COMPRESSED DECISION PROBLEMS IN HYPERBOLIC GROUPS

DEREK HOLT, MARKUS LOHREY, AND SAUL SCHLEIMER

Abstract. We prove that, for any hyperbolic group, the compressed word and the compressed conjugacy problems are solvable in polynomial time. We also prove that the compressed simultaneous conjugacy and the compressed centraliser problems are solvable in polynomial time. Finally, we prove that, for any infinite hyperbolic group, the compressed knapsack problem is NP–complete.

Mathematics Subject Classification – MSC2020: 20F10; 20F67

1. Introduction

1.1. **Background.** Suppose that G is a finitely generated group. Let Σ be a finite generating set which is *symmetric*: if a lies in Σ then so does a^{-1}. The *word problem* for G asks, given a word $w \in \Sigma^*$, if w represents the identity in G. This, along with the *conjugacy* and *isomorphism problems*, was set out by Dehn [16] in 1911. These three decision problems are fundamental in group theory generally [54, 11].

However, Dehn's claimed justification was that solutions to these problems have applications in what is now called *low-dimensional topology*. Dehn's techniques, in particular his solution to the word problem in surface groups, were greatly generalised by Gromov. In [29] Gromov introduced what are now called *word-hyperbolic* or *Gromov hyperbolic* groups. (We will simply call these *hyperbolic* groups. See Section 3.) With this and other innovations Gromov revived the strictly geometric study of groups. For example, he characterised hyperbolic groups as being exactly those that satisfy a linear isoperimetric inequality; see [29, Section 6.8.M] as well as [60, 9, 62]. Gromov also showed that, in certain models of *random groups*, all groups are almost surely hyperbolic; see [29, Section 5.5.F] as well as [61].

Another theme in geometric group theory is the subject of *distortion*. As a concrete example, consider the Baumslag-Solitar group [6]

$$G = \left\langle a, b \,\middle|\, b^{-1}ab = a^2 \right\rangle$$

The subgroup $\langle a \rangle$ is exponentially distorted in G, in the sense that the element $a^{2^n} =_G b^{-n}ab^n$ has length 2^n as an element of $\langle a \rangle$ but length $2n+1$ as an element of G. Thus, to solve the word problem efficiently in G it seems necessary to record exponents of a, say in binary; see also [70].

The second author has been supported by the DFG research project LO 748/13-1.

So, seeking to solve the word problem in groups leads us to consider *compressed words*: elements of the group given by some useful succinct representation. One popular such representation is by *straight-line programs*; we give definitions and examples of these in Section 4.1. We will call the word problem for group elements that are represented by straight-line programs the *compressed word problem*.

The motivating result for this paper is a theorem due to the second author [47, Theorem 4.5]. Let F_n be the free group on n generators. Lohrey gives a polynomial-time algorithm to solve the compressed word problem for F_n; in fact this problem is P–complete. Building on this, the third author shows in [66, Theorems 5.2 and 6.1] that the word problems for $\mathrm{Aut}(F_n)$ and $\mathrm{Out}(F_n)$ can be solved in polynomial time and then goes on to show that the compressed word problem for closed surface groups can be solved in polynomial time. This also gives a new solution to the word problem in mapping class groups of surfaces.

This sequence of results closely parallels Dehn's original development, but in the compressed setting.

1.2. **This paper.** Suppose that w is a word in the generators of G. We say w is *shortlex reduced* if it is shorter than, or of the same length and lexicographically earlier than, any other word representing the same group element; see Definition 2.2. Suppose that $\mathcal{G}$ is a straight-line program over Σ. Then we denote the output of $\mathcal{G}$ by $\mathsf{eval}(\mathcal{G})$. Here is our main result.

Theorem 5.7. *Let G be a hyperbolic group, with symmetric generating set Σ. There is a polynomial-time algorithm that, given a straight-line program $\mathcal{G}$ over Σ, finds a straight-line program $\mathcal{H}$ so that $\mathsf{eval}(\mathcal{H})$ is the shortlex reduction of $\mathsf{eval}(\mathcal{G})$.*

This was previously announced without proof in [49, Theorem 4.12]. From this theorem we deduce the following.

Corollary 5.8. *Let G be a hyperbolic group. Then the compressed word problem for G can be solved in polynomial time.*

In the recent paper [38] the first author, with Sarah Rees, has generalised the techniques of this paper to relatively hyperbolic groups where all peripheral groups are free abelian. So, for a knot $K \subset S^3$, the compressed word problem for the knot complement is polynomial time. This gives a further parallel with Dehn's programme for low-dimensional topology via the study of the fundamental group.

1.3. **Applications.** Given these results, in Section 6.2 we deal with the compressed versions of several other algorithmic problems. Recall that the *order problem* for a group G asks us, given an element $g \in G$, to compute the order of g. Since hyperbolic groups only have torsion elements of bounded order we can prove the following.

Corollary 6.1. *Let G be a hyperbolic group. Then the compressed order problem for G can be solved in polynomial time.*

The first author, with Epstein, proved in [23] that the conjugacy problem in hyperbolic groups is linear time. If, in the conjugacy problem, we replace the given pair of elements by a pair of finite ordered lists of elements then we obtain the *simultaneous conjugacy problem*. See [44] and its references for a discussion of this problem, for various classes of groups.

The *centraliser problem* asks for the centraliser of a given finitely generated subgroup of G. Holt and Buckley proved in [10] that the simultaneous conjugacy problem as well as the centraliser problem for hyperbolic groups is linear time.

Using the results of [23, 10], and our work above, we solve the compressed versions of these problems.

Theorem 6.3. *Let G be a hyperbolic group. Then the compressed simultaneous conjugacy problem for G can be solved in polynomial time. Moreover, if the two input lists are conjugate, then we can compute a straight-line program for a conjugating element in polynomial time.*

Theorem 6.4. *Let G be a hyperbolic group. Then the compressed centraliser problem for G can be solved in polynomial time.*

We remark that, for finitely generated nilpotent groups, the (compressed) simultaneous conjugacy problem is solvable in polynomial time [53, Theorem 7].

As suggested in [66, Remark A.5], the word problem for a finitely generated subgroup of the automorphism group $\mathrm{Aut}(G)$ is polynomial-time reducible to the compressed word problem for G. Similarly, the word problem for a finitely generated subgroup of the outer automorphism group $\mathrm{Out}(G)$ is polynomial-time reducible to the compressed simultaneous conjugacy problem for G; see [33, Proposition 10].

Note that, if G is hyperbolic then $\mathrm{Aut}(G)$, and thus $\mathrm{Out}(G)$, is finitely generated; see [15, Corollary 8.4]. We deduce the following.

Corollary 1.1. *Let G be a hyperbolic group. Then the word problems for $\mathrm{Aut}(G)$ and $\mathrm{Out}(G)$ can be solved in polynomial time.* □

Our final application is to knapsack problems. Suppose that G is a finitely generated group. The given input is a list $(u_0, u_1, u_2, \ldots, u_k)$ of words over the generators of G. We are asked if there are natural numbers n_i such that

$$u_0 =_G u_1^{n_1} u_2^{n_2} \cdots u_k^{n_k}$$

When G is hyperbolic, the knapsack problem can be solved in polynomial time; see [57, Theorem 6.1].

In the compressed knapsack problem, the words u_i are represented by straight-line programs. For the special case $G = \mathbb{Z}$ this problem is a variant

of the classical knapsack problem for binary encoded integers, which is NP–complete [42, page 95]. Using this, and our results above, we prove the following.

Theorem 6.5. *Let G be an infinite hyperbolic group. Then the compressed knapsack problem for G is* NP*–complete.*

1.4. **Related work.** We here give a brief overview of previous work. For a more in-depth treatment, we refer to [48, 49].

1.4.1. *Compressed word problems.* The use of straight-line programs in group theory dates back to, at least, the methods developed by Sims [67] for computing with a subgroup of the symmetric group S_n defined by generators. The first step in virtually all of the algorithms developed by Sims is to expand the given list of generators to a longer list (a *strong generating set*) by defining a sequence of new generators as words in the existing generators. Straight-line programs were later used, again in the context of finite groups, by Babai and Szemeredi [3] in the proof of their *Reachability Theorem.*

Note that the compressed word problem for a group G is decidable if and only if the word problem for G is decidable. However, the computational complexity of the compressed word problem for G can be strictly more difficult than the word problem itself. We return to this topic below.

It is interesting to note that the compressed word problem for a group G is exactly the *circuit evaluation problem* for G. For finite groups the compressed word problem, and thus the circuit evaluation problem, is nearly linear time. In fact, more is known. The parallel complexity of the circuit evaluation problem over finite groups is investigated in [7]. If G is a finite solvable group, then the compressed word problem for G belongs to the parallel complexity class $\mathrm{DET} \subseteq \mathrm{NC}^2$. If G is finite and not solvable, then the compressed word problem for G is P–complete.

We now turn our attention to infinite, but finitely generated, groups. As mentioned above, the word problem for a finitely generated subgroup of $\mathrm{Aut}(G)$ is polynomial-time reducible to the compressed word problem for G. A similar reduction exists for certain group extensions [49, Theorem 4.8 and 4.9]. These results on automorphisms are tightly connected to the study of distortion of subgroups, mentioned above.

Beyond hyperbolic groups, there are several important classes of groups where the compressed word problem can be solved in polynomial time. These include the following.

- Finitely generated nilpotent groups [49, Section 4.7]. Here, the compressed word problem belongs to the parallel complexity class DET [45].
- Virtually special groups; that is, finite extensions of finitely generated subgroups of right-angled Artin groups [49, Corollary 5.6]. Right-angled Artin groups are also known as graph groups or partially commutative groups. The class of virtually special groups contains

all Coxeter groups [32], one-relator groups with torsion [71], fully residually free groups [71], and fundamental groups of hyperbolic three-manifolds [1]. Note that the case of fully residually free groups is independently due to Macdonald [52].

Furthermore, the class of groups with polynomial time compressed word problem is closed under the following operations.

- Graph products [49, Chapter 5].
- Amalgamated free products (or HNN–extensions) where the edges groups are finite [49, Chapter 6].

We also note that, for finitely generated linear groups, the compressed word problem belongs to the complexity class coRP [49, Theorem 4.15]. That is, there is a randomised polynomial-time algorithm that may err with a small probability on negative input instances.

1.4.2. *Hardness results.* Certain hardness results for the compressed word problem are known or suspected.

- The compressed word problem for every restricted wreath product $G \wr \mathbb{Z}$ with G finitely generated nonabelian is coNP–hard [49, Theorem 4.21]. For G finite non-solvable (or free of rank two) the problem is PSPACE–complete [4, Corollary B]; the authors obtain the same result for Thompson's group F, the Grigorchuk group and the Gupta-Sidki groups.

  On the other hand: the uncompressed word problem for the Grigorchuk group can be solved in logarithmic space [26]. Also, if G is finite then the uncompressed word problem for $G \wr \mathbb{Z}$ belongs to the circuit complexity class NC^1 [68]. Thus we have examples of groups where the compressed word problem is provably more difficult than the uncompressed word problem.
- There exist automaton groups with an EXPSPACE–complete compressed word problem [69].

  On the other hand, the uncompressed word problem for any automaton group belongs to PSPACE. Again this gives examples where the compressed word problem is provably more difficult than the uncompressed.
- The compressed word problem for the linear group $\mathrm{SL}(3, \mathbb{Z})$ is equivalent, up to polynomial-time reductions, to the problem of *polynomial identity testing*. This last is the decision problem of whether a given circuit over the polynomial ring $\mathbb{Z}[x_1, \ldots, x_n]$ evaluates to the zero polynomial [49, Theorem 4.16]. The existence of a polynomial-time algorithm for polynomial identity testing is an outstanding open problem in the area of algebraic complexity theory.

  On the other hand, the uncompressed word problem for $\mathrm{SL}(3, \mathbb{Z})$ is polynomial time.

1.4.3. *Knapsack problems over groups.* The uncompressed knapsack problem has been studied for various classes of groups; see [24, 25, 46]. For non-elementary hyperbolic groups, the knapsack problem lies in LogCFL (the logspace closure of the class of context-free languages); see [50, Theorem 4.1]. The second author further shows, in [51, Theorem 3.1], that the compressed knapsack problem for every virtually special group belongs to NP.

1.4.4. *Compressing integers.* In addition to straight-line programs, there are other methods of compression that arise in significant ways in computational group theory. Here we will mention just a few with particular relevance to the word problem. These are techniques for recording extremely large integers, as opposed to recording long words.

The binary representation of an integer n can be translated into a straight-line program $\mathcal{G}_n$ of size $O(\log n)$ with output a^n. Following our discussion of circuit evaluation above, we could replace "concatenation of strings" by the primitive operator "addition of integers". Likewise, we replace the alphabet $\{a\}$ by the alphabet $\{1\}$. This transforms the straight-line program $\mathcal{G}_n$ into an additive circuit with output n.

If we allow multiplication as well as addition gates we obtain *arithmetic circuits*. For example, a circuit with n gates can produce an integer of size 2^{2^n} using iterated squaring. *Power circuits*, the topic of [59], replace multiplication of x and y by the operation $x \cdot 2^y$: that is, shifting the first input by the second. Thus a power circuit of depth n can represent an integer of the size of a tower of exponentials of height n. The same authors use their new theory, in [58], to show that the word problem in Baumslag's group [5]

$$\left\langle a, b \;\middle|\; \left(b^{-1}ab\right)^{-1} a \left(b^{-1}ab\right) = a^2 \right\rangle$$

is polynomial time. We note that Baumslag's group has a non-elementary Dehn function [64]; this demonstrates one of the many possible separations between the computational and the geometric theories of a group.

Again exploiting various properties of power circuits, the authors of [18] give a cubic time algorithm for the word problem in Baumslag's group [18, Theorem 16]. They also show that the word problem for Higman's group [34]

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

is polynomial time [18, Theorem 19].

Of course, even more extreme compression is possible and this leads to polynomial-time algorithms for even more extreme groups. The authors of [19, 20] construct certain HNN–extensions of the *hydra groups* [21]

$$H_k = \langle a, b \,|\, [\cdots[[a, b], b], \ldots, b] = 1 \rangle,$$

where $[x, y] = x^{-1}y^{-1}xy$ is a *commutator* and $[\cdots[[a, b], b], \ldots, b]$ is a nested commutator of depth k, for which the Dehn function grows roughly like the Ackermann function and the word problem is still solvable in polynomial time. For this, they use a compression scheme for integers that yields a compression ratio of order of the Ackermann function on some integers.

1.5. **Acknowledgements.** An extended abstract of this paper appeared in [37]. The third author thanks the first two for their patience during the writing of this paper.

2. General notation

We include zero in the set of natural numbers; that is, $\mathbb{N} = \{0, 1, 2, \ldots\}$.

2.1. **Words.** Suppose that Σ is an *alphabet*; the elements of Σ are called *letters*. We write Σ^* for the *Kleene closure* of Σ; that is, the set of all finite words over Σ. We call any subset $L \subseteq \Sigma^*$ a *language* over Σ.

For any alphabet Σ, we use $\varepsilon \in \Sigma^*$ to denote the *empty word*. Suppose that u, v, and w are words over Σ. We denote the *concatenation* of u and v by $u \cdot v$; we often simplify this to just uv ($u \cdot v$ is sometimes preferred for better readability). So, for example, $w \cdot \varepsilon = \varepsilon \cdot w = w$. We say that u is a *factor* of v if there are words x and y so that $v = xuy$. We say that u is a *rotation* of v if there are words x and y so that $v = xy$ and $u = yx$. We have the following easy but useful result.

