign}(m_\lambda)}, \ldots, a_1^{\text{sign}(m_\lambda)}, a_2^{-1}}_{|m_1|} \quad \underbrace{a_1^{\text{sign}(m_1)}, \ldots, a_1^{\text{sign}(m_1)} , a_2^{-1} , a_1^{\text{sign}(m_2)}, \ldots, a_1^{\text{sign}(m_2)} , a_2^{-1} , \ldots, a_1^{\text{sign}(m_\lambda)}, \ldots, a_1^{\text{sign}(m_\lambda)}, a_2^{-1}}_{|m_\lambda|}.
\]

But (19) and the hypothesis that \( e_p = 1 \) 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

\[
u = a_1^{m_1}\theta^{e_p-m_1}(a_3)\cdots a_1^{m_1}\theta^{e_p-m_1-m_\lambda+1}(a_3^{-1}) \cdots a_1^{m_1}\theta^{e_p-m_1-m_\lambda+1}(a_3^{-1})
\]

\[
= a_1^{-(e_p+m_1)}a_2^{-1}a_1^{1}\cdots a_1^{-(e_p+m_1+2m_\lambda+1)}a_2^{-1}a_2^{-1}a_1^{-1}a_2^{-1}a_2^{-1}a_3^{-1}.
\]
Comparing the powers of $a_1$ here with those in (24), we get:

$$
\begin{align*}
-2 + i_p & = 2m_1 \\
-1 + i_p & = m_1 + 2m_2 \\
i_p & = -2 + m_1 + m_2 + 2m_3 \\
& \vdots \\
l - 3 + i_p & = 1 - l + m_1 + m_2 + \cdots + m_{l-1} + 2m_l,
\end{align*}
$$

which simplifies to

$$i_p + 2^{l+1} - 6 = 2^j m_j \quad \text{for } j = 1, \ldots, l.$$  

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

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

$$i_p \geq 4.$$  

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

$$e_p - i_p - 1 = e_q - i_q.$$  

Also

$$z = \theta^s(a_{i_p})\theta^{s+1}(a_1^{\text{sign}(m_1)})\cdots\theta^{s+j-1}(a_1^{\text{sign}(m_1)})\theta^{r-m_1}(a_2^{-1})\theta^{r+1}(a_3^{-1})_{\text{[m]}}$$

by (27), and so (19) applied to $\theta^{r-m_1}(a_2^{-1})$ and $\theta^{s+1}(a_3^{-1})_{\text{[m]}}$ 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

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

By (32) and (33),

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

and so

$$i_p + 6 = 2i_q.$$  

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 $e_q - i_q + 3 > 0$ or $e_q - i_q + 3 = 0$. If $e_q - i_q + 3 > 0$, there are at least two $a_1$ in $\theta^{r}(a_{i_q})$ (because $i_p \geq 4$) and hence at least two $a_1^{-1}$ in $\theta^{r+1}(a_3^{-1})_{\text{[m]}}$. Likewise, if $e_q - i_q + 3 = 0$, then there are at least two $a_1^{-1}$ in $\theta^{r}(a_{i_q})$ (because $i_q \geq 4$), and so two $a_3$ in $\theta^{r}(a_{i_q})$. In either case, using Lemma 4.4 to identify the relevant suffix of $\theta^{r}(a_{i_q})$ and prefix of $\theta^{r+1}(a_3^{-1})$, there is a subword

$$\theta^{r-i_q+2}(a_1)\theta^{r-i_q+3}(a_3)\theta^{r-i_q+3}(a_3^{-1})\theta^{r-i_q+3}(a_3^{-1}),$$

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 $\theta^{r-i_q+2}(a_3)\theta^{r-i_q+3}(a_3^{-1})$, 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 $\theta_{r-1}(a_j)\mu_{r-1}$ or first letter $a_p$ or suffix $a_{r-1}^{-1}(\theta_{r-1}(a_j)^{-1})$ or final letter $a_{r-1}^{2}$, 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), $u = a_{r-1}^{-1}(\theta_{r-1}(a_j)^{-1})$.

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 $i_p \not= 4$ 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 $\theta_{r-1}(a_j)$, specifically its final letter. So the subword $a_{r-1}(\theta_{r-1}(a_j))$ must freely equal the empty word. But $u = a_{r-1}(\theta_{r-1}(a_j))$, because the $a_j$ freely reduces to the empty word, and therefore the first and last letters ($a_r$ and $a_{r-1}$, respectively) of $z$ are also those of $z'$, as required. And, as $i_p \geq 6$, if $e_p > 0$, then the prefix $\theta_{r-1}(a_j)$ of $z$ survives into $z'$ as it ends with a letter of rank at least 5 which is not cancelled away.

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

The exponent sum of the $a_2$ in $z$ between the rightmost $a_3$ of $\theta_{r-1}(a_j)$ and the leftmost $a_3^{-1}$ of $\theta_{r-1}(a_j)^{-1}$ is zero, so

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

Applying (19) to the suffix $\theta_{r-1}^{-m_1 - \cdots - m_{\lambda-1}}(a_j^{-1})$ of $u$ (expressed as per (27)) and $\theta_{r-1}(a_j)^{-1}$, 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 $m_\lambda$ 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.

\[\square\]

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 $ii-1$.

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

$$z = \theta^s(a_{i_p}^{-1})\theta^{s+1}(a_{i_{p+1}}^{-1})\cdots\theta^s(a_{i_q}^{-1}),$$

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

$$z' = \theta^s(a_{i_q-1})\theta^{s+1}(a_{i_q}^{-1}).$$

Well,

$$\theta^{s+1}(a_{i_q}^{-1}) = \theta^s(a_{i_q}^{-1})\theta^s(a_{i_q}^{-1}) = \theta^{s-1}(a_{i_{q-1}})\theta^{s-1}(a_{i_{q-1}})\theta^s(a_{i_q}^{-1}) = \cdots = \theta^s(a_{i_{p-1}})\theta^{s+1}(a_{i_{p+1}}^{-1})\cdots\theta^s(a_{i_q}^{-1}),$$

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 $\theta^s(a_{i_{q-1}})$ is $a_{i_q}^{-1}$. When $i_q > 0$, it tells us that the first letter of $\theta^{s+1}(a_{i_q}^{-1})$ is $a_{i_{q-1}}$. Our hypotheses include that $e_q < 0$, which implies that $e_p < 0$ as $e_q < e_p$, 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 $e_q \neq -1$,

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

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

$$z' = \theta^s(a_{i_{q-1}})a_{i_q}^{-1},$$

which is freely reduced because the last letter of $\theta^s(a_{i_{q-1}})$ is $a_{i_q} - 2$. 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 $a_{i_{q-1}}$ by Lemma 4.4 applied to $\theta^s(a_{i_{q-1}})$. The final letter of $z$ is $a_{i_q}^{-1}$ because the first letter of $\theta^{s+1}(a_{i_q}^{-1})$ is $a_{i_q}^{-1}$ by the same lemma.

Proof of Proposition 4.7 in type iii. We have that

$$z = \theta^s(a_{i_p})\theta^{s'}(a_{i_{p'}}^{-1})\cdots\theta^s(a_{i_q}^{-1})$$

where $i_p, i_{p'}, \ldots, i_q \geq 3, i_{p+1}, \ldots, i_q - 1 < 3, e_p \geq 0, e_q < 0$ (and so $e_{p'}, \ldots, e_{q-1} < 0$ 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 $\theta^{s-1}(a_{i_p})$ is a prefix of $z'$.

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

$$(36) \quad \theta^s(a_{i_p}) a \theta^{s'}(a_{i_{q-1}})\theta^{s+1}(a_{i_{q+1}}^{-1})$$

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

By hypothesis, $i_{q'} \geq 3$. We again organize our proof by cases.
1. Case: \( i_q \geq 4 \). As the suffix \( \theta^r(a_{i_q-1})\theta^r(e_{q-q'-1})(a_{i_q-q'}) \) 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 \( \theta^r(a_{i_p}) \). Therefore letters in \( u \) can only cancel with the \( \theta^r(a_{i_p}) \) to its left. So the final letter of \( z' \) is \( a_{i_q-q'-1}^{*-1} = a_{i_q}^{*-1} \), 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 \( e_p > 0 \) and \( \theta^r(a_{i_p}) \) of \( z' \) is also a prefix of \( z' \). If \( i_p > 3 \), this is immediate because \( a_{i_q-1} \) has rank at least 3 and so cannot cancel into \( u \). If \( i_p = 3 \), then no \( a_{i_q}^{*-1} \) in \( u \) cancel with \( \theta^r(a_{i_p}) \) 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 \( \theta^r(a_{i_q-1})a_{i_q-1} \) (which is one letter longer than we need) of \( \theta^r(a_{i_p}) \) 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

   \[
   \theta^r(a_{i_q}) u \theta^r(a_2) a_{i_q}^{-1}.
   \]

   Then \( \theta^r(a_{i_q}) \) can contain no \( a_3 \) since there is no \( a^*_3 \) to cancel with. Therefore, \( \theta^r(a_{i_q}) \) ends with a letter of rank greater than 3 by Lemma 4.4. For this reason, \( u \) cannot cancel to its left, and so \( u \theta^r(a_2) \) freely equals the empty word. The word \( u \) cannot contain a rank 2 subword that freely equals the empty word because such a subword either has the form of the word \( w \) described in the statement of Lemma 4.5, or has the form \( a_{i_q}^{-1} \mu a_0 \) where \( a_0 \) has that same form, so \( u = a_{i_q}^* \theta^r(a_{i_q}) \) for some \( \mu \in \mathbb{Z} \). But then by (19) \( e_{q-1} = e_q - 1 \), and \( u = a_0^* \theta^r(a_{i_q}) \). Counting the exponent sum of the \( a_{i_q}^{-1} \) in \( u \theta^r(a_2) \), we find

   \[
   \mu - e_q + 1 + e_{q'} = 0.
   \]

   So \( \mu = -1 \), and \( u \) must be \( \theta^r(a_{i_q}) \theta^r(a_{i_q}) \). But then applying (19) to \( \theta^r(a_{i_q}) \theta^r(a_{i_q}) \theta^r(a_{i_q}) \), 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

   \[
   \theta^r(a_{i_q}) u \theta^r(a_2) \theta^r(e_{q-q'-1})(a_{i_q}^{-1}).
   \]

   The first letter \( a_{i_q-1} \) of the suffix \( \theta^r(e_{q-q'-1})(a_{i_q}) \) has rank at least 3 (by Lemma 4.4), 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 \( a_{i_q}^{-1} \) and start with \( a_{i_q} \).