Lemma 2.1. *Let u and v be words over Σ. Then u is a rotation of v if and only if $|u| = |v|$ and u is a factor of $v \cdot v$.* □

Suppose that $w = a_0 \cdot a_1 \cdots a_{n-1}$ lies in Σ^*, where the a_i are letters. Then we define $|w|$ to be the *length* of w; that is, $|w| = n$. For any i between zero and $n - 1$ (inclusive) we define $w[i] = a_i$. Note that the empty word ε is the unique word of length zero.

We now define the *cut operators*. Let w be a word, as above, and let i and j be indices with $0 \leq i \leq j \leq n = |w|$. We define $w[i : j] = a_i \cdots a_{j-1}$. We use $w[: j]$ to denote $w[0 : j]$, the *prefix* of length j. We use $w[i :]$ to denote $w[i : n]$, the *suffix* of length $n - i$. Note that $w[i : i] = \varepsilon$ and $w = w[: i] \cdot w[i :]$.

Suppose that Σ is a finite alphabet equipped with a total order $<$ (the concrete choice of $<$ will never be important for us).

Definition 2.2. We define the *shortlex order* on Σ^* as follows. For words u, v we have $u <_{\mathsf{slex}} v$ if

- $|u| < |v|$ or
- $|u| = |v|$ and there are words $x, y, z \in \Sigma^*$ and letters $a, b \in \Sigma$ so that
  - $u = xay$,
  - $v = xbz$, and
  - $a < b$.

Note that shortlex is a *well-order* on Σ^*: that is, every nonempty subset of Σ^* has a unique shortlex least element.

2.2. **Finite state automata.** We refer to [40] for background in automata theory. A *(deterministic) finite state automaton* is a tuple $M = (Q, \Sigma, q_0, \delta, F)$, where

- Q is a finite set of *states*,

- Σ is a finite alphabet,
- $q_0 \in Q$ is the *initial* state,
- $\delta\colon Q \times \Sigma \to Q$ is a *transition* function, and
- $F \subseteq Q$ is the set of *accept* states.

Intuitively, if the automaton M is "in" state $q \in Q$, and receives input $a \in \Sigma$, then it transitions to the new state $\delta(q, a)$. We extend δ to a function $\delta'\colon Q \times \Sigma^* \to Q$ recursively. That is, for any state q, word w, and letter a we have

- $\delta'(q, \varepsilon) = q$ and
- $\delta'(q, wa) = \delta(\delta'(q, w), a)$.

Since δ and δ' agree on words of length at most one, we will suppress δ' in what follows and instead reuse δ. We define

$$L(M) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$$

to be the language *accepted* by M. Intuitively, if w lies in $L(M)$ then w, when input into M, takes it from the initial state to an accept state.

We say that a language $L \subseteq \Sigma^*$ is *regular* if there exists a finite state automaton M so that $L = L(M)$.

3. Hyperbolic groups

We refer to [2] as a general reference on (word) hyperbolic groups.

Let G be a finitely generated group. Let 1_G denote the identity element of G. Let Σ be a finite, *symmetric* generating set for G. That is, if a lies in Σ then so does a^{-1}. For two words $u, v \in \Sigma^*$, we will use $u =_G v$ to mean that u and v represent the same element of G. We fix a total order $<$ on Σ.

The *(right) Cayley graph* $\Gamma = \Gamma(G, \Sigma)$ of G with respect to Σ is defined as follows.

- The vertices of Γ are the elements of G.
- The undirected edges of Γ are of the form $\{g, ga\}$ for $g \in G$ and $a \in \Sigma$.

We will label a directed edge (g, ga) with the letter a. Note that G acts, by graph automorphisms, on Γ on the left.

Giving all edges length one makes Γ into a geodesic metric space. We do this in such a way so that the action of G is by isometries. The distance between two points p, q is denoted $d_\Gamma(p, q)$. For $g \in G$ we define $|g| = d_\Gamma(1, g)$. We deduce that $|g|$ is the smallest length among all words $w \in \Sigma^*$ that represent g. Fix $r \geq 0$. The ball of radius r in Γ is the set

$$\mathsf{B}(r) = \mathsf{B}_\Gamma(r) = \{g \in G : |g| \leq r\}$$

Fix a word $w \in \Sigma^*$. We define $P_w \subseteq \Gamma$ to be the path starting at 1_G which is labelled by w. Thus the path $g \cdot P_w$ starts at g and is again labelled by w. In general, we will take $P\colon [0, n] \to \Gamma$ to be an edge path from $P(0)$ to $P(n)$. In particular, we must allow real $t \in [0, n]$ as we traverse edges. We use $\bar{P}$ to denote P with its parametrisation reversed. Note that $\bar{P}_w = g_w \cdot P_{w^{-1}}$.

We call a path P *geodesic* if for all real $t \geq 0$ we have $d_\Gamma(P(0), P(t)) = t$. Suppose that the word $w \in \Sigma^*$ represents the group element $g_w \in G$. We say that $w \in \Sigma^*$ is *geodesic* if the path P_w is geodesic. We say that $w \in \Sigma^*$ is *shortlex reduced* if for all $u \in \Sigma^*$ the equality $g_u = g_w$ implies $w \leq_{\mathsf{slex}} u$. We use $\mathsf{slex}(w)$ to denote the shortlex reduced representative of g_w.

Remark. Suppose that w is geodesic or shortlex reduced. Suppose that u is a factor of w. Then u is also, respectively, geodesic or shortlex reduced.

A *geodesic triangle* in Γ consists of three *vertices* $p, q, r \in G$ and three *sides* $P, Q, R \subset \Gamma$. The sides are geodesic paths connecting the vertices; see Figure 3.1. Fix $\delta \geq 0$. We now follow [2, Definition 1.3]. We say that a geodesic triangle is *δ–slim*, if every point x in the side P is distance at most δ from some point of $R \cup Q$, and similarly for the sides Q and R.

Fix G and Σ as above. We say G is *δ–hyperbolic* if every geodesic triangle in the Cayley graph $\Gamma = \Gamma(G, \Sigma)$ is δ–slim. Finally, we simply say G is *hyperbolic*, if it is δ–hyperbolic for some $\delta \geq 0$. For example, the group G is 0–hyperbolic (with respect to Σ) if and only if G is a free group, freely generated by (half of) Σ.

Remark 3.1. In his article [29], Gromov proves that hyperbolic groups have many good properties. In Corollary 2.3.B he states that such groups satisfy a linear isoperimetric inequality; hence they have solvable word problem. In Corollary 2.3.E he shows that the notion of hyperbolicity is independent of the choice of finite generating set. In Section 7.4.B he proves that they have solvable conjugacy problem. For another exposition of these results (excepting the conjugacy problem) we refer to [2]: see Theorem 2.5, Proposition 2.10, and Theorem 2.18 of that work. For an exposition of the conjugacy problem we refer to [23].

We will need a seemingly stronger condition on our geodesic triangles, called *δ–thinness*. We here follow [2, Definition 1.5]. Suppose again that we have a geodesic triangle with vertices $p, q, r \in G$ and with sides $P, Q, R \subset \Gamma$; see Figure 3.1. Let $c_P \in P$, $c_Q \in Q$, and $c_R \in R$ be the unique points so that

$$d_\Gamma(p, c_Q) = d_\Gamma(p, c_R), \quad d_\Gamma(q, c_R) = d_\Gamma(q, c_P), \quad d_\Gamma(r, c_P) = d_\Gamma(r, c_Q)$$

We call these the *meeting points* of the triangle. Note that the meeting points may be elements of G or midpoints of edges of Γ. Suppose that $x \in P$ and $y \in Q$ are points with

- $d_\Gamma(r, x) = d_\Gamma(r, y) = t$ and
- $t \leq d_\Gamma(r, c_P) = d_\Gamma(r, c_Q)$.

Then we call x and y *corresponding points* with respect to r. Note that if one of x or y lies in G then so does the other. We make the same definition with respect to the vertices p and q. Note that the three meeting points are all in correspondence. Fix $\delta \geq 0$. The triangle is called *δ–thin* if for all corresponding pairs (x, y) we have $d_\Gamma(x, y) \leq \delta$. See Figure 3.1; there the dotted arcs indicate corresponding pairs. Note that a δ–thin triangle

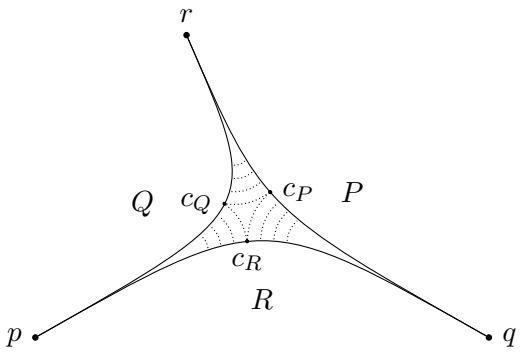

FIGURE 3.1. A geodesic triangle in a hyperbolic metric space. Note how the three sides "bow in" to a common centre. Dotted lines represent paths of length at most δ between corresponding points.

is δ–slim. A converse also holds: every geodesic triangle in a δ–hyperbolic space is 4δ–thin; see [2, Proposition 2.1].

We now fix a group G and a symmetric generating set Σ; we assume that G is δ–hyperbolic. We choose δ large enough to ensure that all geodesic triangles in Γ are δ–thin.

Remark. From a computational viewpoint, hyperbolic groups have many nice properties. For example, their word problems can be solved in linear time [2, Theorem 2.18] as can their conjugacy problems [23]. (Here we gloss over the details of the required model of computation.) In a more recent and noteworthy achievement, their isomorphism problem has also been solved; see [14, 15]. Thus all three of Dehn's fundamental problems have been settled positively for hyperbolic groups.

Other positive results include the *simultaneous conjugacy problem* [10] and the *knapsack problem* [50]. We will return to both of these below.

Note that the compressed word problem easily reduces to the problem of checking the solvability of a system of equations. There is a substantial body of work on the latter, over hyperbolic groups. Dahmani and Guirardel [14] prove (building on earlier work of [65]) that the problem is decidable. The compressibility by straight-line programs of solutions of equations in hyperbolic groups is studied in [17]. Ciobanu and Elder [13] give a complete description of the set of all solutions of a given system of equations over a hyperbolic group. They obtain, as a corollary, a polynomial-space algorithm for deciding the existential theory of a hyperbolic group.

The following results come from the fact that hyperbolic groups have automatic structures with respect to any shortlex ordering [22, Theorem 3.4.5 and Corollary 2.5.2].

Lemma 3.2 [22, Theorem 2.3.10]**.** *There is a polynomial-time (in fact, quadratic) algorithm that, given a word $w \in \Sigma^*$, produces* $\mathsf{slex}(w)$. □

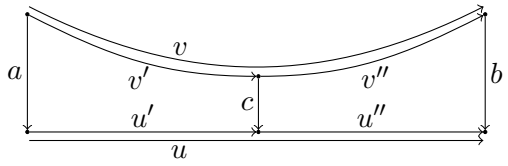

FIGURE 3.2. Splitting a geodesic quadrilateral according to Lemma 3.4.

Lemma 3.3 [22, Proposition 2.5.11 and Theorem 3.4.5]**.** *The languages in Σ^*, of geodesic words and of shortlex reduced words, are regular.* □

Remark. We will in fact need both geodesic and shortlex reduced words in our proof of Theorem 5.7. This is because the inverse of a geodesic word is again geodesic; this need not be the case for shortlex reduced words. On the other hand, shortlex reduced words provide unique representatives of group elements; this is almost never the case for geodesic words.

We will need the following standard lemma on geodesic quadrilaterals. See, for example, the proof of [2, Proposition 3.5].

Lemma 3.4. *Let $a, b, u, v \in \Sigma^*$ be geodesic words such that $vb =_G au$. Suppose that u has a factorisation $u = u'u''$ with $|u'| \geq |a| + 2\delta$ and $|u''| \geq |b| + 2\delta$. Then there exists a factorisation $v = v'v''$ and a geodesic word c so that*

- *$|c| \leq 2\delta$,*
- *$v'c =_G au'$, and*
- *$v''b =_G cu''$.*

Proof. We consider the quadrilateral with sides P_v, P_a, $g_a \cdot P_u$, and $g_v \cdot P_b$. Here g_w is the group element represented by the word w. We are given a factorisation $u = u'u''$. Set $g = g_a g_{u'}$ and note that g lies in $g_a \cdot P_u$. See Figure 3.2. Since geodesic quadrilaterals are 2δ–slim there is a group element h with $d_\Gamma(g, h) \leq 2\delta$ lying in the union of the three other sides.

We now consider cases. Suppose that h lies in $P_a - \{1_G\}$. Then the triangle inequality implies $|u'| < |a| + 2\delta$. Similarly, if h lies in $g_v \cdot P_b - \{g_v\}$, then $|u''| < |b| + 2\delta$. Both of these are contrary to hypothesis. We deduce that h lies in P_v, proving the lemma. □

The lemma has a useful corollary.

Corollary 3.5. *Let $a, b, u, v \in \Sigma^*$ be geodesic words such that $vb =_G au$. Suppose that u has a factorisation $u = u'u''u'''$ with $|u'| \geq |a| + 2\delta$, $|u''| \geq 4\delta$, and $|u'''| \geq |b| + 2\delta$. Then there exists a factorisation $v = v'v''v'''$ and geodesic words c, d so that*

- *$|c|, |d| \leq 2\delta$,*
- *$v'c =_G au'$,*
- *$v''d =_G cu''$, and*
- *$v'''b =_G du'''$.*

Proof. We prove this with two applications of Lemma 3.4. The first application gives us c. In the second application we restrict our attention to the quadrilateral with sides labelled c, b, $u'' \cdot u'''$, and the fourth side labelled by the resulting suffix of v. This gives d. □

Suppose that S is a path in Γ of length n, and i is an integer. We adopt the convention that the use of the expression $S(i)$ implies that i lies in $[0, n]$. Recall that $\bar{S}$ denotes S with its parametrisation reversed.

Lemma 3.6. *Let T be a δ–thin geodesic triangle with vertices at p, q, and r and with sides P, Q, and R. Suppose that $P(0) = q = \bar{R}(0)$, $Q(0) = r = \bar{P}(0)$, and $R(0) = p = \bar{Q}(0)$. Let j be any integer so that $d_\Gamma(R(j), \bar{Q}(j)) > \delta$. Then there are integers $i_R < i_Q$ so that $d_\Gamma(R(j), P(i_R)) \leq \delta$ and $d_\Gamma(\bar{Q}(j), P(i_Q)) \leq \delta$.*

In the statement and the proof we follow the notation of Figure 3.1.

Proof of Lemma 3.6. Since $d_\Gamma(R(j), \bar{Q}(j)) > \delta$, the group elements $R(j)$ and $\bar{Q}(j)$ do not correspond to each other. Thus $R(j)$ is strictly after the meeting point c_R along R. Similarly $\bar{Q}(j)$ is strictly after the meeting point c_Q along $\bar{Q}$. Since T is δ–thin, there are integers i_R and i_Q so that

- $R(j)$ corresponds to $P(i_R)$ and so $d_\Gamma(R(j), P(i_R)) \leq \delta$ and
- $\bar{Q}(j)$ corresponds to $P(i_Q)$ and so $d_\Gamma(\bar{Q}(j), P(i_Q)) \leq \delta$.

We deduce that $P(i_R)$ is strictly before, and $P(i_Q)$ is strictly after, c_P along P. Thus $i_R < i_Q$ and we are done. □

4. Compressed words and the compressed word problem

4.1. **Straight-line programs.** Straight-line programs offer succinct representations of long words that contain many repeated substrings. We here review the basics, referring to [49] for a more in-depth introduction.