   If \( i_p > 3 \) and \( e_p > 0 \), the letter immediately after the prefix \( \theta^r(a_{i_p}) \) of \( z \) is \( a_{i_q-1} \), which is of rank at least 3, so the prefix \( \theta^r(a_{i_p}) \) 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 \( \theta^r(e_{q-q'-1})(a_{i_q}) \). This establishes the required contradiction in this case.

   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 \( \theta^r(a_{i_p})u \) of
rank 2 or higher cancel, then $e_p - i_p + 2 \geq 0$ because otherwise $\Theta^s(a_i)$ ends with a letter of rank greater than 3. However, then $u$ must have a prefix that cancels with $\Theta^{s+y+12}(a_2)$ and so is $\Theta^{s+y+12}(a_1) \cdots \Theta^{s+y+12}(a_2)$ or $\Theta^{s+y+12}(a_1) \cdots \Theta^{s+y+12}(a_2)$ for some $s$. In either case, this simplifies to $\Theta^s(a_i)$ for some $\mu \in \mathbb{Z}$ and, by (19), $e_p - \mu = e_i$. By summing the exponents of the $a_i$ in $\Theta^{s+y+12}(a_2)$ and in $\Theta^s(a_i)$, we find that: $e_p - i_p + 2 - e_s + \mu = 0$. But combined with $e_p - \mu = e_i$, 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

$z = \Theta^s(a_i) u \Theta^s(a_i^-)$.

By Lemma 4.4, there is one $a_i$ in $\Theta^s(a_i^-)$, specifically its final letter. Suppose this $a_i$ cancels with an $a_i$ (necessarily the rightmost) in $\Theta^s(a_i)$. Then the intervening subword (which has rank at most 2) freely reduces to the empty word—violating Lemma 4.5 as in Case 2.2.1.

Now $\Theta^s(a_i)$ contains no $a_i$ because $e_p \geq 0$. The same is true of $\Theta^s(a_i^-)$ by Lemma 4.4 and the fact that $e_p < 0$. So, if $u$ contains an $a_i$, it must cancel with an $a_i$ 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 and includes no intervening $a_i$—conclude that $u$ contains no $a_i$.

2.3.1. Case: $e_p - i_p + 2 \geq 0$. The rightmost $a_i$ in $\Theta^s(a_i)$ is the first letter of the suffix $a_i a_2^0(\Theta(a_2) \cdots \Theta^s(a_2))$, so some prefix of $u$ freely equals the inverse of $a_2^0(\Theta(a_2) \cdots \Theta^s(a_2))$. This prefix of $u$ must be

$$a_i^{\nu_1 \cdots \nu_s \cdots a_i}$$

for some $s$. (The prefix does not end in the midst of some $\Theta^s(a_i)$, because it must have final letter $a_i$.)

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

$$\begin{align*}
& a_i^{\nu_1} \Theta^{s+y+12}(a_2) \cdots a_i^{\nu_s} \Theta^{s+y+v_s} \cdots a_i^{\nu_1} \Theta^{s+y+12}(a_2) \\
& = a_i^{\nu_1} \cdots a_i^{\nu_s} \cdots a_i^{\nu_1} \cdots a_i^{\nu_s} \cdots a_i^{-\chi} \cdots a_i^{\nu_1} \cdots a_i^{\nu_s} \cdots a_i \\
& 
\end{align*}$$

for some $s$ where $\chi := e_p - i_p + 2$ (so that $\chi + 1$ is the number of $a_2$ in $a_2^0(\Theta(a_2) \cdots \Theta^s(a_2))$ and $\nu_1, \ldots, \nu_s \in \mathbb{Z}$ record the number of and exponents of the $a_i$ between the $a_i$). As this freely equals

$$(a_2^0(\Theta(a_2) \cdots \Theta(a_2)))^{-1} = a_i^{-\chi} \cdots a_i^{-1} a_i a_i^{-1} a_i^{-1} a_i^{-1},$$

we find that

$$v_1 - e_s = 0$$
$$v_2 - (e_s + v_1 - 1) = -1$$
$$\vdots$$
$$v_{\chi + 1} - (e_s + v_1 + \cdots + v_\chi - \chi) = -\chi.$$

It follows that

$$v_{\chi + 1} = 2^\chi e_s - 2^\chi + 2.$$

The suffix $\Theta^{s+y+12}(a_2)$ of $\Theta^s(a_i)$ must be the inverse of the prefix $a_i^{\nu_1} \cdots a_i^{\nu_s} \cdots a_i^{\nu_1} \cdots a_i^{\nu_s} \cdots a_i^{-\chi} \cdots a_i^{-1} a_i^{-1} a_i^{-1}$, so $\Theta^{s+y+12}(a_2) a_i^{\nu_1} \cdots a_i^{\nu_s} \cdots a_i^{-\chi} \cdots a_i^{-1} a_i^{-1} a_i^{-1}$ freely reduces to the empty word. By (19) applied to $\Theta^s(a_i) a_i^{\nu_1} \cdots a_i^{\nu_s} \cdots a_i^{-\chi} \cdots a_i^{-1} a_i^{-1} a_i^{-1}$, $e_p - v_{\chi + 1} = e_s + v_1 + \cdots + v_\chi - \chi$. 

\[ e_p - i_p + 2 \geq 0 \]
By counting the $a^{i_1}_1$ in $\theta^{e_p-i_p+2}(a_2)\theta^{e_p-i_p+3}(a^{-1}_2)$, which freely reduces to the empty word, we find
\[
e_p - i_p + 2 + \nu_{k+1} = e_p - \nu_{k+1},
\]
so that $\nu_{k+1} = (i_p - 2)/2$. But then $\nu_{k+1} > 0$, since $i_p \geq 3$. Further, we conclude that for $u$ to even cancel an $a_2$ from $\theta^u(a_{i_p})$, $i_p$ must be even. So $i_p \geq 4$. Thus after rewriting (38) as
\[
es = \frac{1}{2^\chi}(\nu_{k+1} + 2^{k+1} - 2)
\]
and using the fact that $\nu_{k+1} > 0$ and $\chi \geq 1$, we conclude that $e_s > 0$.

The remainder
\[
\theta^{e_q}(a^{s'}_1) \cdots \theta^{e_q-1}(a^{s'-1}_1),
\]
(where $s' = s + 1$) of $u$ cancels with all but the $a^{-1}_3$ of $u$
\[
\theta^{e_q}(a^{-1}_1) = \theta^{e_q}(a_2)\theta^{e_q+1}(a_2) \cdots \theta^{e_q+1}(a_2)a^{-1}_1.
\]
We claim that, similarly to (27), we can rewrite (40) as
\[
da^{d}_1 \theta^{e_q+\eta_1+n_1+n_2+\cdots+n_{k-1}}(a^{i_1}_1) \cdots da^{d}_1 \theta^{e_q+\eta_1+2}(a^{i_2}_2) \cdots da^{d}_1 \theta^{e_q-1}(a^{-1}_2) = da^{d}_1 a^{-1}_1 a^{-1}_2 a^{d}_1.
\]
where $r$ is the number of $a^{i_1}_1$ in (40), and $\eta_1, \ldots, \eta_r \in \mathbb{Z}$ record the number of and the signs of the intervening terms $\theta^u(a^{a'}_{i'})$. There is no power of $a_1$ at the righthand end because the first letter of (41) is $a_2$. The iterates of $\theta$ are identified by using (19).

Now compare with (41), with which it cancels (to leave only $a^{-1}_3$), to see that $r = |e_q|$ and
\[
0 = \eta_1 - (e_q - 1) + e_q
\]
\[
0 = \eta_2 - (e_q + \eta_1 - 2) + e_q + 1
\]
\[
\vdots
\]
\[
0 = \eta_r - (e_q + \eta_1 + \eta_2 + \cdots + \eta_{r-1} - r) + e_q + (r - 1).
\]
Next we establish by induction that $\eta < 0$ and
\[
e_q + \eta_1 + \eta_2 + \cdots + \eta_{i-1} - i < 0
\]
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
\[
e_q = \eta_i + (e_q + \eta_1 + \eta_2 + \cdots + \eta_{i-1} - r) - 1
\]
by (19). Conclude that $e_q < 0$. 