Definition 4.1. Fix Σ, a finite alphabet. A *straight-line program* over Σ is a triple $\mathcal{G} = (V, S, \rho)$ where

- V is a finite set of *variables*, disjoint from Σ,
- $S \in V$ is the *start variable*, and
- $\rho : V \to (V \cup \Sigma)^*$ is an *acyclic production mapping*: that is, the relation

$$\{(B, A) \in V \times V \mid B \text{ appears in } \rho(A)\}$$

is acyclic. We call $\rho(A)$ the *right-hand side* of A.

Example 4.2. Let $\Sigma = \{a, b\}$ and fix $n \geq 0$. We define $\mathcal{G}_n = (\{A_0, \ldots, A_n\}, A_n, \rho)$, where $\rho(A_0) = ab$ and $\rho(A_{i+1}) = A_i A_i$ for $0 \leq i \leq n - 1$.

Definition 4.3. Given a straight-line program $\mathcal{G}$ as above, we define an *evaluation* function $\mathsf{eval} = \mathsf{eval}_\mathcal{G} \colon (V \cup \Sigma)^* \to \Sigma^*$ as follows.

- $\mathsf{eval}(a) = a$ for $a \in \Sigma$,
- $\mathsf{eval}(uv) = \mathsf{eval}(u)\mathsf{eval}(v)$ for $uv \in (V \cup \Sigma)^*$, and

- $\mathsf{eval}(A) = \mathsf{eval}(\rho(A))$ for $A \in V$.

One proves by a delicate induction that eval is well defined. We finally take $\mathsf{eval}(\mathcal{G}) = \mathsf{eval}(S)$. We call $\mathsf{eval}(\mathcal{G})$ the *output* of the program $\mathcal{G}$.

In other words, $\mathcal{G}$ is a context-free grammar that generates exactly one word $\mathsf{eval}(\mathcal{G})$ of Σ^*.

So, continuing Example 4.2 above we have $\mathsf{eval}(A_0) = ab$ and more generally $\mathsf{eval}(A_i) = (ab)^{2^i}$. Thus $\mathsf{eval}(\mathcal{G}_n) = \mathsf{eval}(A_n) = (ab)^{2^n}$. So the output has length 2^{n+1}.

We say a straight-line program $\mathcal{G} = (V, S, \rho)$ over Σ is *trivial* if S is the only variable and $\rho(S) = \varepsilon = \mathsf{eval}(\mathcal{G})$.

We say that a straight-line program is in *Chomsky normal form* if it is either a trivial program or all right-hand sides are of the form $a \in \Sigma$ or BC with $B, C \in V$. There is a linear-time algorithm that transforms a given straight-line program $\mathcal{G}$ into a program $\mathcal{G}'$ in Chomsky normal form with the same output; see [49, Proposition 3.8].

Definition 4.4. We define the *size* $|\mathcal{G}|$ of $\mathcal{G} = (V, S, \rho)$ to be the sum of the bit-lengths of the right-hand sides of ρ. Symbols from $V \cup \Sigma$ are encoded by bit strings of length $O(\log(|V| + |\Sigma|))$ using a prefix code.

Again considering Example 4.2, we see that the size of $\mathcal{G}_n$ is $O(n \log(n))$. (Note that we take into account the cost of writing out the indices of the variables A_i.) Thus we see that straight-line programs can achieve (essentially) exponential compression. The following result proves that straight-line programs can do no better; the proof follows the proof of [12, Lemma 1].[1]

Lemma 4.5. *For every straight-line program $\mathcal{G}$ we have* $|\mathsf{eval}(\mathcal{G})| \leq 3^{|\mathcal{G}|/3}$. □

As a convenient short-hand, we will refer to straight-line programs over Σ as *compressed words*.

4.2. **Algorithms for compressed words.** We will assume that all integers given as input to algorithms are given in binary. We will need to know that the following algorithmic tasks can be solved in polynomial time; see [49, Proposition 3.9].

Given a straight-line program $\mathcal{G}$ and natural numbers $i \leq j$:

- find the length $|\mathsf{eval}(\mathcal{G})|$;
- find the letter $\mathsf{eval}(\mathcal{G})[i]$;
- find a straight-line program $\mathcal{G}'$ with $\mathsf{eval}(\mathcal{G}') = \mathsf{eval}(\mathcal{G})[i : j]$.

The following proposition is also well-known [12, Lemma 2].

[1] In [12], $|\mathcal{G}|$ is defined as the sum of all lengths of right-hand sides of $\mathcal{G}$. Note that this value is less than or equal to our value of $|\mathcal{G}|$ (the bit-lengths of the right-hand sides).

Proposition 4.6. *There is a polynomial-time algorithm that, given a straight-line program $\mathcal{G}$ and a natural number $n > 0$, computes a straight-line program $\mathcal{G}_n$ with $\mathsf{eval}(\mathcal{G}_n) = \mathsf{eval}(\mathcal{G})^n$. In fact, the time required is linear in $|\mathcal{G}| + \log n$.* □

The following results are less trivial. A proof of this proposition can be found in [49, Theorem 3.11].

Proposition 4.7. *There is a polynomial-time algorithm that, given*

- *a finite alphabet Σ,*
- *a finite state automaton M over Σ, and*
- *a straight-line program $\mathcal{G}$ over Σ,*

decides if $\mathsf{eval}(\mathcal{G})$ lies in the language $L(M)$. □

We also need the following variant of Proposition 4.7.

Proposition 4.8. *There is a polynomial-time algorithm that, given*

- *a finite alphabet Σ,*
- *a finite state automaton M over Σ, and*
- *a straight-line program $\mathcal{G}$ over Σ,*

decides if $\{\mathsf{eval}(\mathcal{G})^n \mid n \in \mathbb{N}\}$ is a subset of $L(M)$.

Proof. Let $M = (Q, \Sigma, q_0, \delta, F)$ be the automaton. Suppose that $w = \mathsf{eval}(\mathcal{G})$. All non-negative powers of w belong to $L(M)$ if and only if $\delta(q_0, w^n)$ lies in F, for all $n \geq 0$.

Since Q is finite, there are natural numbers k and l, with $0 \leq k < l \leq |Q|$ such that $\delta(q_0, w^k) = \delta(q_0, w^l)$ and hence

$$\delta(q_0, w^{k+i}) = \delta(q_0, w^{l+i}) \text{ for all } i \geq 0.$$

It follows that $w^n \in L(M)$ for all $n \geq 0$ if and only if $w^n \in L(M)$ for all $0 \leq n \leq |Q|$. By Proposition 4.6, we can compute, in polynomial time and for all $0 \leq n \leq |Q|$, a straight-line program $\mathcal{G}_n$ with output $\mathsf{eval}(\mathcal{G}_n) = w^n$. Finally we use Proposition 4.7 to test, in polynomial time, if $\mathsf{eval}(\mathcal{G}_n) \in L(M)$ for these programs. □

The following result is central to our past and present work. It was independently discovered by Hirshfeld, Jerrum, and Moller [35, Proposition 12] (see also [36, Proposition 3.2]), by Mehlhorn, Sundar, and Uhrig [55, 56] (where the result is implicitly stated in terms of dynamic string data structures), and by Plandowski [63, Theorem 13].

Theorem 4.9. *There is a polynomial-time algorithm that, given straight-line programs $\mathcal{G}$ and $\mathcal{H}$, decides if $\mathsf{eval}(\mathcal{G}) = \mathsf{eval}(\mathcal{H})$.* □

We now give a version of [43, Theorem 1]; this generalises Theorem 4.9 to the so-called *fully compressed pattern matching problem*. See [41, Theorem 1.1] for a quadratic time algorithm, which is the best currently known.

Theorem 4.10. *There is a polynomial-time algorithm that, given straight-line programs $\mathcal{G}$ and $\mathcal{H}$, decides if $\mathsf{eval}(\mathcal{G})$ is a factor of $\mathsf{eval}(\mathcal{H})$. Furthermore, if it is a factor, the algorithm returns (in binary) the smallest $m \in \mathbb{N}$ so that $\mathsf{eval}(\mathcal{G})$ is a prefix of $\mathsf{eval}(\mathcal{H})[n:]$.* □

We obtain the following corollary of Theorem 4.10 and Lemma 2.1.

Corollary 4.11. *There is a polynomial-time algorithm that, given straight-line programs $\mathcal{G}$ and $\mathcal{H}$, decides if $\mathsf{eval}(\mathcal{G})$ is a rotation of $\mathsf{eval}(\mathcal{H})$. Furthermore, if it is, then the algorithm returns straight-line programs $\mathcal{H}'$ and $\mathcal{H}''$ such that*

$$\mathsf{eval}(\mathcal{H}) = \mathsf{eval}(\mathcal{H}')\mathsf{eval}(\mathcal{H}'') \qquad \textit{and} \qquad \mathsf{eval}(\mathcal{G}) = \mathsf{eval}(\mathcal{H}'')\mathsf{eval}(\mathcal{H}')$$ □

4.3. **The compressed word problem.** Suppose that G is a group and Σ is a finite symmetric generating set. The *compressed word problem* for G, over Σ, is the following decision problem.

Input: A straight-line program $\mathcal{G}$ over Σ.
Question: Does $\mathsf{eval}(\mathcal{G})$ represent the identity of G?

Note that the compressed word problem for a group G is decidable if and only if the word problem for G is decidable. As discussed in the introduction, there are in fact groups G where the compressed word problem is strictly harder than the word problem itself.

Observe that the computational complexity of the compressed word problem for G does not depend on the chosen generating set Σ. That is, if Σ' is another such, then the compressed word problem for G over Σ is logspace reducible to the compressed word problem for G over Σ' [49, Lemma 4.2]. Thus, when proving that the compressed word problem is polynomial time, we are allowed to use whatever generating set is most convenient for our purposes.

Remark 4.12. As a simple but useful tool, note that if $\mathcal{G}$ is a straight-line program over Σ with output w then there is a straight-line program $\bar{\mathcal{G}}$ with output w^{-1}.

4.4. **Cut programs.** A useful generalisation of straight-line programs are the *composition systems* of [31, Definition 8.1.2]. These are also called *cut straight-line programs* in [49]. We shall simply call them *cut programs*. They are used, for example, in the polynomial-time algorithm for the compressed word problem of a free group [47].

A *cut program* over Σ is a tuple $\mathcal{G} = (V, S, \rho)$, with V and S as in Section 4.1, and where we also allow, as right-hand sides for ρ, expressions of the form $B[i:j]$, with $B \in V$ and with $i \leq j$. We again require ρ to be acyclic. When $\rho(A) = B[i:j]$ we define

$$\mathsf{eval}(A) = \mathsf{eval}(B)[i:j]$$

with the cut operator $[i:j]$ as defined in Section 2. The *size* of a cut program $\mathcal{G}$ is the sum of the bit-lengths of the right-hand sides; as usual all natural numbers are written in binary.

We can now state a straightforward but important result of Hagenah; see [31, Algorithmus 8.1.4] as well as [49, Theorem 3.14].

Theorem 4.13. *There is a polynomial-time algorithm that, given a cut program $\mathcal{G}$, finds a straight-line program $\mathcal{G}'$ such that* $\mathsf{eval}(\mathcal{G}) = \mathsf{eval}(\mathcal{G}')$.

Theorems 4.9 and 4.13 imply that there is a polynomial-time algorithm that, given two cut programs, decides if they have the same output.

Remark 4.14. In fact, in what follows, we will only ever need the prefix and suffix cut operators $[: j]$ and $[i :]$. This is because, when using a word to represent a group element, cancellation appears where two factors meet.

We also note that iterating the cut operator can be done using arithmetic alone. That is, the cut variables

$$B[i : j][k : \ell] \quad \text{and} \quad B[i + k : i + \ell]$$

have the same evaluation. This "cut elimination" is, in some sense, the heart of the proof of Theorem 4.13.

5. The compressed word problem for hyperbolic groups

Suppose that G is a group and Σ is a finite symmetric generating set. We fix a total order $<$ on Σ. Suppose that G is δ–hyperbolic; here we take δ large enough so that all geodesic triangles are δ–thin, and we assume also that $\delta > 0$ is an integer. (This assumption is used in Case 3.2 inside of the proof of Lemma 5.3.) In what follows, we take $\zeta = 2\delta$. Recall that $\mathsf{B}(r)$ is the ball of radius r about 1_G in the Cayley graph $\Gamma = \Gamma(G, \Sigma)$.

5.1. Tethered programs.

We introduce a new type of program using the *tether* operator.

A *tethered program* over Σ is a tuple $\mathcal{G} = (V, S, \rho)$, with V and S as in Section 4.1, and where we also allow, as right-hand sides for ρ, expressions of the form $B\langle a, b\rangle$, with $B \in V$ and with $a, b \in \mathsf{B}(\zeta)$. We again require ρ to be acyclic. If $\rho(A) = B\langle a, b\rangle$ then we define

$$\mathsf{eval}(A) = \mathsf{slex}(a \cdot \mathsf{eval}(B) \cdot b^{-1})$$

We call the suffix $\langle a, b\rangle$ a *tether* operator. The *size* of a tethered program $\mathcal{G}$ is the sum of the bit-lengths of the right-hand sides; group elements in $\mathsf{B}(\zeta)$ are represented by their shortlex representatives.

Finally, in a *tether-cut* program $\mathcal{G}$ over Σ we allow right-hand sides which are words from $(V \cup \Sigma)^*$, a cut variable, or a tethered variable. It is sometimes convenient to allow more complicated right-hand sides such as

$$B[: i]\langle a, b\rangle \cdot C[j :]\langle c, d\rangle$$

Again, we define the size of $\mathcal{G}$ as the sum of the bit-lengths of the right-hand sides. In what follows we will assume that all programs arising are over a fixed alphabet Σ.

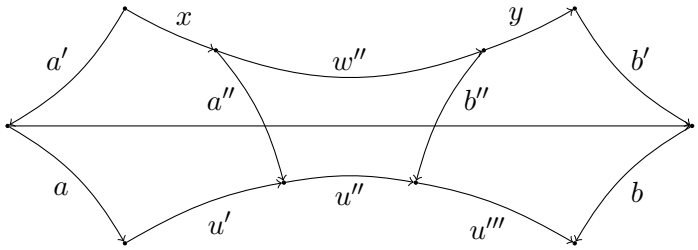

FIGURE 5.2. The evaluation of $B\langle a, b\rangle\langle a', b'\rangle$ agrees with the evaluation of $x \cdot B[i:j]\langle a'', b''\rangle \cdot y$. Here we are assuming that $\mathsf{eval}(B) = u = u'u''u'''$ and that $\mathsf{eval}(B[i:j]\langle a'', b''\rangle) = w''$.

Remark 5.1. In what follows we mostly need the prefix and suffix tether operators $\langle a, 1\rangle$ and $\langle 1, b\rangle$. For, suppose that $\Gamma(G, \Sigma)$ is hyperbolic and that u and v are geodesic words. Let $w =_G uv$ be a geodesic word representing their product. Then we can describe w (up to bounded Hausdorff distance) by taking a prefix of u, tethering the result at the end, concatenating with a short word, and then tethering (at the front) a suffix of v. See Figure 5.12 below.

We also note that iterating tether operators can be done "locally". That is, for any $a, a', b, b' \in \mathsf{B}(\zeta)$ there are elements $a'', b'' \in \mathsf{B}(\zeta)$, elements $x, y \in \mathsf{B}(2\zeta)$, and natural numbers i, j so that the expressions

$$B\langle a, b\rangle\langle a', b'\rangle \quad \text{and} \quad x \cdot B[i:j]\langle a'', b''\rangle \cdot y$$

have the same evaluation: that is, they represent the same shortlex reduced word. See Figure 5.2. Again, this "tether-elimination" is, in some sense, the heart of our proof of Lemma 5.3.