But
\[ u = \theta^{e_1} \left( a_{i_{p+1}}^{e_{i_{p+1}}} \right) \cdots \theta^{e_j} \left( a_{i_{j}}^{e_{i_{j}}} \right) \theta^{e_{i_{j}+1}} \left( a_{i_{j+1}}^{e_{i_{j+1}}} \right) \cdots \theta^{e_{i_q-1}} \left( a_{i_{q-1}}^{e_{i_{q-1}}} \right) \]
and by (19), \( e_s \) and \( e_c \) 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 \( \theta^{e_{i_p-1}}(a_{i_p}) \), since cancelling any part of this prefix in \( \theta^{e_{i_p-1}}(a_{i_p}) = \theta^{e_{i_p-1}}(a_{i_p})\theta^{e_{i_p-1}}(a_{i_p-1}) \) 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 \( \theta^{e_{i_p}}(a_{i_p}) \). 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 \( z = \theta^q(a_{i_p})u\theta^{e_{i_p}}(a_{i_p}^{-1}) \) where \( i_q = 3 \), \( q = q' \), \( u = \theta^{e_{i_p+1}}(a_{i_{p+1}}^{e_{i_{p+1}}}) \cdots \theta^{e_{i_q-1}}(a_{i_{q-1}}^{e_{i_{q-1}}}) \), and \( \theta^{e_{i_q}}(a_{i_q}) \) ends with a letter of rank at least 3. Suppose, for a contradiction, some letter of the prefix \( \theta^{e_{i_q}}(a_{i_q}) \) is cancelled when \( z \) is freely reduced to \( z' \). No cancellation is possible between \( \theta^{e_{i_q}}(a_{i_q}) \) and \( u \) because every letter of \( \theta^{e_{i_q}}(a_{i_q}) \) is rank 3 or higher. By the argument used in Case 2.3.1 to show that \( e_{i_q} < 0 \), we find here that \( e_{i_{p+1}} < 0 \), and by the argument there (immediately after (42)) to show that \( e_{i_q} < 0 \), we find here that \( e_{i_{p+1}} = -1 \). But then by (19), \( e_p = e_{i_{p+1}} \), 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 \( a_{i_q}^{-1} \) of \( \theta^{e_{i_q}}(a_{i_q}^{-1}) \) is also the last letter of \( z' \). Moreover, if \( e_p > 0 \), then the prefix \( \theta^{e_{i_{p+1}}}(a_{i_p}) \) of \( \theta^{e_{i_q}}(a_{i_q}) \) is also a prefix of \( z' \). \( \square \)

**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). \( \square \)

**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 \( j = q', \ldots, q \), and \( e_p, e_q < 0 \), the freely reduced form \( z' \) of
\[ z = \theta^{e_{i_p}}(a_{i_p}) \cdots \theta^{e_{i_{p+1}}} \left( a_{i_{p+1}}^{e_{i_{p+1}}} \right) \cdots \theta^{e_{i_q}} \left( a_{i_q}^{e_{i_q}} \right) \]
starts with \( a_{i_p} \) and ends with \( a_{i_q}^{-1} \).

By Proposition 4.7 in type ii^{-1}, proved above, \( z \) freely reduces to
\begin{equation}
\theta^{e_{i_p+1}}(a_{i_p})\theta^{e_{i_{p+1}}}(a_{i_{p+1}}^{-1})u\theta^{e_{i_q}}(a_{i_q}^{-1})\theta^{e_{i_{q-1}}}(a_{i_{q-1}})
\end{equation}
where \( \theta^{e_{i_p+1}}(a_{i_p}) \theta^{e_{i_{p+1}}}(a_{i_{p+1}}^{-1}) \) and \( \theta^{e_{i_q}}(a_{i_q}^{-1}) \theta^{e_{i_{q-1}}}(a_{i_{q-1}}) \) 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 \( \theta^{-1} \) 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 \( \theta^{e_{i_p+1}}(a_{i_p}) \theta^{e_{i_{p+1}}}(a_{i_{p+1}}^{-1}) \) and the first letter \( a_{i_{q-1}} \) of
\( \theta^\varphi (a_{i_p}^-) = \theta^\varphi (a_{i_q}^-) \) cancel—that is, \( i_p' = i_q' \), so \( \theta^\varphi (a_{i_p}) \theta^\varphi (a_{i_q}^-) \) 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 \( \epsilon_p, \epsilon_q < -1 \), the word in (43) has a subword of the form

\[
\theta^\varphi (a_{i_p}^-) a \theta^\varphi (a_{i_q}^-) a,
\]

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

\[
\theta^\varphi (a_{i_p}^-) a \theta^\varphi (a_{i_q}^-) a
\]

must freely equal the empty word.

1.2.1.1. Case: \( i_p = i_{p'} \neq i_q \neq i_{q'} \). Then \( i_q = i_{q'} \neq i_p \neq i_{p'} \). However, then \( a_{i_p}^- \) and \( a_{i_q}^- \) 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: \( i_{p'} = i_{q'} = 2 \). 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

\[
\theta^\varphi (a_{i_p}^-) a \theta^\varphi (a_{i_q}^-) a,
\]

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

\[
epsilon_p' - (\epsilon_p' - 1) + \mu + (\epsilon_q' - 1) - \epsilon_q' = 0,
\]

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

\[
\theta^\varphi (a_{i_p}) a \theta^\varphi (a_{i_q}^-) a \theta^\varphi (a_{i_q}^-).
\]

Recall that the suffix \( \theta^\varphi (a_{i_q}^-) \theta^\varphi (a_{i_q}^-) a \) is freely reduced and so its first letter \( a_{i_q}^- \) cannot cancel to its right. So it must cancel to its left, and therefore either \( i_q' = 3 \) or it cancels with the terminal \( a_{i_p}^- \) of \( \theta^\varphi (a_{i_p}) \). 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

\[
a_{i_p}^- a \theta^\varphi (a_2) a_{i_q}^- a
\]

whose first letter \( a_{i_p}^- \) cannot cancel to the left and whose last letter \( a_{i_q}^- \) cannot cancel to the right. They have rank at least 3, so they must cancel each other. So \( a \theta^\varphi (a_2) \) 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
$$d_i^t \theta^\nu(a_{i-1}^{-1})$$
for some $\mu \in \mathbb{Z}$. So (46) is
$$a_{i-1}^{-1} d_i^t \theta^\nu(a_{i-1}) \theta^\nu(a_2) a_i^{-1} = a_{i-1}^{-1} a_i^e (a_2 a_{i-1}^{-1})^{-1} a_2 a_i^{-1} a_i^{-1}.$$  
By counting the exponent sum of $a_1$ it contains, we find 
$$\mu - (e_q - 1) + e_{\varphi} = 0.$$  
So $\mu = -1$. Now
$$u = a_{i-1}^{-1} \theta^\nu(a_{i-1})^{-1}$$
for some $e \in \mathbb{Z}$. So $\theta^\nu(a_j) \theta^\nu(a_{i-1})$ 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 $u \theta^\nu(a_2)$ freely equals the empty word and $p = p'$, (43) freely reduces to
$$\theta^\nu(a_j) \theta^\nu(a_{i-1}) \theta^\nu(a_1^{-1}) = \theta^\nu(a_j) \theta^\nu(a_{i-1}^{-1}).$$
So, as $i_p = i_q > 1$, we find $e_p = e_q + 1$. But $e_q \geq e_{\varphi}$, so this contradicts $e_p + 2 = e_q$.

1.2.2. Case: $q = q'$. In this instance,
$$z = \theta^\nu(a_j) u \theta^\nu(a_u)$$
freely reduces to the identity. Hence $\theta^\nu(e_{\varphi}^{-1} e_{\nu}^{-1} z)$ 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 $ii^{k+1}$, $w$ freely reduces to a word of the form:
$$\theta^\nu(a_j) \theta^\nu(a_{i-1}^{-1}) u \theta^\nu(a_{i-1}^{-1}) \theta^\nu(a_{i-1}^{-1}).$$

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 1 letter appearing to the right of $u$. Also, $a_{i_q}^{-1} 1$, the final letter of $\theta^\nu(a_j)$ is an obstruction to cancelling $a_{i_q}$ with any letter from $\theta^\nu(a_{i-1})$ and $a_{i_q}^{-1}$ and has rank at least $i_q$. Thus the only letters of rank $i_p - 1$ in $w$ come from $\theta^\nu(a_j)$, so every letter of rank $i_p - 1$ has exponent $-1$. To cancel $a_{i_q}^{-1}$ with a letter from $\theta^\nu(a_j)$ requires cancelling the rightmost $a_{i_q}^{-1}$ from $\theta^\nu(a_j)$ which is impossible.

Similarly, if $a_{i_q}^{-1}$ cancels with a letter from $\theta^\nu(a_{i_q}^{-1})$, the rightmost letter of $\theta^\nu(a_{i_q}^{-1})$, which is $a_{i_q}^{-1}$, must cancel too. By Proposition 4.7 in type $ii^{k+1}$, $\theta^\nu(a_j) \theta^\nu(a_{i_q}^{-1})$ is freely reduced, so its rightmost $a_{i_q}^{-1}$ must cancel to the right. However, $a_{i_q}^{-1}$ is the highest rank letter in $\theta^\nu(a_{i-1}^{-1})^{-1}$, so $e_{\nu} - 1 \geq i_q$. Also $i_p - 1 \leq i_q$ because $a_{i_p}^{-1}$ can only cancel with an $a_{i_q}^{-1}$. We cannot cancel $a_{i_q}^{-1}$ from $\theta^\nu(a_{i_q}^{-1})$ 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.

\[ \square \]

Proof of Proposition 4.7 in type $v$. We have
$$z = \theta^\nu(a_{i_1}^e) \cdots \theta^\nu(a_{i_k}^e)$$
and no type i–iv subword $z$ overlaps with $w$. More precisely, there is no $0 \leq p' < q' \leq l + 1$ with $p \leq q' \leq q$ such that $\theta^{\epsilon'}(a_{q'}) \cdots \theta^{\epsilon'}(a_{p'})$ 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 $\epsilon_p = 1$, $i_p \geq 3$ and $\epsilon_p > 0$, then $z'$ (the reduced form of $z$) has prefix $\theta^{\epsilon-1}(a_{i_p})$.

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

Then $z$ has a subword $a_{q'}^\alpha v a_{q''}^{-\epsilon}$ which freely equals the empty word. Since $\alpha \geq 3$, we know that $a_{q'}$ comes from some $\theta^{\epsilon'}(a_{q'}^{\prime})$ where $i_{q'} \geq 3$ while $a_{q''}^{-1}$ comes from some $\theta^{\epsilon'}(a_{q''}^{\prime})$ where $i_{q''} \geq 3$. Note that $p' \neq q'$ because otherwise $a_{q'}^\alpha v a_{q''}^{-\epsilon}$ would be a subword of $\theta^{\epsilon'}(a_{i_q'})$, which is freely reduced. We may assume that $v$ contains no letter $a_0^{\beta}$ with $\beta \geq 3$ and $\delta \in \{\pm 1\}$ that cancels to its right with an $a_0^{\beta}$ in $v$, because otherwise we could replace our original choice of $a_{q'}^\alpha v a_{q''}^{-\epsilon}$ with a shorter subword $a_0^{\beta} \cdots a_0^{\beta}$. So rank($v$) $\leq 2$, and $z$ has a subword $\theta^{\epsilon'}(a_{q'}^{\prime}) u \theta^{\epsilon'}(a_{i_q'}^{\prime})$

where $u$ is either empty or rank($u$) $\leq 2$.

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 $a_{q'}^{\alpha} a_{q''}^{-\epsilon}$ to cancel, the $a_{i_{q''}}$ at the start of $\theta^{\epsilon'}(a_{i_{q''}}^{\prime})$ must cancel to its left. If $\epsilon_{q'} \geq 0$, then $\theta^{\epsilon'}(a_{i_{q'}}^{\prime})$ is a positive word, so the only letters to the left of $a_{i_{q'}}$ with exponent $-1$ have lower rank (in fact, rank at most 2), and such cancellation is not possible. If $\epsilon_{q'} < 0$, then the last letter of $\theta^{\epsilon'}(a_{i_{q'}})$ is $a_{i_{q'}-1}^{-1}$, so either $i_{q'} - 1 = 2$ or ($u$ is the empty word and $i_q = i_{q'-1}$). In the former case: $\alpha = 3$, but then $a_{q'}^{\alpha} v a_{q''}^{-\epsilon}$ cannot freely equal the empty word because $a_{q'}^{\alpha} = a_q$ cannot cancel with the first letter $a_{i_q}$ of $\theta^{\epsilon'}(a_{i_q})$. In the latter case: by (19), $\epsilon_q = \epsilon_{q'} - 1 < 0$, 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 $a_{i_{q'}}^{-1} u a_{i_q}$

where $a_{i_{q'}}^{-1}$ 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 $\epsilon_p = 1$, $i_p \geq 3$ and $\epsilon_p > 0$, then $z'$ has prefix $\theta^{\epsilon-1}(a_{i_p})$—is proved exactly as per the final paragraph of Case 2.2.2 of our proof above Proposition 4.7 in case iii.\[\square\]

4.4. The Piece Criterion. The Piece Criterion is the main technical result behind the correctness of our algorithm Member$. 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 freely reduced word $h$ on $(a_1 t)^{\pm 1}$, $(a_2 t)^{\pm 1}$, which is to say that $h$ contains no subwords $(a_1 t)^{\pm 1}(a_2 t)^{\pm 1}$.

**Proposition 4.8.** Suppose $u = u(a_1, \ldots, a_{m-1})$ is freely reduced and non-empty, $h = h(a_1, \ldots, a_t)$ is freely reduced, $r, s \in \mathbb{Z}$, and $2 \leq m \leq k$. In $G_k$,
We will first argue that $hr$ from left to right through the word, with the effect of applying $\theta^1$ to the intervening letters $a_m^1$, and then freely reducing, so as to arrive at $amu$.

We will first prove the case $f amu = ht^s$, and so $amu = t^{-1}ht^s$, in $G_k$. Consider carrying all the $t^j$ in $t^{-1}ht^s$ from left to right through the word, with the effect of applying $\theta^1$ to the intervening letters $a_m^1$, and then freely reducing, so as to arrive at $amu$.

We will first argue that $h$ contains no $(a_{m-1})^{s_1}, \ldots, (a_1)^{s_1}$. Suppose otherwise. Let $i$ be maximal such that $h$ contains an $(a_i)^{s_1}$. As carrying all the $t^j$ to the right and cancelling gives $amu$, there must be an $(a_i)^{s_1}$ in $h$ so that there is an $a_i^{s_1}$ to cancel with the $a_i^{s_1}$ in our $(a_i)^{s_1}$—this is because applying $\theta^1$ to $a_i^{s_1}, \ldots, a_i^{s_1}$, neither creates nor destroys any $a_i^{s_1}$. But then if $h^r$ is such a subword of $h$ that has first and last (or last and first) letters these $(a_i)^{s_1}$ and $(a_i)^{s_1}$. In the case where $h^r = (a_i)^{s_1}h''(a_i)^{s_1}$, we have that $h^r = t^s$ for some $r', s' \in \mathbb{Z}$. In the case where $h^r = (a_i)^{s_1}h''(a_i)$, we have that $h^r = t^s$ for some $r'', s'' \in \mathbb{Z}$ (since for the $a_i^{s_1}$ and $a_i$ to cancel, all the intervening $a_i, \ldots, a_{i-1}$ must cancel).

These cases imply that $h^r \in (i)$ and $h^s \in (i)$, respectively. But $H_k \cap (i) = \{1\}$ by Lemma 6.1 of [12], so in both cases we find $h = 1$ in $G_k$. But $H_k = F(a_{i1}, \ldots, a_{it})$ by Proposition 4.1 of [12], and so our assumption that $h$ is freely reduced is contradicted.

Next notice that there must be an $(a_{m-1})$ in $h$ because $amu$ contains an $a_m$ and applying $\theta^1$ to $a_m^1, \ldots, a_m^1$ neither creates nor destroys any $a_m^1$. Suppose, for a contradiction, that the first $(a_{m-1})$ in $h$ is not at the front. Express $h$ as $\alpha(a_{m-1})\beta$ where $\alpha = (a_{i1}, \ldots, a_{i1-1})$ is non-empty.

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

$$t^{-1}ht^s = t^{-1}am\beta = vt^s(a_{m})\beta$$

for some $v = v(a_{i1}, \ldots, a_{i1-1})$ 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 $t^j$ are carried to the right: applying $\theta^1$ to $a_m$ only produces new letters $a_m^{s_1}, \ldots, a_m^{s_1}$ (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 $a \in (i)$, leading to a contradiction as before.

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

We now turn to the case where $f amu a_m^{-1} = ht^s$. The argument that $h$ contains no letters of rank greater than $m$ is the same. We see that $amu a_m^{-1} = t^{-1}ht^s$. The left side is freely reduced and therefore $amu$ is the first letter of its freely reduced form. This is the vital fact for then repeating the above argument.

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 $u = u(a_{i1}, \ldots, a_{i1-1})$ is freely reduced, $h = h(a_{i1}, \ldots, a_{i1})$ is freely reduced, $r, s \in \mathbb{Z}$, $3 \leq m \leq k$, and $f amu = ht^s$ or $f amu a_m^{-1} = ht^s$ in $G_k$. If $r > 0$, then $\theta^{-1}(a_m^1)$ is a prefix of $amu$. 
Proof. We will prove the case where \( t' a_m u a_m^{-1} = h t' \) in \( G_k \). The proof for the case \( t' a_m u = h t' \) is similar.

Proposition 4.8 tells us that the first and last letters of \( h \) are \((a_m t)^{-1} \) and \((a_m t)^{-1} \), respectively. Express \( h \) as \((a_i t)^{e_0} \cdots (a_j t)^{e_j} \) where \( e_0 = 1 \), and \( e_1, \ldots, e_j = \pm 1 \), and \( e_{j+1} = -1 \), and \( i_0 = i, j+1 = m, \) and \( i_1, \ldots, i_j \in \{1, \ldots, m-1\} \).