We say that a program is in *Chomsky normal form* if it is either a trivial program or all right hand sides $\rho(A)$ have one of the following forms, where $B, C \in V$, $a \in \Sigma$, $i \leq j$ and $b, c \in \mathsf{B}(\zeta)$: a, BC, $B[i:j]$, $B\langle b, c\rangle$. Similar to the case of straight-line programs, there is a linear-time algorithm that transforms a given program $\mathcal{G}$ (with $\mathsf{eval}(\mathcal{G}) \neq \varepsilon$) into a program $\mathcal{G}'$ in Chomsky normal form with the same output.

We say that a program $\mathcal{G}$ is *geodesic* (or *shortlex*) if for every variable A, the word $\mathsf{eval}(A)$ is geodesic (shortlex reduced).

Lemma 5.2. *There is a polynomial-time algorithm that, given a geodesic tether-cut program $\mathcal{G}$, returns a geodesic tether-cut program $\mathcal{G}'$ with the same evaluation which is in Chomsky normal form.*

Proof. We essentially use the usual algorithm; see, for example [49, Proposition 3.8]. However, some care must be taken with tethered variables. If $\rho(A) = B\langle a, b\rangle$ and $\rho(B) = \varepsilon$ then we set $\rho(A) = \mathsf{slex}(ab^{-1})$ and repeat the process. □

Note that the concatenation of geodesic words may not itself be geodesic; however the concatenation does provide two sides of a geodesic triangle. When the group G is hyperbolic, this gives us the beginnings of a reduction procedure.

We now turn to the task of proving Proposition 5.5. We will give a sequence of results that allows us to transform a geodesic tether-cut program into a straight-line program, whose evaluation is the shortlex representative of the original. The first step, in Lemma 5.3, gives such a transformation for tethered programs. The second step, finishing the proof of Proposition 5.5, is to transform a geodesic tether-cut program into an geodesic tethered program with the same output. This second step is inspired by Hagenah's result (Theorem 4.13) transforming a cut program into an equivalent straight-line program.

5.3. **Transforming tethered programs.** Suppose that $\mathcal{G} = (V, S, \rho)$ is a program, as above.

We recursively define the *height* of elements of $\Sigma \cup V$. If $a \in \Sigma$ then we take $\mathsf{height}(a) = 0$. For $A \in V$ we define

$$\mathsf{height}(A) = \max\{\mathsf{height}(B) + 1 \mid B \in \Sigma \cup V \text{ occurs in } \rho(A)\}$$

Finally we set $\mathsf{height}(\mathcal{G}) = \mathsf{height}(S)$.

Suppose that $\mathcal{G}$ is a tethered program in Chomsky normal form. If $A \in V$ is a variable we define its *tether-height*, denoted $\mathsf{height}_t(A)$, recursively as follows.

- If $\rho(A) = a$, then $\mathsf{height}_t(A) = 0$,
- if $\rho(A) = BC$, then $\mathsf{height}_t(A) = \max\{\mathsf{height}_t(B), \mathsf{height}_t(C)\}$, and
- if $\rho(A) = B\langle s, t\rangle$ then $\mathsf{height}_t(A) = \mathsf{height}_t(B) + 1$.

For a variable A we define its *tether-depth* to be

$$\mathsf{depth}_t(A) = \mathsf{height}_t(S) - \mathsf{height}_t(A) + 1$$

Lemma 5.3. *There is a polynomial-time algorithm that, given a geodesic tethered program $\mathcal{G}$, finds a shortlex straight-line program $\mathcal{G}'$ so that* $\mathsf{eval}(\mathcal{G}') = \mathsf{slex}(\mathsf{eval}(\mathcal{G}))$.

Proof. Set $\mathcal{G} = (V, S, \rho)$. The straight-line program $\mathcal{G}'$ that we construct will be of the form $\mathcal{G}' = (V', S', \rho')$ for suitable V' and ρ'.

Applying Lemma 5.2 we may assume that $\mathcal{G}$ is in Chomsky normal form. Introducing a new start variable, if needed, we may assume that $\rho(S)$ has the form $A\langle 1, 1\rangle$ for a variable A. We do this to force the evaluation of $\mathcal{G}$ to be shortlex reduced, not just geodesic. By removing unused variables, we can assume that S has maximal height and maximal tether-height among all variables. This implies that, for all $A \in V$, the tether-depth of A is greater than zero. Finally, for every variable $A \in V$ such that $\rho(A) = BC$ with $B, C \in V$ we can assume that

$$\mathsf{depth}_t(A) = \mathsf{depth}_t(B) = \mathsf{depth}_t(C)$$

To ensure this property we add dummy variables to $\mathcal{G}$, with productions of the form $X\langle 1,1\rangle$, as needed.

In the rest of the proof, height, height_t and depth_t always refer to the original tethered program $\mathcal{G}$.

We carry out the proof in a bottom-up fashion; that is, we consider the variables of $\mathcal{G}$ in order of increasing height. Here is an outline of the proof; we give the details below. Set $w = \mathsf{eval}(A)$. If $|w| \leq 16\zeta\mathsf{depth}_t(A) + 2\zeta$ then we compute and record w, as a word. If w is longer than that, then we instead compute words ℓ_A and r_A such that

$$w = \ell_A \cdot w' \cdot r_A$$

for some word w' of length at least 2ζ. The details of the computation depend on the production $\rho(A)$. We require that the word ℓ_A satisfies the following length constraint

$$8\zeta\mathsf{depth}_t(A) \leq |\ell_A| \leq 8\zeta\mathsf{depth}_t(A) + 2\zeta\mathsf{height}(A) \tag{4}$$

and similarly for r_A.

When w is long, we also add to the program $\mathcal{G}'$ the *decorated* variables $A'_{a,b}$ for all $a, b \in \mathsf{B}(\zeta)$. We arrange the following:

$$\mathsf{eval}(A'_{a,b}) = \mathsf{slex}(a \cdot w' \cdot b^{-1})$$

These new variables $A'_{a,b}$, and also a new start variable S', are the *only* variables appearing in $\mathcal{G}'$, that is, they form the set V'. All of the words that we compute and record along the way, such as the short words w and the prefixes and suffixes ℓ_A and r_A, are not separately stored as part of $\mathcal{G}'$.

That completes our outline of the proof. We now consider the possibilities for the right-hand side $\rho(A)$.

Case 1. Suppose $\rho(A) \in \Sigma$. Thus $w = \mathsf{eval}(A)$ is geodesic and shorter than $16\zeta\mathsf{depth}_t(A) + 2\zeta$. We record it and continue.

Case 2. Suppose $\rho(A) = BC$ for variables B and C. Recall that we have

$$\mathsf{depth}_t(A) = \mathsf{depth}_t(B) = \mathsf{depth}_t(C)$$

Set $\eta = \mathsf{depth}_t(A)$. Let $u = \mathsf{eval}(B)$, $v = \mathsf{eval}(C)$, and $w = \mathsf{eval}(A) = uv$. Recall that u, v, w are geodesic by assumption.

Case 2.1. Suppose $|u| > 16\zeta\eta + 2\zeta$ and $|v| > 16\zeta\eta + 2\zeta$. Hence, in previous stages of the algorithm we computed words ℓ_B, r_B, ℓ_C, r_C such that the following properties hold.

- The prefixes and suffixes ℓ_B, r_B, ℓ_C, r_C satisfy the length constraint of Equation 4.
- There are geodesic words u', v' of length at least 2ζ with $u = \ell_B \cdot u' \cdot r_B$ and $v = \ell_C \cdot v' \cdot r_C$.

Also, we have already defined variables $B'_{a,c}$ and $C'_{d,b}$ for all $a, b, c, d \in \mathsf{B}(\zeta)$, which produce $\mathsf{slex}(a \cdot u' \cdot c^{-1})$ and $\mathsf{slex}(d \cdot v' \cdot b^{-1})$, respectively. See Figure 5.5.

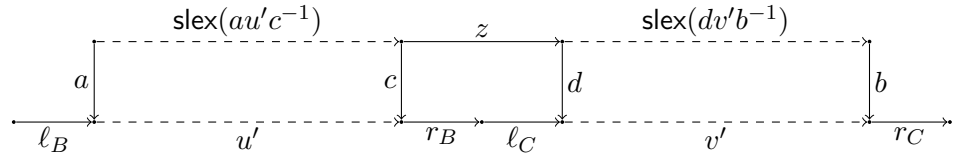

FIGURE 5.5. Case 2.1 from the proof of Lemma 5.3. Dashed lines represent words that are given by straight-line programs.

We now set $\ell_A = \ell_B$ and $r_A = r_C$. Since the tether-depths of A, B, C are all the same, but A has greater height, we deduce that ℓ_A and r_A satisfy the length constraint of Equation 4. We also note that

$$|u' \cdot r_B \cdot \ell_C \cdot v'| \geq 2\zeta$$

because $|u'| \geq 2\zeta$.

It remains to define the right-hand sides for the variables $A'_{a,b}$ for all $a, b \in \mathsf{B}(\zeta)$. Fix $a, b \in \mathsf{B}(\zeta)$. For all $c, d \in \mathsf{B}(\zeta)$ we compute

$$z = \mathsf{slex}(c \cdot r_B \cdot \ell_C \cdot d^{-1})$$

in polynomial time using Lemma 3.2. We then check, using Proposition 4.7 and Lemma 3.3, whether the word

$$\mathsf{eval}(B'_{a,c}) \cdot z \cdot \mathsf{eval}(C'_{d,b}) = \mathsf{slex}(a \cdot u' \cdot c^{-1}) \cdot \mathsf{slex}(c \cdot r_B \cdot \ell_C \cdot d^{-1}) \cdot \mathsf{slex}(d \cdot v' \cdot b^{-1})$$

is shortlex reduced, in which case it is equal to

$$\mathsf{slex}(a \cdot u' \cdot r_B \cdot \ell_C \cdot v' \cdot b^{-1})$$

Again, see Figure 5.5. Since $|u'| \geq 2\zeta \geq |a| + \zeta = |a| + 2\delta$, $|v'| \geq 2\zeta \geq |b| + \zeta = |b| + 2\delta$, and $|r_B \ell_C| \geq 16\zeta \geq 4\delta$, Corollary 3.5 ensures that there must be at least one such pair c, d. (If there are several, we stop as soon as we find the first such.) We then define

$$\rho'(A'_{a,b}) = B'_{a,c} \cdot z \cdot C'_{d,b}$$

Case 2.2. Suppose $|u| > 16\zeta\eta + 2\zeta$ and $|v| \leq 16\zeta\eta + 2\zeta$. Thus, at previous stages of the algorithm we computed the geodesic word v explicitly and also computed explicit words ℓ_B and r_B such that the following properties hold.

- The prefix and suffix ℓ_B, r_B satisfy the length constraint of Equation 4.
- There is a geodesic word u' of length at least 2ζ with $u = \ell_B \cdot u' \cdot r_B$.

Also, we already defined variables $B'_{a,b}$ for all $a, b \in \mathsf{B}(\zeta)$ such that $B'_{a,b}$ produces $\mathsf{slex}(a \cdot u' \cdot b^{-1})$.

If $|v| \leq 2\zeta$, then we set $\ell_A = \ell_B$ and $r_A = r_B v$. In this case, we also define $\rho'(A'_{a,b}) = B'_{a,b}$ for all $a, b \in \mathsf{B}(\zeta)$. Since $\mathsf{height}(B) + 1 \leq \mathsf{height}(A)$, we have the following.

$$8\zeta\eta \leq |\ell_A| \leq 8\zeta\eta + 2\zeta\mathsf{height}(B) \leq 8\zeta\eta + 2\zeta\mathsf{height}(A)$$
$$8\zeta\eta \leq |r_A| \leq 8\zeta\eta + 2\zeta(\mathsf{height}(B) + 1) \leq 8\zeta\eta + 2\zeta\mathsf{height}(A)$$

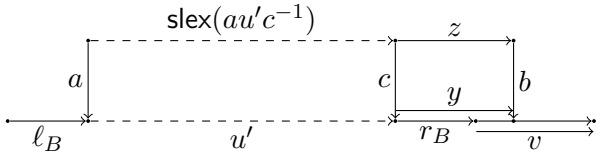

FIGURE 5.6. Case 2.2 from the proof of Lemma 5.3. Again, dashed lines represent words that are given by straight-line programs.

Thus the length bounds of Equation 4 are satisfied.

Now assume that $|v| > 2\zeta$. Again, we set $\ell_A = \ell_B$. Since $|r_B \cdot v| \geq |r_B| \geq 8\zeta\eta$ we can define r_A as the suffix of $r_B \cdot v$ of length $8\zeta\eta$; that is, $r_B \cdot v = y \cdot r_A$ for some word y of length $|y| = |r_B| + |v| - |r_A| \geq |v| > 2\zeta$. This satisfies the required bounds on the lengths of ℓ_A and r_A.

It remains to define the right-hand sides for the variables $A'_{a,b}$ for all $a, b \in \mathsf{B}(\zeta)$. Let us fix $a, b \in \mathsf{B}(\zeta)$. For all $c \in \mathsf{B}(\zeta)$ we compute $z = \mathsf{slex}(c \cdot y \cdot b^{-1})$ and check whether the word

$$\mathsf{eval}(B'_{a,c}) \cdot z = \mathsf{slex}(a \cdot u' \cdot c^{-1}) \cdot \mathsf{slex}(c \cdot y \cdot b^{-1})$$

is shortlex reduced. If it is, then it equals $\mathsf{slex}(a \cdot u' \cdot y \cdot b^{-1})$; see Figure 5.6. By Lemma 3.4, there must be at least one such c, for which we define

$$\rho'(A'_{a,b}) = B'_{a,c} \cdot z$$

Case 2.3. Suppose $|u| \leq 16\zeta\eta + 2\zeta$ and $|v| > 16\zeta\eta + 2\zeta$. This is dealt with in similar fashion to the previous case.

Case 2.4. Suppose $|u| \leq 16\zeta\eta + 2\zeta$ and $|v| \leq 16\zeta\eta + 2\zeta$. In this case, we have computed u and v explicitly at a previous stage. We now distinguish between the cases $|w| \leq 16\zeta\eta + 2\zeta$ and $|w| > 16\zeta\eta + 2\zeta$. In the first case, we record the word w for later use. In the second, we factorise w as $w = \ell_A \cdot w' \cdot r_A$ with $|\ell_A| = |r_A| = 8\zeta\eta$, and thus $|w'| \geq 2\zeta$. We then explicitly compute, for each $a, b \in \mathsf{B}(\zeta)$, the word $\mathsf{slex}(a \cdot w' \cdot b^{-1})$ and set $\rho'(A'_{a,b})$ equal to it. This again uses Lemma 3.2.

Case 3. Suppose $\rho(A) = B\langle a, b\rangle$ for $a, b \in \mathsf{B}(\zeta)$. Let $u = \mathsf{eval}(B)$ and $v = \mathsf{eval}(A) = \mathsf{slex}(a \cdot u \cdot b^{-1})$. The word u is geodesic by assumption, and v is shortlex reduced by definition. Let $\eta = \mathsf{depth}_t(B)$. We have $\mathsf{depth}_t(A) = \eta - 1 \geq 1$.

Case 3.1. Suppose $|u| \leq 16\zeta\eta + 2\zeta$. Hence, at a previous stage, we explicitly computed the word u. Using Lemma 3.2 we explicitly compute the word $v = \mathsf{slex}(a \cdot u \cdot b^{-1})$. The rest of the work divides into cases as $|v|$ is less than or equal to $16\zeta\eta + 2\zeta$ or is greater. This is analogous to Case 2.4 (where w plays the role of v).