If we shuffle all the \( t^{\pm 1} \) in \( t' h t' \) to the right, then the power of \( t \) emerging on the right cancels away since \( t' h t' \) equals \( a_m u a_m^{-1} \) and \( u = u(a_1, \ldots, a_{m-1}) \) in \( G_k \), and we get

\[
\pi := a_m u a_m^{-1} \quad z := \theta_0^{e_0}(a_0^{e_0}) \cdots \theta_j^{e_j}(a_j^{e_j}) \theta_{j+1}^{e_{j+1}}(a_{j+1}^{e_{j+1}})
\]

where \( \pi \in G_k \) and \( e_1 \) is, for \( 0 \leq l \leq j+1 \), the exponent sum of the \( t^{\pm 1} \) in \( h \) that precede \( a_0 \) in \( t' h t' a_m \) (which includes the \( t^{-1} \) of \( (a_i t)^{e_l} \) if \( e_l = -1 \)):

\[
e_l = \begin{cases} r + e_1 + \ldots + e_{l-1} & \text{if } e_l = 1 \\ r + 1 + e_1 + \ldots + e_{l-1} & \text{if } e_l = -1. \end{cases}
\]

Also \( a_i^{e_l} \neq a_{i+1}^{-e_{l+1}} \) for \( x = 0, \ldots, j \) because \( h \) is freely reduced as a word on \((a_1 t)^{e_1}, \ldots, (a_i t)^{e_i}\). So, \( z \) is of the form in which it appears in Definition 4.6.

We will work right to left through \( z \) choosing subwords \( z_1, z_2, \ldots \) until we have \( z \) expressed as a concatenation \( z_1 \cdots z_{l-1} z_l \). Define \( \pi_1 := z \) and define \( z_l \) to be the maximal length suffix of \( \pi_1 \) of one of the five types of Definition 4.6. (Such a suffix exists if \( \pi_1 \) is non-empty, as there must be a type \( v \) suffix if no other type.) Let \( \pi_2 \) be \( \pi_1 \) with the suffix \( z_1 \) removed, and then define \( z_2 \) to be the maximal length suffix of \( \pi_2 \) of one of the five types of Definition 4.6. Continue likewise until \( z \) is exhausted and we have \( z = z_1 \cdots z_{l-1} z_l \). Note that \( z = \pi_1 z_1 \cdots z_{l-1} z_l \) in \( G_k \).

Let \( z', z'_1, \ldots, z'_l \) denote the freely reduced forms of \( z, z_1, \ldots, z_{l-1} \), respectively. We will use Proposition 4.7 to argue that \( z' = z'_1 \cdots z'_l \). In other words, when freely reducing \( z \), all cancellation is within the \( z_i \)—none occurs between a \( z_i \) 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 \( a_i^{e_j} \neq a_{i+1}^{-e_{j+1}} \) 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:

\[
z_{i+1} = \theta^{e_i}(a_i^{e_i}) \cdots \theta^{e_j}(a_j^{e_j}) \quad \text{and} \quad z_i = \theta^{e_i'}(a_i^{e_i'}) \cdots \theta^{e_j'}(a_j^{e_j'}).\]

(\( q' = q + 1 \).

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 \( z'_i \) is \( a_{i^p-1} \) (remember \( p' = 1 - 1 = q \)) which cannot cancel with the \( a_{i^p-1} \) at the start of \( z'_i \) since \( a_{i^p-1} \) and \( a_{i^p} \) are not mutual inverses and \( e_{j+1} = 1 \) and \( e_{j'} = -1 \). If \( z_{i+1} \) is of type \( ii^{-1} \), \( ii^{-1} \), or \( iv \), then the final letter of \( z_{i+1} \) is \( a_{i^p-1} \) which cannot be \( a_{i^p-1} \) as that would contradict the maximality of \( z_i \): prepending \( \theta^{e_i'}(a_{i^p-1}) \) 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$^{-1}$, or iv appending $\theta^{e_{q+1}}(a_{i,p})$ 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 $z_{i+1}$ would be of type v contrary to maximality of $z_i$. So $z_{i+1}$ is of type i, ii$^{-1}$, iii$^{2+1}$ or iv, and therefore $i_q \geq 3$ and $e_q = -1$, and by Proposition 4.7, the final letter of $z'_{i+1}$ is $a_{i_p}$.

So if there is cancellation between $z'_{i+1}$ and $z'_i$, then the first letter of $z'_i$ must be $a_{i_q}$.

Case: $z_{i+1}$ type v, $z_i$ not type v$^{-1}$. As in the previous case, $z_i$ cannot be of type v, so $z_i$ is type i, ii, iii$^{-1}$ or iv and $i_{p+1} \geq 3$. 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$^{2+1}$ or iv word $z_i$ being contradicted.

Case: $z_{i+1}$ type v, $z_i$ type v$^{-1}$. We have that $z_{i+1} = \theta^{e_m}(a_{i_{m-1}}') \cdots \theta^{e_p}(a_{i_p})$ and $z'_i = \theta^{e_q}(a_{i_{p+1}})\theta^{e_{q+1}}(a_{i_p})$

by definition and by Proposition 4.7 in type i, respectively, and $e_q < 0$, $i_{p+1} \geq 3$, and $i_{p+1} \geq 2$.

Moreover, the first letter of $z'_i$ is $a_{i_{p+1}}$ by Proposition 4.7 in type ii. Suppose $i_{p+1}$ is 2 or 3.

Then $z_{i+1}$ has suffix $\theta^{e_q}(a_{i_p}) = \theta^{e_q}(a_{i_{p+1}})$ or something of rank at most 2 which could be prepended to $z_i$ contradicting its maximality. Suppose, on the other hand, $i_{p+1} > 3$. 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 of $z'_i$ so $i_q = i_{p+1}$ and $e_q = -1$, 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 $z'_{i+1}$, 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 \( a_mz' \) has prefix \( \theta^{-1}(a_m) \), for it will then follow that \( a_mz' \) has the same prefix.

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

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 \( \pi = a_m^{\epsilon_1}ua_m^{\epsilon_2} \) is a freely reduced word such that \( u = u(a_1, \ldots, a_{m-1}) \) and \( \epsilon_1, \epsilon_2 \in \{0, 1\} \). Define

\[
\begin{align*}
x_1 &:= a_m^{-1}\theta'(a_m) \quad \text{for } l \in \mathbb{Z}, \\
x &:= \begin{cases} 
   x_r & \text{if } r > 0 \text{ and } \epsilon_1 = 1 \\
   \text{empty word} & \text{otherwise,}
\end{cases} \\
\delta &:= \begin{cases} 
   r & \text{if } \epsilon_1 = 0 \\
   \psi_m(r) & \text{if } \epsilon_1 = 1 \text{ and } r \leq 0 \\
   r - 1 & \text{if } \epsilon_1 = 1 \text{ and } r > 0.
\end{cases}
\end{align*}
\]

Suppose \( s \in \mathbb{Z} \). Let \( \pi' \) be the freely reduced form of \( x^{-\epsilon_1}ua_m^{\epsilon_2} \). Consider the following conditions.

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

(a) \( \epsilon_2 = 0 \) and \( \hat{t}x^{-\epsilon_1}u \in H_kt^i \).
(b) \( \epsilon_2 = 1, s \leq 0 \) and \( \hat{t}x^{-\epsilon_1}u \in H_k^{\psi_m(s)} \).
(c) \( \epsilon_2 = 1, s > 0 \) and \( \hat{t}x^{-\epsilon_1}uxs \in H_kt^{\psi_m + 1} \) and \( \theta^{-1}(a_m) \) is a suffix of \( \pi \).

We have \( t\pi \in H_kt^i \) if and only if ((i, ii or iii) and \( \hat{t}\pi' \in H_kt^i \)). Moreover, \( t\pi' \in H_kt^i \) if and only if (a, b or c).

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

Next we will prove that \( t\pi \in H_kt^i \) is equivalent to \( \hat{t}\pi' \in H_kt^i \) under the assumption that (i, ii or iii) holds.

Under i, \( \epsilon_1 = 0 \), \( x \) is the empty word, and \( \delta = r \). So \( \hat{t}\pi' = \hat{t}x^{-\epsilon_1}ua_m^{\epsilon_2} = t'ua_m^{\epsilon_2} = t\pi \) and the equivalence is immediate.

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

\[ t^i \pi \in H_k t^i \]

\[ \Leftrightarrow t^i a_m u a_m^{-1} e_i \in H_k t^i \]

\[ \Leftrightarrow t^i \psi_m a_m^{e_i} \in H_k t^i \]

\[ \Leftrightarrow t^i \pi' \in H_k t^i. \]

Under iii, \( \epsilon_1 = 1, r > 0, s = x_r, \) and \( \delta = r - 1. \) Observe that

\[ t^i \pi' = t^{r-1} x_r^{-1} u a_m^{e_i} \in H_k t^i \Leftrightarrow t^i \pi = t^i a_m u a_m^{-1} e_i \in H_k t^i \]

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

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

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

Suppose \( \epsilon_2 = 0. \) Then \( t^i \pi' = t^i x_r^{-1} u a_m^{e_i} = t^i x_r^{-1} u \) and so \( t^i \pi' \in H_k t^i \) 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 \( t^i a_m \in H_k \psi_m(s) \) since \( s \leq 0 \) and so is in the domain of \( \psi_m. \) So \( t^i \pi' = t^i x_r^{-1} u a_m^{e_i} \in H_k t^i \)

if and only if \( t^i x_r^{-1} u \in H_k t^{\psi_m(s)}. \) So \( t^i \pi' \in H_k t^i \) is equivalent to Condition \( b. \)

Finally, observe that

\[ t^i \pi' = t^i x_r^{-1} u a_m^{e_i} \in H_k t^i \]

\[ \Leftrightarrow t^i x_r^{-1} u a_m^{e_i} r^{-s} \in H_k \]

\[ \Leftrightarrow t^i x_r^{-1} u a_m^{e_i} r^{-s} (t^{r-1} a_m u x_r t^{(r-1)}) \in H_k \]

\[ \Leftrightarrow t^i x_r^{-1} u x_r \in H_k t^{(r-1)} \]

because \( t^i a_m x_r t^{(r-1)} = a_m \in H_k. \) Suppose now that \( s > 0. \) The part of Condition \( c \)

cconcerning the suffix of \( x \) follows from Proposition 4.9 (applied to \( h^{-1}. \) So \( t^i \pi' \in H_k t^i \)

is equivalent to Condition \( c. \)

We conclude that \( t^i \pi \in H_k t^i \) implies (i, ii, or iii) and \( (a, b, \ or \ c). \)

4.5. Our algorithm in detail. Here we construct \textbf{Member}_k, \ where \( k \) is, as usual, any integer greater than or equal to 1, and is kept fixed. \textbf{Member}_k inputs a word \( w = w(a_1, \ldots, a_k, \ell) \)

and declares whether or not \( w \) represents an element of \( H_k. \)

Most of the workings of \textbf{Member}_k are contained in a subroutine \textbf{Push}_k, \ which inputs a valid \( \psi \)-word \( f \) and a reduced word \( v = v(a_1, \ldots, a_k), \) and declares whether or not \( f^{(0)} v \in H_k t^i \)

for some \( s \in \mathbb{Z} \) and, if so, returns a \( \psi \)-word \( f' \) with \( s = f'(0). \) (If such an \( s \) exists, it is unique by Lemma 6.1 in [12].) The key subroutine for \textbf{Push}_k \ when \( k \geq 2 \) is \textbf{Piece}_k, \ which handles the special case in which \( w \) is a rank-\( k \) piece. \textbf{Piece}_k calls a subroutine \textbf{Back}_k, \ which in turn calls a subroutine \textbf{Push}_{k-1}. \) So the construction of these three families of subroutines is inductive.

Additionally, subroutines \textbf{Prefix}_m, \ and \textbf{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 \textbf{Prefix}_m, \ \textbf{Front}_m (and also \textbf{Back}_m) are motivated by the Piece Criterion (Proposition 4.10).

Correctness of \textbf{Prefix}_m. As \( l(\theta^{-1}(a_m)) \geq i \) for \( i = 1, 2, \ldots, \) the algorithm returns the appropriate \( i \) in time \( O(\ell(\pi)^3). \) \hfill \( \square \)
Algorithm 4.1 — \textbf{Prefix}_m, m ≥ 3.
\begin{itemize}
  \item Input a rank-\(m\) piece \(\pi = a_mu a_m^{-2}\) (so, \(u = u(a_1, \ldots, a_{m-1})\) is reduced and \(e_2 \in \{0, 1\}\)).
  \item Return the largest integer \(i > 0\) (if any) such that \(\theta^{-1}(a_m)\) is a prefix of \(\pi\).
  \item Halt in time in \(O(\ell(\pi)^2)\).
\end{itemize}
\begin{itemize}
  \item construct \(\theta^{-1}(a_m)\) for \(i = 1, 2, \ldots\) until \(\ell(\theta^{-1}(a_m)) > \ell(\pi)\), and compare to \(\pi\)
  \item return the maximum \(i\) encountered (if any) such that \(\theta^{-1}(a_m)\) is a prefix of \(\pi\)
\end{itemize}

\textbf{Front}_m takes a rank-\(m\) piece \(\pi\) and \(\psi\)-word \(f\) and reduces the task of determining whether \(t^{f(0)}\pi \in H^t\) to performing a similar determination: specifically whether \(t^{f(0)}\pi' \in H^t\) where \(f(0) = \delta\) and \(\pi'\) and \(\delta\) are as per the Piece Criterion. This will represent progress because \(\pi'\) 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 — \textbf{Front}_m, \(m ≥ 3\).
\begin{itemize}
  \item Input a rank-\(m\) piece \(\pi = a_m u a_m^{-2}\) with \(e_1, e_2 \in \{0, 1\}\), and a valid \(\psi\)-word \(f = f(\psi_1, \ldots, \psi_k)\). Let \(r := f(0)\).
  \item Declare whether or not \(i\), \(ii\) or \(iii\) of the Piece Criterion holds. If so, output \(\pi'\) of the Criterion and a valid \(\psi\)-word \(f' = f'(\psi_1, \ldots, \psi_k)\) such that \(f'(0) = \delta\) of the Criterion. These satisfy \(\ell(\pi') ≤ \ell(\pi)\) and \(\ell(f') ≤ \ell(f) + 1\), and \(t^f \pi \in H^t\) if and only if \(t^{f(0)}\pi' \in H^t\).
  \item Halt in time \(O((\ell(f) + \ell(f))^2)\).
\end{itemize}
\begin{itemize}
  \item if \(e_1 = 0\) (so \(i\) holds), return \(\pi' := u a_m^{-2}\) and \(f' := f\).
  \item run \textbf{Psi}(\(f\)) to determine whether or not \(r \leq 0\)
  \item if \(e_1 = 1\) and \(r ≤ 0\) (so \(ii\) holds), return \(\pi' := u a_m^{-2}\) and \(f' := \psi_m f\)
  \item we now have that \(e_1 = 1\) and \(r > 0\) (so \(i\) and \(ii\) both fail, and it remains to test \(iii\))
  \item run \textbf{Prefix}_m on \(\pi\), which returns some \(i\)
  \item run \textbf{Psi} on input \(\psi_i f\) to check whether \(i < r\)
  \item if \(i < r\), then declare that \(i\), \(ii\) and \(iii\) all fail
  \item else \(iii\) holds, so return the reduced form \(\pi'\) of \(\theta(a_m^{-1})\pi\) and \(f' := \psi_i f\)
\end{itemize}

\textbf{Correctness of} \textbf{Front}_m.
\begin{itemize}
  \item In was established in Section 3.3 that \textbf{Psi} on input \(f\) halts in time \(O(\ell(f)^2)\).
  \item Whether \(iii\) holds depends on whether \(\theta^{-1}(a_m)\) is a prefix of \(\pi\), so that is what the remainder of the algorithm computes.
  \item \textbf{Prefix}_m halts in time \(O(\ell(\pi)^2)\).
  \item At this point we know that \(\theta^{-1}(a_m)\) is a prefix of \(\pi\), and so \(i ≤ \ell(\pi)\). Therefore, \(\ell(\psi_i f) ≤ \ell(\pi) + \ell(f)\), and so, by the bounds established in Section 3.3, \textbf{Psi} halts in time \(O(\ell(\pi) + \ell(f)^2)\).
  \item For all \(0 ≤ p ≤ q\), \(\theta^p(a_m)\) is a prefix of \(\theta^q(a_m)\): after all, for \(q ≥ 0\), \(\theta^{q+1}(a_m) = \theta^q(a_m)\theta^1(a_m)\). So, given that we know at this point that \(\theta^{-1}(a_m)\) is a prefix of \(\pi\) and \(r ≤ i\), it is the case that \(\theta^{-1}(a_m)\) is also a prefix of \(\pi\). Note that \(\theta(a_m^{-1})\pi\) is \(\theta^{-1}(a_m^{-1}) u a_m^{-2}\) of the Criterion when \(iii\) holds.
\end{itemize}

In lines 1, 3 and 9, the claimed bound \(\ell(f') ≤ \ell(\pi) + 1\) is immediate, as is \(\ell(\pi') ≤ \ell(\pi)\) in lines 1 and 3. In line 9, \(\pi'\) is the reduced form of \(\theta(a_m^{-1})\pi\) and \(\theta^{-1}(a_m)\) is a prefix of \(\pi\). Now \(\theta(a_m^{-1}) = \theta^{-1}(a_m^{-1})\theta^{-1}(a_m^{-1})\) and the length of \(\theta^{-1}(a_m^{-1})\) is at least half of \(\theta^r(a_m^{-1})\) (as \(r > 0\)), and the last letter of \(\theta^{-1}(a_m^{-1})\) is \(a_m^{-1}\). So all of the prefix \(\theta^{-1}(a_m)\) of \(\pi\) is cancelled away when \(\theta(a_m^{-1})\pi\) is freely reduced to give \(\pi'\), and \(\ell(\pi') ≤ \ell(\pi)\), as claimed.
The algorithm halts in time \( O((f(\pi) + \ell(f))^{k+4}) \) by our comments on lines 5, 6 and 7 and the fact that \( \theta'(a_{m-1}) \pi \) in the final line has length at most \( 3f(\pi) \): after all, \( \theta'(a_{m-1}) = \theta^{-1}(a_{m-1}) ) \theta^{-1}(a_{m-1}) \) and \( \ell(\theta^{-1}(a_{m-1})) \) is at most \( \ell(\theta^{-1}(a_{m-1})) \), and \( \theta^{-1}(a_{m-1}) \) is the inverse of a prefix of \( \pi \).

\[ \square \]

Next we construct \( \text{Back}_m \), \( \text{Piece}_m \) and \( \text{Push}_m \).