Case 3.2. Suppose $|u| > 16\zeta\eta + 2\zeta$. At a previous stage we computed words ℓ_B, r_B with the following properties.

- The prefix and suffix ℓ_B, r_B satisfy the length constraint of Equation 4.
- There is a geodesic word u' of length at least 2ζ with $u = \ell_B \cdot u' \cdot r_B$.

Also, we already defined variables $B'_{c,d}$ for all $c, d \in \mathsf{B}(\zeta)$ such that $B'_{c,d}$ produces $z = \mathsf{slex}(cu'd^{-1})$.

We check for all $c, d \in \mathsf{B}(\zeta)$ whether

$$\begin{aligned} &\mathsf{slex}(a \cdot \ell_B \cdot c^{-1}) \cdot z \cdot \mathsf{slex}(d \cdot r_B \cdot b^{-1}) \\ = \; &\mathsf{slex}(a \cdot \ell_B \cdot c^{-1}) \cdot \mathsf{slex}(c \cdot u' \cdot d^{-1}) \cdot \mathsf{slex}(d \cdot r_B \cdot b^{-1}) \end{aligned}$$

is shortlex reduced. If it is shortlex reduced then it equals

$$\mathsf{slex}(a \cdot \ell_B \cdot u' \cdot r_B \cdot b^{-1}) = \mathsf{slex}(a \cdot u \cdot b^{-1}) = v$$

See Figure 5.7. By Corollary 3.5, there must exist such $c, d \in \mathsf{B}(\zeta)$.

Let $s = \mathsf{slex}(a \cdot \ell_B \cdot c^{-1})$ and $t = \mathsf{slex}(d \cdot r_B \cdot b^{-1})$. By the triangle inequality, these words have length at least $8\zeta\eta - 2\zeta$. Hence we can factorise these words as $s = wx$ and $t = yz$ with

$$|w| = |z| = 8\zeta(\eta - 1) = 8\zeta\mathsf{depth}_t(A) \geq 8\zeta$$

Again, see Figure 5.7. The words x and y have length at least 6ζ. We set $\ell_A = w$ and $r_A = z$. These words satisfy the required bounds on their lengths. Note that

$$\begin{aligned} &\mathsf{eval}(A) = \mathsf{slex}(a \cdot u \cdot b^{-1}) = \ell_A \cdot x \cdot \mathsf{slex}(c \cdot u' \cdot d^{-1}) \cdot y \cdot r_A \quad \text{and} \\ &|x \cdot \mathsf{slex}(c \cdot u' \cdot d^{-1}) \cdot y| \geq 12\zeta \geq 2\zeta \end{aligned}$$

It remains to define the right-hand sides of the variables $A'_{a',b'}$ for all $a', b' \in \mathsf{B}(\zeta)$. (This, in essence, is where we call upon Remark 5.1.)

Fix $a', b' \in \mathsf{B}(\zeta)$. The lower bounds on the lengths of w, x, y, z allow us to apply Lemma 3.4 to the geodesic quadrilaterals with sides labelled a, ℓ_B, c, wx and d, r_B, b, yz, respectively. Note that all of these words have been computed explicitly. Applying Lemma 3.2, we compute in polynomial time words $e, f \in \mathsf{B}(\zeta)$ and factorisations $\ell_B = w'x'$ and $r_B = y'z'$ such that $aw' =_G we$, $ex' =_G xc$, $dy' =_G yf$, and $fz' =_G zb$. Once again, see Figure 5.7. Now consider the geodesic quadrilateral with sides labelled $x' \cdot u' \cdot y'$, $\mathsf{slex}(a'e)$, $\mathsf{slex}(b'f)$, and $\mathsf{slex}(a'e \cdot x' \cdot u' \cdot y' \cdot (b'f)^{-1})$. The triangle inequality implies $|x'|, |y'| \geq 4\zeta$ and $|\mathsf{slex}(a'e)|, |\mathsf{slex}(b'f)| \leq 2\zeta$. Again applying Corollary 3.5 there are $c', d' \in \mathsf{B}(\zeta)$ such that the word

$$\begin{aligned} &\mathsf{slex}(a'e \cdot x' \cdot (c')^{-1}) \cdot \mathsf{eval}_{G'}(B'_{c',d'}) \cdot \mathsf{slex}(d' \cdot y' \cdot (b'f)^{-1}) \\ = \; &\mathsf{slex}(a'e \cdot x' \cdot (c')^{-1}) \cdot \mathsf{slex}(c' \cdot u' \cdot (d')^{-1}) \cdot \mathsf{slex}(d' \cdot y' \cdot (b'f)^{-1}) \end{aligned}$$

is shortlex reduced. Thus the above word is $\mathsf{slex}(a'e \cdot x' \cdot u' \cdot y' \cdot (b'f)^{-1})$. As before, we can compute such $c', d' \in \mathsf{B}(\zeta)$ in polynomial time. We finally define the right-hand side of $A'_{a',b'}$ as

$$\rho'(A'_{a',b'}) = \mathsf{slex}(a'e \cdot x' \cdot (c')^{-1}) \cdot B'_{c',d'} \cdot \mathsf{slex}(d' \cdot y' \cdot (b'f)^{-1})$$

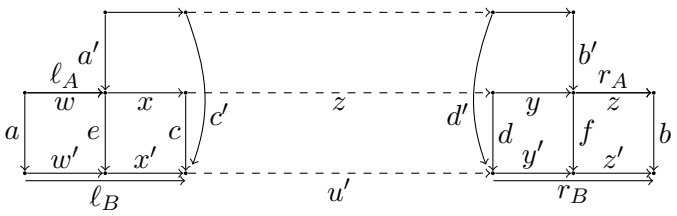

FIGURE 5.7. Case 3 from the proof of Lemma 5.3. Again, dashed lines represent words that are given by straight-line programs.

This concludes the definition of the right-hand sides for the variables $A'_{a,b}$.

We complete the definition of the straight-line program $\mathcal{G}'$. We add a new start variable S' to $\mathcal{G}'$. If $\mathsf{eval}(S)$ is short then we set $\rho'(S') = \mathsf{eval}(S)$ and we are done. If $\mathsf{eval}(S)$ is long then we set $\rho'(S') = \ell_S \cdot S'_{1,1} \cdot r_S$. This ensures $\mathsf{eval}(\mathcal{G}') = \ell_S \cdot \mathsf{slex}(s') \cdot r_S$, where s' is such that $\ell_S \cdot s' \cdot r_S = \mathsf{eval}(S) = \mathsf{eval}(\mathcal{G})$. But $\mathsf{eval}(\mathcal{G})$ is shortlex reduced (since $\rho(S)$ has the form $A\langle 1,1\rangle$). Hence s' is also shortlex reduced and we find

$$\mathsf{eval}(\mathcal{G}') = \ell_S \cdot \mathsf{slex}(s') \cdot r_S = \ell_S \cdot s' \cdot r_S = \mathsf{eval}(\mathcal{G})$$

This concludes the proof of the lemma. □

The next lemma follows from Lemma 5.3.

Lemma 5.4. *There is a polynomial-time algorithm that, given a geodesic tethered program $\mathcal{G}$, computes for every A the length $|\mathsf{eval}(A)|$.*

Proof. By Lemma 5.3, we can compute for every variable A a straight-line program $\mathcal{G}_A$ with $\mathsf{eval}(\mathcal{G}_A) = \mathsf{slex}(\mathsf{eval}(A))$. As in Section 4.2, we can compute $|\mathsf{eval}(\mathcal{G}_A)| = |\mathsf{slex}(\mathsf{eval}(A))| = |\mathsf{eval}(A)|$ in polynomial time. Here the last equality holds since $\mathcal{G}$ is a geodesic program. □

We now can prove our proposition; this generalises Lemma 5.3 to tether-cut programs.

Proposition 5.5. *There is a polynomial-time algorithm that, given a geodesic tether-cut program $\mathcal{G}$, computes a shortlex straight-line program $\mathcal{G}'$ such that $\mathsf{eval}(\mathcal{G}') = \mathsf{slex}(\mathsf{eval}(\mathcal{G}))$.*

Proof. The idea of the proof is taken from the proof of Theorem 4.13; see [31, Algorithmus 8.1.4]. That is, we will eliminate cut operators by pushing them towards smaller variables. We then appeal to Lemma 5.3 to eliminate tether operators.

Let $\mathcal{G} = (V, S, \rho)$ be the input geodesic tether-cut program. By Lemma 5.2, we can assume that $\mathcal{G}$ is in Chomsky normal form. Let $\mu = \mathsf{height}(\mathcal{G})$. By Lemma 5.3 it suffices to transform $\mathcal{G}$ into a geodesic tethered program for $\mathsf{eval}(\mathcal{G})$.

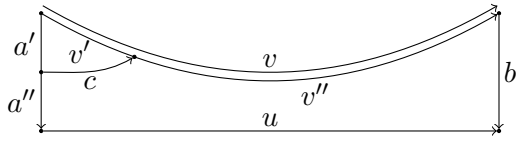

FIGURE 5.8. Case 3.1 in the proof of Proposition 5.5.

We will only consider cuts of the form $[: i]$ and $[i :]$; see Remark 4.14. It is not difficult to include also general cuts of the form $[i : j]$.

Consider a variable A such that $\rho(A) = B[: i]$; the case that $\rho(A) = B[i :]$ is dealt with analogously. We consider variables in order of *increasing* height; so the algorithm is bottom-up. By induction we may assume that no cut operator occurs in the right-hand side of any variable C with height less than that of A.

We now must eliminate the cut operator in $\rho(A)$. In so doing, we add at most μ new variables to the tether-cut program. Moreover the height of the tether-cut program after the cut elimination will still be bounded by μ. Hence, the final tethered program will have at most $\mu \cdot |V|$ variables. In addition, the bit length of every new right-hand side will be polynomially bounded in the input length. Thus, the size of the final tethered program will be polynomially bounded in the input length.

Recall that $\rho(A) = B[: i]$. We divide the work into cases, depending on the form of $\rho(B)$. Since we already have processed B, only one of the following cases can occur.

Case 1. Suppose $\rho(B) = a \in \Sigma$. If $i = 1$ we redefine $\rho(A) = a$, and if $i = 0$ we redefine $\rho(A) = \varepsilon$.

Case 2. Suppose $\rho(B) = CD$ with $C, D \in V$. We compute $n_C = |\mathsf{eval}(C)|$ using Lemma 5.4 and with an appeal to the induction hypothesis. If $i \leq n_C$ then we redefine $\rho(A) = C[: i]$. If $i > n_C$ then we add a new variable X, we define $\rho(X) = D[: i - n_C]$, and we redefine $\rho(A) = CX$. We then eliminate the new cut operator in $C[: i]$ or in $D[: i - n_C]$ with a top-down sub-routine. (This is what leads to the quadratic growth of new variables.)

Case 3. Suppose $\rho(B) = C\langle a, b\rangle$ with $C \in V$ and $a, b \in \mathsf{B}(\zeta)$. Let $u = \mathsf{eval}_{\mathcal{G}}(C)$, $v = \mathsf{eval}_{\mathcal{G}}(B)$, and $v = v'v''$ with $|v'| = i$. Thus, we have $\mathsf{eval}_{\mathcal{G}}(A) = v'$ and $v = \mathsf{slex}(aub^{-1})$. By Lemma 5.3 we can assume that we have straight-line programs for the words u and v.

Case 3.1. There exists $c \in \mathsf{B}(\zeta)$ and a factorisation $a = a'a''$ such that $v' =_G a'c$; see Figure 5.8. Note that this implies that $i = |v'| \leq 2\zeta$. Hence, we can check in polynomial time whether this case holds by computing the prefix of the compressed word v of length i. We redefine $\rho(A) = v'$.

Case 3.2. There exists $c \in \mathsf{B}(\zeta)$ and a factorisation $b = b''b'$ such that $v''b'' =_G c$; see Figure 5.9. As in Case 3.1 we can check in polynomial

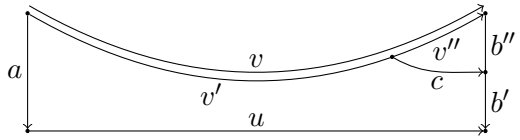

FIGURE 5.9. Case 3.2 in the proof of Proposition 5.5.

time whether this condition holds. We introduce a new variable X, we set $\rho(X) = C\langle 1, b'\rangle$, and we redefine $\rho(A) = X\langle a, c\rangle$.

Case 3.3. Neither Case 3.1 nor Case 3.2 holds. In this case, there exists a factorisation $u = u'u''$ and $c \in \mathsf{B}(\zeta)$ such that $v'c =_G au'$ and $v''b =_G cu''$; see Figure 3.2. The triangle inequality implies $i - 2\zeta \leq |u'| \leq i + 2\zeta$. We can find such a factorisation of u in polynomial time; we note that $j = |u'|$ lies in $\mathbb{N}$ and satisfies $|i - j| \leq 2\zeta$. So, using Theorem 4.13, we find straight-line programs for the $4\zeta + 1$ many words $u' = u[: j]$, where $j \in \mathbb{N}$, $|i - j| \leq 2\zeta$. Since u is geodesic, also all factors of u are geodesic. Hence, the straight-line programs for the words $u' = u[: j]$ must be geodesic too. Then we apply Lemma 5.3 and compute for every $c \in \mathsf{B}(\zeta)$ a shortlex straight-line program for the word $w' = \mathsf{slex}(au'c^{-1})$. Theorem 4.13 yields a shortlex straight-line program for $v' = v[: i]$. Finally, we check, using Theorem 4.9, whether $v' = w'$.

Hyperbolicity ensures that we will find at least one such j and c. We introduce a new variable X, we set $\rho(X) = C[: j]$, and we redefine $\rho(A) = X\langle a, c\rangle$ We then continue with the elimination of the cut operator in $C[: j]$, as in Case 2.

This concludes the proof of the lemma. □

Recall our convention: if $w \in \Sigma^*$ is a word then $g_w \in G$ is the corresponding group element. Thus g_w is a vertex of the Cayley graph $\Gamma = \Gamma(G, \Sigma)$.

Lemma 5.6. *There is a polynomial-time algorithm that, given geodesic tether-cut programs $\mathcal{G}$ and $\mathcal{H}$, determines if $d_\Gamma(g, h) \leq \delta$, where $g = g_{\mathsf{eval}(\mathcal{G})}$ and $h = g_{\mathsf{eval}(\mathcal{H})}$. Moreover, when this holds, the algorithm also finds an element $b \in \mathsf{B}(\delta)$ such that $g =_G hb$.*

Proof. Let S and T be the start variables of $\mathcal{G}$ and $\mathcal{H}$, respectively. For all $b \in \mathsf{B}(\delta)$ we produce a new geodesic tether-cut program $\mathcal{G}^b$ for $\mathsf{slex}(\mathsf{eval}(\mathcal{G})b^{-1})$. We do this by adding to $\mathcal{G}$ a new start variable with right-hand side $S\langle 1, b\rangle$.

We also add to $\mathcal{H}$ a new start variable with right-hand side $T\langle 1, 1\rangle$ and denote the resulting tether-cut program by $\mathcal{H}^1$. This ensures that the evaluation of $\mathcal{H}^1$ is $\mathsf{slex}(\mathsf{eval}(\mathcal{H}))$. Using Proposition 5.5 and Theorem 4.9 we now check, in polynomial time, if $\mathsf{eval}(\mathcal{G}^b) = \mathsf{eval}(\mathcal{H}^1)$. This is equivalent to $g =_G hb$. □

5.10. **Solving the compressed word problem.** We now prove our main result. Recall that Σ is a symmetric generating set for the hyperbolic group G.