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

Algorithm 4.3 — \( \text{Back}_m \), \( m \geq 3 \).

\[ \text{run Push}_{m-1}(u, f) \text{ to test whether or not } t' u \in \bigcup_{s \in \mathbb{Z}} H_k t^s \]

\[ \text{if it is, let } g \text{ be the valid } \psi \text{-word it outputs such that } t' u \in H_k t^{(0)} \]

\[ \text{if } e_2 = 0, \]

\[ \text{if } t' u \in H_k t^{(0)} \text{ (so, (a) of the Criterion holds with } s = g(0)), \text{ return } f' := g \]

\[ \text{else declare } t' \pi \notin \bigcup_{s \in \mathbb{Z}} H_k t^s \]

\[ \text{halt} \]

9: we now have that \( e_2 = 1 \)

\[ \text{run } \text{Psi}(\psi_m^{-1} g) \text{ to check validity of } \psi_m^{-1} g \text{ (so whether } g(0) \in \text{Img } \psi_m) \]

\[ \text{and, if so, to check } \psi_m^{-1} g(0) \leq 0 \text{ (so, whether (b) of the Criterion holds with } s = \psi_m^{-1} g(0)) \]

12: \[ \text{if so, return } f' := \psi_m^{-1} g \]

\[ \text{run } \text{Prefix}_{m-1}(\pi^{-1}) \text{ to determine the maximum } i \text{ (if any) such that } \theta^{-1}(a_{m-1}) \text{ is a suffix} \]

\[ \text{of } \pi \]

15: \[ \text{if there is no such } i \text{ declare } t' \pi \notin \bigcup_{s \in \mathbb{Z}} H_k t^s \]

\[ \text{for } s = 1 \text{ to } i \text{ do } \]

\[ \text{run } \text{Push}_{m-1}(u', f) \text{ where } u' \text{ is the freely reduced word representing } u a_{m-1} \theta'(a_m) \]

18: \[ \text{if it outputs a } \psi \text{-word } h, \text{ run } \text{Psi}(\psi_i^{-1} h) \text{ to check if } h(0) = s - 1 \]

\[ \text{if so return } f' := \psi_i h \]

end for

21: \[ \text{declares that } t^{(0)} w \notin \bigcup_{s \in \mathbb{Z}} H_k t^s \]

For \( m \geq 3 \), correctness of \( \text{Push}_{m-1} \) (as specified below) implies correctness of \( \text{Back}_m \). The idea is to employ the Piece Criterion in the instance when \( e_1 = 0 \), and therefore \( \delta = r \), \( \pi' = \pi \) and Condition \( i \) holds. In this circumstance, the Criterion tells us that \( t' \pi \in H_k t^{(0)} \) (that is, \( t' \pi' \in H_k t^s \)) if and only if \( (a, b \text{ or } c) \) holds.
2: Referring to the specifications of $\text{Push}_{m-1}$, we see that $\ell(g) \leq \ell(u) + \ell(f)$ and $\text{rank}(g) \leq \max[\text{rank}(f), m]$.

4–6: $\text{Push}_{m-1}$ in lines 1–2 tests whether or not $t^i x^{-i} u$ (that is, $t^i u$) is in $\bigcup_{r \in Z} H_k t^r$, and, if so, it identifies the $s$ such that $t^i x^{-i} u \in H_k t^s$. The Piece Criterion then tells us that the answer to whether $t^i \pi \in \bigcup_{r \in Z} H_k t^r$ is the same, and if affirmative the $s$ agrees. (This instance of the Criterion has no real content because $t^i x^{-i} u = t^i \pi$. 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 $\text{rank}(f') \leq \max[\text{rank}(f), m]$, as required.

10–12: Again, we refer back to lines 1–2 for whether or not $t^i x^{-i} u$ (that is, $t^i u$) is in $\bigcup_{r \in Z} H_k t^r$. Assuming that it is, in fact, it is in $H_k t^{0(0)}$, 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 $t^i \pi \in \bigcup_{r \in Z} H_k t^r$, and, if affirmative, the $s$ agrees. By our comment on line 2, $\ell(f') = \ell(g) + 1 \leq \ell(f) + \ell(u) + 1 = \ell(f) + \ell(\pi)$ and $\text{rank}(f') \leq \max[\text{rank}(f), m]$, as required.

14–20: The aim here is to determine whether Condition $c$ holds—that is, whether $t^i u a_m^{-1} \theta^i(a_m) \in H_k t^{i-1}$ and $a_m^{-1} \theta^{-1}(a_m)$ is a suffix of $\pi$ for some $s > 0$—and, if so, output a $\psi$-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 $t^i \pi \in H_k t^s$, 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 $\theta^{-1}(a_m)$ of $\pi$, then $\theta^{-1}(a_m)$ is a suffix of $\pi$ 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), $t^i \pi \notin \bigcup_{r \in Z} H_k t^r$.

For each $s$ in the range $1, \ldots, i$, lines 16–20 address the question of whether or not $t^i u a_m^{-1} \theta^i(a_m) \in H_k t^{i-1}$. First $\text{Push}_{m-1}$ is called, which can be done because on freely reducing $u a_m^{-1} \theta^i(a_m)$, the $a_m^{-1}$ cancels with the $a_m$ at the start of $\theta^i(a_m)$ to give a word of rank at most $m - 1$. $\text{Push}_{m-1}$ either tells us that $t^i u a_m^{-1} \theta^i(a_m) \notin \bigcup_{r \in Z} H_k t^r$, or it gives a $\psi$-word $h$ such that $t^i u a_m^{-1} \theta^i(a_m) \in H_k t^{0(0)}$. In the latter case, $\text{Psi}$ is then used to test whether or not $h(0) = s - 1$.

By the specifications of $\text{Push}_{m-1}$, $\ell(h) \leq \ell(f) + 2(m - 1)\ell(u')$. And, as $\pi = u a_m^{-1}$ has suffix $\theta^{-1}(a_m)$, when we form $u'$ by freely reducing $u a_m^{-1} \theta^i(a_m)$, at least half of $\theta^i(a_m) = \theta^{-1}(a_m)\theta^{-1}(a_m-1)$ cancels into $\pi$. 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 $\text{rank}(f') \leq \max[\text{rank}(f), m]$, as required.

22: At this point, we know $a$, $b$, and $c$ fail for all $s \in Z$, so $t^i \pi \notin \bigcup_{r \in Z} H_k t^r$.

$\text{Back}_m$ runs $\text{Push}_{m-1}(u, f)$ once (with $\ell(u) \leq \ell(\pi)$), $\text{Psi}(\psi_m g)$ at most once (with $\ell(g) \leq \ell(\pi) + \ell(f)$), $\text{Prefix}_{m}(\pi^{-1})$ at most once, $\text{Push}_{m-1}(u', f)$ at most $i \leq \ell(\pi)$ times (with $\ell(u') < \ell(\pi)$), and $\text{Psi}(\psi_m h)$ at most $i \leq \ell(\pi)$ times (with $1 \leq s \leq \ell(\pi)$ and $\ell(h) \leq \ell(f) + \ell(\pi)$).

Other operations such as free reductions of words etc. do not contribute significantly to the running time. Referring to the specifications of $\text{Push}_{m-1}$, $\text{Psi}$, and $\text{Prefix}_m$, we see that they (respectively) contribute:

$$
\ell(\pi)O((\ell(\pi) + \ell(f))^{2m-1+r+k+1}) + \ell(\pi)O((\ell(f) + 2\ell(\pi))^{3+k} + O(\ell(\pi)^2)
= O((\ell(\pi) + \ell(f))^{2m+k})
$$

which is the claimed bound on the halting time of $\text{Back}_m$. $\square$
Algorithm 4.4 — Piece\textsubscript{m}, \(k \geq m \geq 2\).