Theorem 5.7. *There is a polynomial-time algorithm that, given a straight-line program $\mathcal{G}$ over Σ, finds a straight-line program $\mathcal{H}$ with evaluation* $\mathsf{slex}(\mathsf{eval}(\mathcal{G}))$.

Proof. By Proposition 5.5 it suffices to build, in polynomial time, a geodesic tether-cut program $\mathcal{H}$ for $\mathsf{slex}(\mathsf{eval}(\mathcal{G}))$. We process $\mathcal{G}$ from the bottom-up; that is, we consider its variables in order of increasing height. Set $\mathcal{G} = (V, S, \rho)$; applying [49, Proposition 3.8], we may assume that $\mathcal{G}$ is in Chomsky normal form. We build by induction on the height a new tether-cut program $\mathcal{G}' = (V', S', \rho')$ over Σ; here $V' = \{A' \mid A \in V\}$ is a copy of V and $S' \in V'$ is the variable corresponding to S. The construction will ensure that $\mathsf{eval}(A') = \mathsf{slex}(\mathsf{eval}(A))$ for every $A \in V$,

In the base cases we have $\rho(A) = a \in \Sigma$. Here we set $\rho'(A') = \mathsf{slex}(a)$.

In the inductive step we have $\rho(A) = BC$. Since B and C have smaller height than A they satisfy the induction hypotheses. Set

$$u = \mathsf{slex}(\mathsf{eval}(B)) = \mathsf{eval}(B') \quad \text{and} \quad v = \mathsf{slex}(\mathsf{eval}(C)) = \mathsf{eval}(C')$$

By Proposition 5.5 we can transform the geodesic tether-cut programs with start variables B' and C' into shortlex straight-line programs. Using these, we compute the lengths $m = |u|$ and $n = |v|$. If one or both of these have length zero then we accordingly take $\rho'(A') = C'$ or $\rho'(A') = B'$ or $\rho'(A') = \varepsilon$. We now assume that m and n are both non-zero. Breaking symmetry, we assume that $m \leq n$.

Let P be the path in the Cayley graph Γ starting at 1_G, ending at u, and labelled by u. Similarly, let Q be the path starting at u, ending at uv, and labelled by v. Finally, let R be the path starting at 1_G, ending at uv, and labelled by $\mathsf{slex}(uv)$. See Figure 5.12. The path $\bar{P}$, the reverse of P, is labelled by u^{-1}. Applying Remark 4.12 we invert the geodesic straight-line program for u to give a straight-line program for u^{-1}. Using Lemma 5.6 we can check whether or not

$$d_\Gamma(\bar{P}(m), Q(m)) \leq \delta$$

We break into cases accordingly.

Case 1. Suppose that $d_\Gamma(\bar{P}(m), Q(m)) \leq \delta$. We compute, again using Lemma 5.6, a word a of length at most δ such that $a =_G uv[: m]$. See the left-hand side of Figure 5.12. In this case we set $\rho'(A') = C'[m :]\langle a, 1\rangle$.

Case 2. Suppose that $d_\Gamma(\bar{P}(m), Q(m)) > \delta$. Using binary search, we compute an integer $k \in [0, m-1]$ such that

$$d_\Gamma(\bar{P}(k), Q(k)) \leq \delta \quad \text{and} \quad d_\Gamma(\bar{P}(k+1), Q(k+1)) > \delta$$

Here are the details of the binary search. We store an interval $[p, q] \subseteq [0, m]$ such that

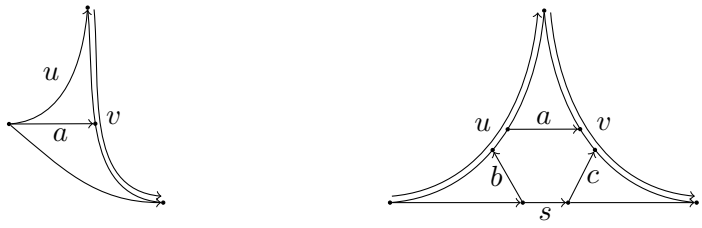

FIGURE 5.12. Case 1 (left) and 2 (right) from the proof of Theorem 5.7.

- $p < q$,
- $d_\Gamma(\bar{P}(p), Q(p)) \leq \delta$, and
- $d_\Gamma(\bar{P}(q), Q(q)) > \delta$.

We begin with $p = 0$ and $q = m$. We stop when $q = p + 1$. In each iteration, we compute $r = \lceil (p + q)/2 \rceil$ and check, using Lemma 5.6, whether

$$d_\Gamma(\bar{P}(r), Q(r)) \leq \delta \quad \text{or} \quad d_\Gamma(\bar{P}(r), Q(r)) > \delta$$

In the first case we set $p = r$ and do not change q; in the second case we set $q = r$ and do not change p. In each iteration the size of the interval $[p, q]$ is roughly halved. Thus the binary search halts after $O(\log(m))$ iterations; this is polynomial in the input size. In addition to the final position k, we record a word $a \in \mathsf{B}(\delta)$ that labels a path from $\bar{P}(k)$ to $Q(k)$. Let $j = k + 1$.

Recall that R is the path from $\bar{P}(m) = 1_G$ to $Q(n)$ labelled by $\mathsf{slex}(uv)$. By Lemma 3.6 there exist $i_P \leq i_Q$ such that

$$d_\Gamma(\bar{P}(j), R(i_P)) \leq \delta \quad \text{and} \quad d_\Gamma(Q(j), R(i_Q)) \leq \delta$$

For all pairs $b, c \in \mathsf{B}(\delta)$ we explicitly compute the word

$$s = \mathsf{slex}(b \cdot u[m - j] \cdot a \cdot v[k] \cdot c^{-1})$$

The symbols $u[m - j]$ and $v[k]$ can be computed in polynomial time from the available straight-line programs for u and v using [49, Proposition 3.9]; see Section 4.2. For each of these words s, we must check if the word

$$\mathsf{slex}(u[: m - j] \cdot b^{-1}) \cdot s \cdot \mathsf{slex}(c \cdot v[j :]) \tag{11}$$

is shortlex reduced; if so, it equals $\mathsf{slex}(uv)$. This step can be done using Lemma 3.3 and using the given geodesic tether-cut programs for u and v. From these we obtain geodesic tether-cut programs for $\mathsf{slex}(u[: m - j] \cdot b^{-1})$ and $\mathsf{slex}(c \cdot v[j :])$. We then use Proposition 5.5 to transform these into equivalent straight-line programs.

Lemma 3.6 ensures that that we will find a pair $b, c \in \mathsf{B}(\delta)$ such that the word in (11) is shortlex reduced. Using the first such pair we find, we set

$$\rho'(A') = (B'[: m - j]\langle 1, b\rangle) \cdot s \cdot (C'[j :]\langle c, 1\rangle)$$

This concludes the proof of the theorem. □

Theorem 5.7 now solves the compressed word problem.

Corollary 5.8. *The compressed word problem for a hyperbolic group can be solved in polynomial time.*

Proof. Suppose that $\mathcal{G}$ is the given compressed word. Note that $\mathsf{eval}(\mathcal{G}) \in \Sigma^*$ represents 1_G if and only if $\mathsf{slex}(\mathsf{eval}(\mathcal{G})) = \varepsilon$. This, in turn, happens if and only if $\mathsf{slex}(\mathsf{eval}(\mathcal{G}))$ has length zero. Since Theorem 5.7 gives a straight-line program $\mathcal{G}'$ for $\mathsf{slex}(\mathsf{eval}(\mathcal{G}))$, we are done. □

6. Further compressed decision problems

6.1. **Compressed order problem.** Suppose that G is a group. Suppose that Σ is a finite, symmetric generating set for G. For any $g \in G$ we define the *order* of g to be the smallest positive integer k so that $g^k = 1_G$. If there is no such k we define the order to be infinity. We define the *compressed order problem* as follows.

Input: Straight-line program $\mathcal{G}$ over Σ.
Output: The order of the group element $\mathsf{eval}(\mathcal{G})$.

As a consequence of Corollary 5.8 we have the following result.

Corollary 6.1. *Let G be a hyperbolic group. Then the compressed order problem for G can be solved in polynomial time.*

Proof. Let $\mathcal{G}$ be the given compressed word. Since G is hyperbolic, it has only finitely many conjugacy classes of finite subgroups [39, Theorem 6.8.4]. Thus there is a bound $c = c(G)$ on the size of its finite cyclic subgroups. Hence the order of any element $g \in G$ belongs to $\{1, \ldots, c\} \cup \{\infty\}$.

To compute the order of $\mathsf{eval}(\mathcal{G})$ it suffices to check whether $\mathsf{eval}(\mathcal{G})^k =_G 1_G$ for some integer k in the range $\{1, \ldots, c\}$. Proposition 4.6 gives us the desired compressed word and Corollary 5.8 checks it, both in polynomial time. □

6.2. **The compressed (simultaneous) conjugacy and compressed centraliser problems.** Suppose that G is a group. Suppose that Σ is a finite, symmetric generating set for G. For group elements $g, h \in G$ we have the standard abbreviation $g^h = h^{-1}gh$. If $\mathcal{L} = (g_1, \ldots, g_k)$ is a finite list of group elements then we write $\mathcal{L}^h = (g_1^h, \ldots, g_k^h)$. We extend these definitions to words over Σ in the obvious way.

6.3. **The problems.** The *compressed conjugacy problem* for G is the following.

Input: Straight-line programs $\mathcal{G}$ and $\mathcal{H}$ over Σ.
Question: Do $\mathsf{eval}(\mathcal{G})$ and $\mathsf{eval}(\mathcal{H})$ represent conjugate elements in G?

If $\mathcal{L}$ is a list of straight-line programs over Σ, then we define $\mathsf{eval}(\mathcal{L})$ to be the corresponding list of evaluations. We now define the *compressed simultaneous conjugacy problem* for G.

Input: Finite lists $\mathcal{L} = (\mathcal{G}_1, \ldots, \mathcal{G}_k)$ and $\mathcal{M} = (\mathcal{H}_1, \ldots, \mathcal{H}_k)$ of straight-line programs over Σ.
Question: Are $\mathsf{eval}(\mathcal{L})$ and $\mathsf{eval}(\mathcal{M})$ conjugate lists in G?

In the case when the answer to either of these questions is positive, we might also want to compute a straight-line program for an element that conjugates $\mathsf{eval}(\mathcal{G})$ to $\mathsf{eval}(\mathcal{H})$ or $\mathsf{eval}(\mathcal{L})$ to $\mathsf{eval}(\mathcal{M})$.

The *compressed centraliser problem* for G is the following computation problem.

Input: A finite list $\mathcal{L} = (\mathcal{G}_1, \ldots, \mathcal{G}_k)$ of straight-line programs over Σ.
Output: A finite list $\mathcal{M} = (\mathcal{H}_1, \ldots, \mathcal{H}_l)$ such $\mathsf{eval}(\mathcal{M})$ generates the intersection of the centralisers of the elements $\mathsf{eval}(\mathcal{L})$.

Note that this intersection is in fact the centraliser of the subgroup generated by the elements $\mathsf{eval}(\mathcal{L})$. When the desired centraliser is not finitely generated, by convention the problem has no solution.

6.4. **The proofs.** A linear-time algorithm for solving the conjugacy problem in a hyperbolic group G is described in [23, Section 3]. This was generalised in [10] to linear-time algorithms for the (uncompressed) simultaneous conjugacy, and the centraliser, problems. We will show that essentially the same algorithms can be used to solve the compressed (simultaneous) conjugacy problem and the compressed centraliser problem, in polynomial time.

We deal with the compressed conjugacy problem in Section 6.4.1. Building on that, and making the special assumption that one of the input elements has infinite order, we solve the compressed simultaneous conjugacy problem and the compressed centraliser problem in Section 6.4.2. Finally, we deal with the case that all input group elements have finite order in Section 6.4.3.

6.4.1. *Compressed conjugacy problem.* We now have the following.

Theorem 6.2. *Let G be a hyperbolic group. Then the compressed conjugacy problem in G is polynomial time.*

Proof. The input consists of two straight-line programs $\mathcal{G}$ and $\mathcal{H}$; we wish to test if $u = \mathsf{eval}(\mathcal{G})$ and $v = \mathsf{eval}(\mathcal{H})$ are conjugate. To do this we essentially use the conjugacy algorithm from [23, Theorem 1.1], applied to the words u and v. We will describe our modification of their algorithm, step-by-step, in the following.

Our description of each step consists of two parts. The first describes operations relating to the words u and v; the second explains how we effect these operations in polynomial time using only the straight-line programs $\mathcal{G}$ and $\mathcal{H}$. All assertions that we make in the uncompressed setting are justified in [23]. All corresponding assertions are then justified again, in the compressed setting, using the work in previous sections of this paper.

Let δ be a positive integer that serves as a thinness constant for the Cayley graph $\Gamma = \Gamma(G, \Sigma)$; see Section 3. We define constants $L = 34\delta + 2$ and $K = 17(2L + 1)/7$; see [23, pages 298].

A word $w \in \Sigma^*$ is said to be *shortlex straight* if, for all non-negative powers k, the word w^k is shortlex reduced. Applying Lemma 3.3 and Proposition 4.8 we can determine, in polynomial time, if a given compressed word $\mathsf{eval}(\mathcal{G})$ is shortlex straight.

In the preprocessing stage, we make a look-up table of all pairs of shortlex reduced words of length at most K that are conjugate in G.

Step 1. We replace u and v by $\mathsf{slex}(u)$ and $\mathsf{slex}(v)$.

By Theorem 5.7 we can replace, in polynomial time, the programs $\mathcal{G}$ and $\mathcal{H}$ by straight-line programs for $\mathsf{slex}(\mathsf{eval}(\mathcal{G}))$ and $\mathsf{slex}(\mathsf{eval}(\mathcal{H}))$, respectively.

Step 2. For a word w, we define $w_C = w_R w_L$, where $w = w_L w_R$ with $|w_L| \leq |w_R| \leq |w_L| + 1$. Replace u by $\mathsf{slex}(u_C)$ and v by $\mathsf{slex}(v_C)$.

Using cut operators and Theorem 5.7 we can make the corresponding substitutions on $\mathcal{G}$ and $\mathcal{H}$.

Step 3. If $|u|, |v| \leq K$ then use the look-up table to test for conjugacy of u and v. Otherwise, at least one of the words, u say, satisfies $|u| \geq K > 2L+1$. If $|v| < 2L + 1$ then u and v are not conjugate [23, Section 3.1], and we return false. We assume from now on that $|u|, |v| \geq 2L + 1$.

For the compressed conjugacy problem, if $|\mathsf{eval}(\mathcal{G})|, |\mathsf{eval}(\mathcal{H})| \leq K$ then we can compute $\mathsf{eval}(\mathcal{G})$ and $\mathsf{eval}(\mathcal{H})$ explicitly. We then proceed as in the uncompressed setting.

Step 4. There exists a group element $g \in \mathsf{B}(4\delta)$ and a positive integer m, of size at most $|\mathsf{B}(4\delta)|^2$, such that the shortlex reduction of $g^{-1}u^m g$ is shortlex straight [23, Section 3.2]. To find such, for every pair (g, m) of at most those sizes, we replace u by $\mathsf{slex}(g^{-1}ug)$ and test $z = \mathsf{slex}(u^m)$ to see if it is shortlex straight.

Using Proposition 4.8, we can perform the corresponding operations with $\mathcal{G}$. Thus we find g and m and also find a straight-line program $\mathcal{G}'$ with $\mathsf{eval}(\mathcal{G}') = z$.

Step 5. We now test for the following necessary (but not sufficient) property for the conjugacy of u and v: is v^m conjugate to z? We decide this as follows. For all $h \in \mathsf{B}(6\delta)$, we compute $v_h = \mathsf{slex}(hv^m h^{-1})$, and then test whether v_h is a rotation of z. If this fails for all h, then v^m and $z =_G u^m$ are not conjugate [23, Section 3.3]. But then, u and v are not conjugate, so we may stop and return false.

Otherwise, we find h and v_h with this property. Let z_1 be a prefix of z such that $z =_G z_1 h v^m h^{-1} z_1^{-1}$. We replace v by $\mathsf{slex}(z_1 h v h^{-1} z_1^{-1})$. Now we have $v^m =_G u^m =_G z$. From this we get that every $g \in G$ with $g^{-1}ug =_G v$ belongs to the centraliser $C_G(z)$ of z in G. In particular, u and v are conjugate in G if and only if they are conjugate in $C_G(z)$.

Using Corollary 4.11 we can do the corresponding calculations with $\mathcal{H}$ and $\mathcal{G}'$. Checking whether v_h is a rotation of z can be accomplished in polynomial time by the first statement of Corollary 4.11; the second statement allows us to compute in polynomial time a straight-line program for z_1.

Step 6. Find the shortest prefix y of z that is a *root* of z: that is, there is an $\ell \geq 1$ so that $z = y^\ell$. We do that by finding the second occurrence of the substring z in the word z^2.

To find the root of $\mathsf{eval}(\mathcal{G}')$, we compute a straight-line program for $\mathsf{eval}(\mathcal{G}')^2$ and appeal to Theorem 4.10. We then build a straight-line program $\mathcal{G}''$ with $\mathsf{eval}(\mathcal{G}'') = y$ using cut operators and Theorem 4.13.

Step 7. For each $h \in \mathsf{B}(2\delta)$, compute $\mathsf{slex}(hzh^{-1})$ and test whether it is a rotation of z. If so, find a prefix z_h of z with $hzh^{-1} =_G z_h^{-1} z z_h$, and compute and store $\mathsf{slex}(z_h \cdot h)$ (which lies in $C_G(z)$) in a list C_z. Then $|C_z| \leq J = |\mathsf{B}(2\delta)|$.

Corollary 4.11 allows us to do the corresponding calculations with $\mathcal{G}'$. We obtain a list of straight-line programs that evaluate to the words in the list C_z.

Step 8. For each n with $0 \leq n \leq (J-1)!$ and for each $z' \in C_z$, let $g = y^n z'$. Test if $u =_G gvg^{-1}$. If so, then return true (and a conjugating element). If not, then return false because u and v are not conjugate [23, Section 3.4].

We can perform corresponding operations on the straight-line programs.

This concludes our description of a polynomial-time algorithm for the compressed conjugacy problem. The correctness proof is identical to that in [23, Section 3] □

6.4.2. *Compressed simultaneous conjugacy and centralisers: the infinite order case.* We now turn to the following.

Theorem 6.3. *Let G be a hyperbolic group. Then the compressed simultaneous conjugacy problem for G can be solved in polynomial time. Moreover, if the two input lists are conjugate, then we can compute a straight-line program for a conjugating element in polynomial time.*

Theorem 6.4. *Let G be a hyperbolic group. Then the compressed centraliser problem for G can be solved in polynomial time.*

The input now consists of two lists $\mathcal{L} = (\mathcal{G}_1, \ldots, \mathcal{G}_k)$ and $\mathcal{M} = (\mathcal{H}_1, \ldots, \mathcal{H}_k)$ of straight-line programs over the alphabet Σ. For the compressed centraliser problem we assume that $\mathcal{L} = \mathcal{M}$. For all i we let $u_i = \mathsf{eval}(\mathcal{G}_i)$ and $v_i = \mathsf{eval}(\mathcal{H}_i)$.

By Corollary 6.1 we can check in polynomial time whether some u_i has infinite order. Following [10, Section 3] we begin by assuming that this is indeed the case. Reordering the lists in the same way, as needed, we may assume that u_1 has infinite order. If v_1 does not have infinite order we are done.

The conjugacy testing algorithm proceeds as follows. We first repeatedly replace the elements in $\mathcal{L}$ by conjugates, using a common conjugating element. This culminates in a check for a conjugating element which must lie in an explicit finite set. In each replacement, straight-line programs are known for the conjugating element. By keeping track of these, we can find (if the lists are conjugate) an overall conjugating element for the original input. We omit further details regarding this overall conjugating element.

We proceed by carrying out the eight steps of the algorithm of Section 6.4.1 as applied to u_1 and v_1. If they are not conjugate we are done. Suppose

that they are conjugate. In this case we record the programs for the words z and y produced by Steps 5 and 6. We also record the list (of straight-line programs) C_z given in Step 7. The overall algorithm also gives us a straight-line program for an element $g \in G$ with $u_1^g =_G v_1$. The algorithm also replaced u_1 and v_1 by conjugates in some of the steps; we make the corresponding replacements to the other elements of $\mathcal{L}$ and $\mathcal{M}$. By replacing each u_i by its conjugate under g, we may now assume that $u_1 = v_1$.

Thus we have reduced the problem to the following. Assuming that $u_1 = v_1$ and that u_1 has infinite order, we must decide if there is $g \in C_G(u_1)$ with $u_i^g =_G v_i$ for $2 \leq i \leq k$. We are also given z; thus $u_1^m =_G v_1^m =_G z$ and z is shortlex straight element z and $m \geq 1$. We are also given y with $z = y^\ell$ and for maximal $\ell \geq 1$.

In [23, Section 3.4] it is shown that all elements $g \in C_G(z)$ have the form $g =_G y^n z'$, for some $n \in \mathbb{Z}$ and $z' \in C_z$, where C_z is the list given above. So the same applies to any $g \in C_G(u_1) \subseteq C_G(z)$.

We now try each $z' \in C_z$ in turn. Replacing each v_i by $z' v_i (z')^{-1}$, the problem reduces to the following: is there some $n \in \mathbb{Z}$ such that $u_i^{y^n} =_G v_i$ for $1 \leq i \leq k$?

To solve this problem, we apply [10, Proposition 24] to each pair u_i, v_i in turn. For each i, there are three possibilities.

(i) there exist $0 \leq r_i < t_i \leq |\mathsf{B}(2\delta)|$ such that $u_i^{y^j} =_G v_i$ if and only if $j \equiv r_i \bmod t_i$;
(ii) there is a unique $r_i \in \mathbb{Z}$ with $u_i^{y^{r_i}} =_G v_i$, where $|r_i|$ is bounded by a linear function of $|u_i|$ and $|v_i|$; or
(iii) there is no $r_i \in \mathbb{Z}$ with $u_i^{y^{r_i}} =_G v_i$.

The proof of [10, Proposition 24] provides an algorithm for determining which case applies, and for finding r_i, t_i in cases (i) and (ii). This involves calculating a number of powers u_i^n, v_i^n, and y^n for integers n such that $|n|$ is bounded by a linear function of $|u_i|$ and $|v_i|$, and where the number of powers that need to be calculated is bounded by a constant. So we can perform these calculations in polynomial time with straight-line programs by Proposition 4.6.

After performing this calculation for each i with $1 \leq i \leq k$, the conjugacy problem for the lists reduces to solving some modular linear equations involving the integers r_i and t_i, as described in [10, Section 3.4]. Since the r_i and t_i in case (i) are bounded by a constant and, for r_i in case (ii), $\log |r_i|$ is bounded by a linear function of the size of the straight-line programs representing u_i and v_i, these equations can be solved in polynomial time using standard arithmetical operations on the binary representations of r_i and t_i. This completes our discussion of the compressed simultaneous conjugacy problem in the case where there is a list element of infinite order.

For the compressed centraliser problem, we are in the same situation but with $v_i = u_i$ for all i. We perform the same calculations as above, but we do them for every $z' \in C_z$. If there are solutions, then we find them by solving

modular equations. The set of solutions we find now generates the centraliser. This completes our discussion of the compressed centraliser problem in the case where there is a list element of infinite order.

6.4.3. *Compressed simultaneous conjugacy and centralisers: the finite order case.* Here we continue the proofs of Theorems 6.3 and 6.4. We now consider the case where all of the u_i (in the list $\mathsf{eval}(\mathcal{L})$) have finite order. We now follow [10, Section 4]. No new complications arise when applying those methods to lists of straight-line programs. Indeed, some steps become easier because we are only interested in achieving polynomial, rather than linear, time.

We follow the steps of the algorithm described in [10, Section 4.5]. We deal with the conjugacy and centraliser problems together; the two lists are taken to be equal for the centraliser calculation. At this stage we have already verified that all of the u_i and v_i have finite order. Furthermore all of the words are shortlex reduced. By deleting programs, we can assume that the list $u_1, \ldots, u_k$, and likewise the list $v_1, \ldots, v_k$, has no duplicates. Thus the u_i represent distinct group elements, as do the v_i.

Let $n = \min\{|\mathsf{B}(2\delta)|^4 + 1, k\}$. We consider the prefix sublists $\mathsf{eval}(\mathcal{L}') = (u_1, \ldots, u_n)$ and $\mathsf{eval}(\mathcal{M}') = (v_1, \ldots, v_n)$. We apply the function ShortenWords from [10, Section 4.2] to the lists $\mathcal{L}'$ and $\mathcal{M}'$. This function applies slex to a number of words; this number is bounded above by n^2. Each word is a concatenation (of length at most $n+2$) of words either from the lists $\mathcal{L}'$ or $\mathcal{M}'$, or of words previously calculated during this process. These operations can be executed in polynomial time when working with straight-line programs. Since there is an absolute bound $|\mathsf{B}(2\delta)|^4 + 1$ on the lengths of $\mathcal{L}'$ and $\mathcal{M}'$, the complete application of ShortenWords to each list takes place in polynomial time.

ShortenWords has two possible outcomes. In the first it finds a product $u_r \cdot u_{r+1} \cdots u_s$ of elements of $\mathcal{L}'$ with infinite order. This reduces the problem to the case dealt with in Section 6.4.2.

In the second possible outcome, ShortenWords replaces $\mathcal{L}'$ and $\mathcal{M}'$ by conjugates and then calculates lists $\mathcal{L}'' = (u'_1, \ldots, u'_n)$ and $\mathcal{M}'' = (v'_1, \ldots, v'_n)$ with $|u'_i|$ and $|v'_i|$ bounded by a constant, and such that $\mathcal{L}^g = \mathcal{M}$ if and only if $(\mathcal{L}'')^g = \mathcal{M}''$.

We now test in time $O(1)$ (using our precomputed look-up table) whether there exists $g \in G$ with

$$(u'_1, \ldots, u'_n)^g = (v'_1, \ldots, v'_n)$$

If so, we replace $(u_1, \ldots, u_k)$ by $(u_1, \ldots, u_k)^g$ and thereby assume that $u_i = v_i$ for $1 \leq i \leq n$. For the centraliser problem, methods are described in [27, Proposition 2.3] of finding a generating set of the centraliser of any quasiconvex subgroup of any biautomatic group; finitely generated subgroups of hyperbolic groups satisfy these conditions. Since they need only be applied

to words of bounded length their complexity does not matter - indeed, we could precompute all such centralisers.

This completes the proof in the case $n = k$. In the case $k > n$, it is proved in [10, Corollary 30] that the centraliser C of the subgroup $\langle u_1, \ldots, u_n \rangle$ is finite, and that the elements of C have lengths bounded by a constant. So we can compute the elements of C explicitly (in time $O(1)$). Then we simply need to check whether any $g \in C$ satisfies

$$(u_{n+1}, \ldots, u_k)^g = (v_{n+1}, \ldots, v_k)$$

This completes the proofs of Theorems 6.3 and 6.4. □

6.5. **Compressed knapsack.** In this final section, we prove the following.

Theorem 6.5. *If G is an infinite hyperbolic group then the compressed knapsack problem for G is* NP*–complete.*

As above, fix G a finitely generated group. Fix as well a finite symmetric generating set Σ. A *knapsack expression* over Σ is a regular expression of the form $E = u^{-1}u_1^*u_2^* \cdots u_k^*$ with $k \geq 0$ and $u, u_i \in \Sigma^*$. The *length* of E is defined to be $|E| = |u| + \sum_{i=1}^{k} |u_i|$. A *solution* for E is a tuple $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$ of natural numbers such that $u =_G u_1^{n_1}u_2^{n_2} \cdots u_k^{n_k}$. In other words: the language defined by E contains a word that represents the identity of G.

The *knapsack problem* for G, over Σ, is the following.

Input: A knapsack expression E over Σ.
Question: Does E has a solution?

In [57, Theorem 6.1] it was shown that the knapsack problem for a hyperbolic group can be solved in polynomial time. A crucial step in the proof for this fact is the following result, which is of independent interest.

Theorem 6.6 [57, Theorem 6.7]**.** *For every hyperbolic group G there exists a polynomial $p(x)$ such that the following holds. Suppose that a knapsack expression $E = u^{-1}u_1^*u_2^* \cdots u_k^*$ over G has a solution. Then E has a solution $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$ such that $n_i \leq p(|E|)$ for all i satisfying $1 \leq i \leq k$.* □

Recently, this result has been extended to acylindrically hyperbolic group in [8].

Let us now consider the *compressed knapsack problem* for G. It is defined in the same way as the knapsack problem, except that the words $u, u_i \in \Sigma^*$ are given by straight-line programs. Note that the compressed knapsack problem for $\mathbb{Z}$ is NP–complete [30, Proposition 4.1.1]. Hence, for every group with an element of infinite order, the compressed knapsack problem is NP–hard. This makes it interesting to look for groups where the compressed knapsack problem is NP–complete.

From Corollary 5.8 and Theorem 6.6 we prove Theorem 6.5, which states that compressed knapsack for an infinite hyperbolic group G is NP–complete.

Proof of Theorem 6.5. Consider a knapsack expression $E = u^{-1}u_1^*u_2^* \cdots u_k^*$ over G, where u and the u_i are given by straight-line programs $\mathcal{G}$ and $\mathcal{G}_i$. We then have $|u|, |u_i| \leq 3^{|\mathcal{G}_i|/3}$ by Lemma 4.5. Let $N = |\mathcal{G}| + \sum_{i=1}^k |\mathcal{G}_i|$ be the input length.

By Theorem 6.6, there exists a polynomial $p(x)$ such that E has a solution if and only if it has a solution $(n_1, n_2 \ldots, n_k) \in \mathbb{N}^k$ with $n_i \leq p(|E|)$ for all i so that $1 \leq i \leq k$. Thus we obtain a bound of the form $2^{O(N)}$ on the exponents n_i. Hence, we can guess the binary encoding of a tuple $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$ with all n_i bounded by $2^{O(N)}$ and then check whether it is a solution for E. The latter can be done in polynomial time by constructing from the straight-line programs $\mathcal{G}$ and $\mathcal{G}_i$ a straight-line program $\mathcal{H}$ for $u^{-1}u_1^{n_1}u_2^{n_2} \cdots u_k^{n_k}$ using Proposition 4.6. Finally, we check in polynomial time whether $\mathsf{eval}(\mathcal{H}) =_G 1$ using Corollary 5.8.