- Input a rank-\(m\) piece \(\pi\) and a valid \(\psi\)-word \(f = f(\psi_1, \ldots, \psi_k)\).
- Declare whether or not \(t'^{(0)}\pi \in \bigcup_{l \in \mathbb{Z}} H_l t'\) and, if it is, return a valid \(\psi\)-word \(g\) such that \(t'^{(0)}\pi \in H_{\ell'(0)}\), \(\operatorname{rank}(g) \leq \max\{m, \operatorname{rank}(f)\}\), and \(\ell(g) \leq \ell(f) + 2(m-1)\ell(\pi) + 2\).
- Halts in time \(O((\ell(\pi) + \ell(f))^{3m+1})\).

\[
\begin{align*}
\text{if } m = 2 & \\
\quad \pi & = a_2^{e_1} d_1^{e_2} \text{ for some } l \in \mathbb{Z} \text{ and some } e_1, e_2 \in \{0, 1\} \\
3: & \quad g := \psi_1^{-1} \psi_2^{e_1} \psi_2^{e_2} f \\
& \quad \text{run } \Psi_1(g) \\
& \quad \text{if it declares that } g \text{ is invalid, then declare that } t'^{(0)}\pi \notin \bigcup_{l \in \mathbb{Z}} H_l t' \\
6: & \quad \text{else return } g \\
\text{if } m > 2 & \\
9: & \quad \text{run } \text{Front}_{m}(\pi, f) \\
& \quad \text{if it declares that } i, ii \text{ and } iii \text{ of the Piece Criterion all fail} \\
& \quad \text{declare that } t'^{(0)}\pi \notin \bigcup_{l \in \mathbb{Z}} H_l t' \text{ and halt} \\
12: & \quad \text{else run } \text{Back}_{m} \text{ on the output } (\pi', f') \text{ of } \text{Front}_{m} \text{ and return the result}
\end{align*}
\]

The correctness of Piece\textsubscript{2}. By applying Proposition 3.1 repeatedly, we see that \(t'^{(0)}\pi \in H_l t'\) if and only if \(t'\psi_i f'^{(0)} a_{n_i}^{r_i} \in H_{\ell' t'}\), since \(\psi_i^{a_i} f\) is valid as the domains of \(\psi_i\) and \(\psi_j\) are \(\mathbb{Z}\). So, by Corollary 3.2, \(t'^{(0)}\pi \in H_{\ell' t'}\) if and only if \(g = \psi_2^{-1} \psi_1^{e_1} \psi_2^{e_2} f\) is valid and \(s = \psi_2^{-1} \psi_1^{e_1} \psi_2^{e_2} f(0)\).

It halts in time \(O((\ell(\pi) + \ell(f))^{4+k})\) because \(\Psi_1\) halts in time \(O((\ell(\pi) + \ell(f))^{4+k})\) on input \(\psi_2^{-1} \psi_1^{e_1} \psi_2^{e_2} f\) by the bounds established in Section 3.3. □

For \(k \geq m \geq 3\), correctness of Back\textsubscript{m} implies correctness of Piece\textsubscript{m}. It follows from the specifications of Front\textsubscript{m} and Back\textsubscript{m}, that they combine in the manner of Piece\textsubscript{m} to declare whether or not \(t'^{(0)}\pi \in \bigcup_{l \in \mathbb{Z}} H_l t'\), 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 Front\textsubscript{m} and Back\textsubscript{m}, to deduce that Piece\textsubscript{m} halts in time

\[
O((\ell(\pi) + \ell(f))^{\max(k+4+2m+k)}) = O((\ell(\pi) + \ell(f))^{2m+k}).
\]

Before proving the correctness of Push\textsubscript{1}, we restate Lemma 6.2 of [12]:

**Lemma 4.11** (Lemma 6.2 of [12]). Let \(w = w(a_1, \ldots, a_k)\) be a non-empty freely reduced word of rank \(k\) where \(r, s\) are integers such that \(t'w \in H_t'\). Let \(\pi_1 \pi_2 \cdots \pi_n\) be the partition of \(w\) into pieces. Then there exist integers \(r_0, r_1, \ldots, r_n = s\) such that \(t'^{r_i} w_{r_{i+1}} \in H_{t'^{r_{i+1}}}\) for each \(i\).

The lemma says that in order to check whether \(t'^{r_i} w\) is in \(H_{t'}\), it suffices to work piece by piece, which is precisely what Push\textsubscript{m} does.

The correctness of Push\textsubscript{1}. The case \(m = 1\) is handled in lines 1–2. The point is that in \(G_k\) we have \(t'^{(0)} a_1^{l_1} f'^{(0)} a_2^{r_2} \in H_{t'^{(0)} l_1} H_{t'^{(0)} r_2}\) since \(g(0) = \psi_1^{e_1} f = f(0) - l_1\). That it halts within the time bound is clear. □

For \(k \geq m \geq 2\), correctness of Piece\textsubscript{m} implies correctness of Push\textsubscript{m}. This algorithm runs in accordance with Lemma 6.2 of [12] as we described in Section 4.1.
Finally, we are ready for:

### 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 \( \psi \)-word \( f = f(\psi_1, \ldots , \psi_k) \).
- Declare whether or not \( t^{(0)}v \in \bigcup_{t \in \mathbb{Z}} H_t t^r \). If it is, return a valid \( \psi \)-word \( g \) with \( \ell(g) \leq \ell(f) + 2m\ell(v) \), rank(\( g \)) \leq \max \{ m, \text{rank}(f) \} \) and \( t^{(0)}v \in H_t t^{(0)} \).
- Halt in time \( O((\ell(v) + \ell(f))2m+k+1) \).

\[
\begin{aligned}
\text{if } m = 1 \ (\text{and so } v = a'_1 \ \text{for some } l \in \mathbb{Z}) \\
\text{declare yes, return } g := \psi_1^f & \\
3: \\
\text{if } m > 1 \\
\text{let } \pi_1, \ldots , \pi_p \text{ be the rank}-m \text{ decomposition of } v \text{ into pieces as per Section 4.1} & \\
6: \\
f_0 := f & \\
\text{for } i = 1 \text{ to } p \\
\text{run } \texttt{Piece}_m(\pi_i, f_{i-1}) & \\
9: \\
\text{if it declares } t^{(i-1)}(\pi_i) \not\in \bigcup_{t \in \mathbb{Z}} H_t t^r, \ 	ext{declare } t^{(0)}w \not\in \bigcup_{t \in \mathbb{Z}} H_t t^r \text{ and halt} & \\
\text{else set } f_i \text{ to be its output} & \\
\text{end for} & \\
12: \\
\text{return } g := f_p & \\
\end{aligned}
\]

By the specifications of \texttt{Piece}_m, after the \( i \)-th iteration of the for loop,

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

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

\[
\ell(\pi_i) + \ell(f_i) \leq \ell(\pi_1) + \ell(\pi_1) + \cdots + \ell(\pi_{i-1}) + \ell(f) + i - 1 = O((\ell(v) + \ell(f))).
\]

So \texttt{Push}_m \text{ halts in time } O((\ell(v) + \ell(f))2m+k+1). \quad \Box

**Correctness of \texttt{Piece}_m for } 2 \leq m \leq k, \text{ of } \texttt{Push}_m \text{ for } 1 \leq m \leq k, \text{ and of } \texttt{Back}_m \text{ for } 3 \leq m \leq k.**

We established the correctness of \texttt{Push}_1 \text{ and } \texttt{Piece}_3 \text{ individually. The implications proved above give the correctness of the others by induction in the order:}

\[
\texttt{Piece}_2 \Rightarrow \texttt{Push}_2 \Rightarrow \texttt{Back}_3 \Rightarrow \texttt{Piece}_3 \Rightarrow \texttt{Push}_3 \Rightarrow \texttt{Back}_4 \Rightarrow \cdots. \quad \Box
\]

Finally, we are ready for:

### Algorithm 4.6 — \texttt{Member}_c, \ 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 \( O(\ell(w)3^k) \).

\[
\begin{aligned}
\text{convert } w \text{ to normal form } t^r v \text{ where } v = v(a_1, \ldots , a_k) \text{ is reduced, } r \in \mathbb{Z}, \text{ and } t^r v = w \text{ in } G_k, \text{ as described at the start of Section 4.1} \\
\text{if } w \not\in H_k \text{ and halt} & \\
3: \\
\text{run } \texttt{Push}_i(v, f) & \\
\text{if it outputs a (necessarily valid) } \psi \text{-word } g & \\
\text{then run } \texttt{Psi}_i(g) \text{ to test whether } g(0) = 0 & \\
6: \\
\text{if so, declare } w \not\in H_k \text{ and halt} & \\
\end{aligned}
\]
Suppose we have a word $\Gamma$ for some input word, and so proves Theorem 3.

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

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

It either declares that $v' \notin \bigcup_{k \in \mathbb{Z}} H_k t^k$, and so $w \notin H_k$, or it returns a valid $\Psi$-word $g$ such that $w \in H_k t^k(0)$ and $f(g) \leq f(\ell) + 2k(\ell(v)) \leq \ell(w) + 2k(\ell(w)\ell(w)+1)^{k-1} = O(\ell(w)^3)$. 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 $\Psi(g)$ to check. This call on $\Psi$ halts in time $O((\ell(w)^3)^{k+1}) = O(\ell(w)^{2k+4})$ when $k > 1$ and in time $O(\ell(w))$ when $k = 1$. So, max $\{k^2 + 4k, 3k^2 + k\} = 3k^2 + k$ for all $k > 1$. $\text{Member}_k$ halts in time $O(\ell(w)^{3k+1})$, as required. $\square$

5. Conclusion

The construction and analysis of $\text{Member}_k$ in the last section solves the membership problem for $H_k$ in $G_k$ in polynomial time, indeed in $O(n^{3k+1})$ time, where $n$ is the length of the input word, and so proves Theorem 3.

Here is why a polynomial time (indeed $O(n^{3k+1})$ time) solution to the word problem for $G_k$ 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 $G_k$. Recall that $G_k$ is the HNN-extension of $G_1$ 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 $G_1$, then it has a subword $p^{\pm 1}wp^{\pm 1}$ such that $w = w(a_1, \ldots, a_k, t)$ and represents an element of $H_k$.

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

References


[22] P. Schupp. personal communication.