The NP–hardness of the compressed knapsack problem for G (an infinite hyperbolic group) now follows from the fact that G has elements of infinite order [28, page 156] and the above mentioned result for $\mathbb{Z}$ [30, Proposition 4.1.1]. □

References

[1] Ian Agol. The virtual Haken conjecture. *Doc. Math.*, 18:1045–1087, 2013. With an appendix by Ian Agol, Daniel Groves, and Jason Manning. arXiv:1204.2810, doi:10.1016/j.procs.2013.05.269. [5]

[2] J. M. Alonso, T. Brady, D. Cooper, V. Ferlini, M. Lustig, M. Mihalik, M. Shapiro, and H. Short. Notes on word hyperbolic groups. In *Group theory from a geometrical viewpoint (Trieste, 1990)*, pages 3–63. World Sci. Publ., River Edge, NJ, 1991. Edited by Short. doi:10.1142/1235. [8, 9, 10, 11]

[3] László Babai and Endre Szemerédi. On the complexity of matrix group problems I. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science, FOCS 1984*, pages 229–240. IEEE Computer Society, 1984. doi:10.1109/SFCS.1984.715919. [4]

[4] Laurent Bartholdi, Michael Figelius, Markus Lohrey, and Armin Weiß. Groups with ALOGTIME-hard word problems and PSPACE-complete circuit value problems. In *35th Computational Complexity Conference*, volume 169 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. 29, 29. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2020. arXiv:1909.13781, doi:10.4230/LIPIcs.CCC.2020.29. [5]

[5] Gilbert Baumslag. A non-cyclic one-relator group all of whose finite quotients are cyclic. *J. Austral. Math. Soc.*, 10:497–498, 1969. doi:10.1017/S1446788700007783. [6]

[6] Gilbert Baumslag and Donald Solitar. Some two-generator one-relator non-Hopfian groups. *Bull. Amer. Math. Soc.*, 68:199–201, 1962. doi:10.1090/S0002-9904-1962-10745-9. [1]

[7] Martin Beaudry, Pierre McKenzie, Pierre Péladeau, and Denis Thérien. Finite monoids: from word to circuit evaluation. *SIAM J. Comput.*, 26(1):138–152, 1997. doi:10.1137/S0097539793249530. [4]

[8] Agnieszka Bier and Oleg Bogopolski. Exponential equations in acylindrically hyperbolic groups, 2021. arXiv:2106.11385. [34]

[9] Brian H. Bowditch. A short proof that a subquadratic isoperimetric inequality implies a linear one. *Michigan Math. J.*, 42(1):103–107, 1995. doi:10.1307/mmj/1029005156. [1]

[10] David J. Buckley and Derek F. Holt. The conjugacy problem in hyperbolic groups for finite lists of group elements. *Internat. J. Algebra Comput.*, 23(5):1127–1150, 2013. arXiv:1111.1554, doi:10.1142/S0218196713500203. [3, 10, 29, 31, 32, 33, 34]

[11] Bruce Chandler and Wilhelm Magnus. *The history of combinatorial group theory*, volume 9 of *Studies in the History of Mathematics and Physical Sciences*. Springer-Verlag, New York, 1982. A case study in the history of ideas. doi:10.1007/978-1-4613-9487-7. [1]

[12] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and abhi shelat. The smallest grammar problem. *IEEE Trans. Inform. Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116. [13]

[13] Laura Ciobanu and Murray Elder. Solutions sets to systems of equations in hyperbolic groups are EDT0L in PSPACE. In *46th International Colloquium on Automata, Languages, and Programming*, volume 132 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 110, 15. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2019. arXiv:1902.07349, doi:10.4230/LIPIcs.ICALP.2019.150. [10]

[14] François Dahmani and Vincent Guirardel. Foliations for solving equations in groups: free, virtually free, and hyperbolic groups. *J. Topol.*, 3(2):343–404, 2010. arXiv:1902.07349, doi:10.1112/jtopol/jtq010. [10]

[15] François Dahmani and Vincent Guirardel. The isomorphism problem for all hyperbolic groups. *Geom. Funct. Anal.*, 21(2):223–300, 2011. arXiv:1002.2590, doi:10.1007/s00039-011-0120-0. [3, 10]

[16] Max Dehn. Über unendliche diskontinuierliche Gruppen. *Math. Ann.*, 71(1):116–144, 1911. doi:10.1007/BF01456932. [1]

[17] Volker Diekert, Olga Kharlampovich, and Atefeh Mohajeri Moghaddam. SLP compression for solutions of equations with constraints in free and hyperbolic groups. *Internat. J. Algebra Comput.*, 25(1-2):81–111, 2015. arXiv:1308.5586, doi:10.1142/S0218196715400056. [10]

[18] Volker Diekert, Jürn Laun, and Alexander Ushakov. Efficient algorithms for highly compressed data: the word problem in Higman's group is in P. *Internat. J. Algebra Comput.*, 22(8):1240008, 19, 2012. arXiv:1103.1232, doi:10.1142/S0218196712400085. [6]

[19] Will Dison, Eduard Einstein, and Timothy R. Riley. Ackermannian integer compression and the word problem for hydra groups. In *41st International Symposium on Mathematical Foundations of Computer Science*, volume 58 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 30, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2016. doi:10.4230/LIPIcs.MFCS.2016.30. [6]

[20] Will Dison, Eduard Einstein, and Timothy R. Riley. Taming the hydra: the word problem and extreme integer compression. *Internat. J. Algebra Comput.*, 28(7):1299–1381, 2018. doi:10.1142/S0218196718500583. [6]

[21] Will Dison and Timothy R. Riley. Hydra groups. *Comment. Math. Helv.*, 88(3):507–540, 2013. doi:10.4171/CMH/294. [6]

[22] David B. A. Epstein, James W. Cannon, Derek F. Holt, Silvio V. F. Levy, Michael S. Paterson, and William P. Thurston. *Word processing in groups*. Jones and Bartlett Publishers, Boston, MA, 1992. doi:10.1201/9781439865699. [10, 11]

[23] David B. A. Epstein and Derek F. Holt. The linearity of the conjugacy problem in word-hyperbolic groups. *Internat. J. Algebra Comput.*, 16(2):287–305, 2006. doi:10.1142/S0218196706002986. [3, 9, 10, 29, 30, 31, 32]

[24] Elizaveta Frenkel, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in products of groups. *J. Symbolic Comput.*, 74:96–108, 2016. doi:10.1016/j.jsc.2015.05.006. [6]

[25] Moses Ganardi, Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack problems for wreath products. In *35th Symposium on Theoretical Aspects of Computer Science*,

volume 96 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 32, 13. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018. doi:10.4230/LIPIcs.STACS.2018.32. [6]

[26] Max Garzon and Yechezkel Zalcstein. The complexity of Grigorchuk groups with application to cryptography. *Theoretical Computer Science*, 88(1):83–98, 1991. doi:10.1016/0304-3975(91)90074-C. [5]

[27] Stephen M. Gersten and Hamish B. Short. Rational subgroups of biautomatic groups. *Ann. of Math. (2)*, 134(1):125–158, 1991. doi:10.2307/2944334. [33]

[28] Étienne Ghys and Pierre de la Harpe. Panorama. In *Sur les groupes hyperboliques d'après Mikhael Gromov (Bern, 1988)*, volume 83 of *Progr. Math.*, pages 1–25. Birkhäuser Boston, Boston, MA, 1990. doi:10.1007/978-1-4684-9167-8_1. [35]

[29] Mikhail L. Gromov. Hyperbolic groups. In *Essays in group theory*, volume 8 of *Math. Sci. Res. Inst. Publ.*, pages 75–263. Springer, New York, 1987. doi:10.1007/978-1-4613-9586-7_3. [1, 9]

[30] Christoph Haase. *On the complexity of model checking counter automata*. PhD thesis, University of Oxford, St Catherine's College, 2011. [34, 35]

[31] Christian Hagenah. *Gleichungen mit regulären Randbedingungen über freien Gruppen*. PhD thesis, University of Stuttgart, 2000. [15, 16, 23]

[32] Frédéric Haglund and Daniel T. Wise. Coxeter groups are virtually special. *Adv. Math.*, 224(5):1890–1903, 2010. doi:10.1016/j.aim.2010.01.011. [5]

[33] Nico Haubold, Markus Lohrey, and Christian Mathissen. Compressed decision problems for graph products of groups and applications to (outer) automorphism groups. *International Journal of Algebra and Computation*, 22(8), 2013. doi:10.1142/S0218196712400073. [3]

[34] Graham Higman. A finitely generated infinite simple group. *J. London Math. Soc.*, 26:61–64, 1951. doi:10.1112/jlms/s1-26.1.61. [6]

[35] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial-time algorithm for deciding equivalence of normed context-free processes. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science, FOCS 1994*, pages 623–631. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365729. [14]

[36] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158(1&2):143–159, 1996. doi:10.1016/0304-3975(95)00064-X. [14]

[37] Derek F. Holt, Markus Lohrey, and Saul Schleimer. Compressed decision problems in hyperbolic groups. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPIcs*, pages 37:1–37:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.37. [7]

[38] Derek F. Holt and Sarah Rees. The compressed word problem in relatively hyperbolic groups. *CoRR*, 2020. arXiv:2005.13917. [2]

[39] Derek F. Holt, Sarah Rees, and Claas E. Röver. *Groups, Languages and Automata*, volume 88 of *London Mathematical Society Student Texts*. Cambridge University Press, 2017. doi:10.1017/9781316588246. [28]

[40] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison–Wesley, Reading, MA, 1979. [7]

[41] Artur Jeż. Faster fully compressed pattern matching by recompression. *ACM Transactions on Algorithms*, 11(3):20:1–20:43, 2015. doi:10.1145/2631920. [14]

[42] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. doi:10.1007/978-1-4684-2001-2_9. [4]

[43] Marek Karpinski, Wojciech Rytter, and Ayumi Shinohara. Pattern-matching for strings with short descriptions. In *Combinatorial pattern matching (Espoo, 1995)*, volume 937 of *Lecture Notes in Comput. Sci.*, pages 205–214. Springer, Berlin, 1995. doi:10.1007/3-540-60044-2_44. [14]

[44] Martin Kassabov and Francesco Matucci. The simultaneous conjugacy problem in groups of piecewise linear functions. *Groups, Geometry, and Dynamics*, 6(2):279–315, 2012. doi:10.4171/GGD/158. [3]

[45] Daniel König and Markus Lohrey. Evaluation of circuits over nilpotent and polycyclic groups. *Algorithmica*, 80(5):1459–1492, 2018. doi:10.1007/s00453-017-0343-z. [4]

[46] Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. In *Algebra and Computer Science*, volume 677 of *Contemporary Mathematics*, pages 138–153. American Mathematical Society, 2016. doi:10.1090/conm/677. [6]

[47] Markus Lohrey. Word problems and membership problems on compressed words. *SIAM Journal on Computing*, 35(5):1210 – 1240, 2006. doi:10.1137/S0097539704445950. [2, 15]

[48] Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016. [4]

[49] Markus Lohrey. *The Compressed Word Problem for Groups*. SpringerBriefs in Mathematics. Springer, 2014. doi:10.1007/978-1-4939-0748-9. [2, 4, 5, 12, 13, 14, 15, 16, 17, 26, 27]

[50] Markus Lohrey. Knapsack in hyperbolic groups. *Journal of Algebra*, 545:390–415, 2020. doi:10.1016/j.jalgebra.2019.04.008. [6, 10]

[51] Markus Lohrey and Georg Zetzsche. Knapsack in graph groups. *Theory of Computing Systems*, 62(1):192–246, 2018. doi:10.1007/s00224-017-9808-3. [6]

[52] Jeremy Macdonald. Compressed words and automorphisms in fully residually free groups. *International Journal of Algebra and Computation*, 20(3):343–355, 2010. doi:10.1142/S021819671000542X. [5]

[53] Jeremy MacDonald, Alexei G. Myasnikov, and Denis Ovchinnikov. Low-complexity computations for nilpotent subgroup problems. *International Journal of Algebra and Computation*, 29(4):639–661, 2019. doi:10.1142/S021819671950019X. [3]

[54] Wilhelm Magnus, Abraham Karrass, and Donald Solitar. *Combinatorial group theory*. Dover Publications, Inc., New York, revised edition, 1976. Presentations of groups in terms of generators and relations. [1]

[55] Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality-tests in polylogarithmic time. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1994*, pages 213–222. ACM/SIAM, 1994. [14]

[56] Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997. doi:10.1007/BF02522825. [14]

[57] Alexei G. Myasnikov, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in groups. *Mathematics of Computation*, 84:987–1016, 2015. doi:10.1090/S0025-5718-2014-02880-9. [3, 34]

[58] Alexei G. Myasnikov, Alexander Ushakov, and Dong Wook Won. The word problem in the Baumslag group with a non-elementary Dehn function is polynomial time decidable. *Journal of Algebra*, 345(1):324–342, 2011. doi:10.1016/j.jalgebra.2011.07.024. [6]

[59] Alexei G. Myasnikov, Alexander Ushakov, and Dong Wook Won. Power circuits, exponential algebra, and time complexity. *International Journal of Algebra and Computation*, 22(6), 2012. doi:10.1142/S0218196712500476. [6]

[60] Alexander Yu. Ol′shanskiĭ. Hyperbolicity of groups with subquadratic isoperimetric inequality. *Internat. J. Algebra Comput.*, 1(3):281–289, 1991. doi:10.1142/S0218196791000183. [1]

[61] Alexander Yu. Ol′shanskiĭ. Almost every group is hyperbolic. *Internat. J. Algebra Comput.*, 2(1):1–17, 1992. doi:10.1142/S0218196792000025. [1]

[62] Panagiotis Papasoglu. On the sub-quadratic isoperimetric inequality. In *Geometric group theory (Columbus, OH, 1992)*, volume 3 of *Ohio State Univ. Math. Res. Inst. Publ.*, pages 149–157. de Gruyter, Berlin, 1995. doi:10.1515/9783110810820.149. [1]

[63] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *Proceedings of the 2nd Annual European Symposium on Algorithms, ESA 1994*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470. Springer, 1994. doi:10.1007/BFb0049431. [14]

[64] A. N. Platonov. An isoparametric function of the Baumslag-Gersten group. *Vestnik Moskov. Univ. Ser. I Mat. Mekh.*, 3:12–17, 70, 2004. [6]

[65] Eliyahu Rips and Zlil Sela. Canonical representatives and equations in hyperbolic groups. *Inventiones Mathematicae*, 120:489–512, 1995. doi:10.1007/BF01241140. [10]

[66] Saul Schleimer. Polynomial-time word problems. *Commentarii Mathematici Helvetici*, 83(4):741–765, 2008. doi:10.4171/CMH/142. [2, 3]

[67] Charles C. Sims. Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, pages 169–183. Pergamon, Oxford, 1970. doi:10.1016/C2013-0-02156-1. [4]

[68] Stephan Waack. The parallel complexity of some constructions in combinatorial group theory. *Journal of Information Processing and Cybernetics EIK*, 26:265–281, 1990. [5]

[69] Jan Philipp Wächter and Armin Weiß. An Automaton Group with PSPACE-Complete Word Problem. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020*, volume 154 of *LIPIcs*, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.6. [5]

[70] Armin Weiß. A logspace solution to the word and conjugacy problem of generalized Baumslag-Solitar groups. In *Algebra and Computer Science*, volume 677 of *Contemporary Mathematics*, pages 185–212. American Mathematical Society, 2016. doi:10.1090/conm/677. [1]

[71] Daniel T. Wise. Research announcement: the structure of groups with a quasiconvex hierarchy. *Electronic Research Announcements in Mathematical Sciences*, 16:44–55, 2009. doi:10.3934/era.2009.16.44. [5]

University of Warwick, UK
E-mail address: D.F.Holt@warwick.ac.uk

Universität Siegen, Germany
E-mail address: lohrey@eti.uni-siegen.de

University of Warwick, UK
E-mail address: s.schleimer@warwick.ac.uk