The Use of Straight Line Programs in Computational Group Theory

Derek Holt

University of Warwick

AGTA 2023, Lecce, 5–9 June 2023

Contents

Introduction

Computational group theory splits into two not completely disjoint parts:

computing in **finite** and **infinite** groups.

In both cases, research is divided into

- **theoretical**: proving decidability and complexity results;
- **practical**: implementing algorithms efficiently.

Introduction

Computational group theory splits into two not completely disjoint parts:

computing in **finite** and **infinite** groups.

In both cases, research is divided into

- **theoretical**: proving decidability and complexity results;
- **practical**: implementing algorithms efficiently.

For computation in finite groups these two approaches are generally compatible and mutually complementary; i.e. in general, algorithms with good complexity have implementations that run faster, at least in large examples.

This is less true for computation in infinite groups, where a polynomial-time algorithm might involve constants that are too large for practical purposes.

Computing in finite groups

For algorithmic purposes, finite groups are most conveniently represented as

1. **permutation groups** (subgroups of $\mathrm{Sym}(n)$ for some n);
2. **matrix groups over finite fields** (subgroups of $\mathrm{GL}(d, q)$ for some $d > 0$ and prime power q); or
3. solvable groups defined by **power-conjugate presentations**.

Computing in finite groups

For algorithmic purposes, finite groups are most conveniently represented as

1. **permutation groups** (subgroups of $\mathrm{Sym}(n)$ for some n);
2. **matrix groups over finite fields** (subgroups of $\mathrm{GL}(d, q)$ for some $d > 0$ and prime power q); or
3. solvable groups defined by **power-conjugate presentations**.

For permutation and matrix groups, BSGS (base and strong generating set) based methods, introduced originally by Charles Sims are used extensively.

These methods are less suitable for large finite matrix groups without subgroups of reasonably small index, and methods involving computing a **Composition Tree** for the group, due to Leedham-Green, O'Brien and many others, have now been effectively implemented.

Computing in infinite groups

There are two main strands here:

computing in **finitely presented** groups and in **matrix** groups.

Computing in infinite groups

There are two main strands here:

computing in **finitely presented** groups and in **matrix** groups.

Exact computation in infinite groups of matrices (over $\mathbb{Z}$, $\mathbb{Q}$, number fields, function fields, for example) is a relatively new field of research, in which significant advances (both theoretical and practical) have been made recently by Flannery, Detinko, O'Brien, Hulpke, et al.

Computing in infinite groups

There are two main strands here:

computing in **finitely presented** groups and in **matrix** groups.

Exact computation in infinite groups of matrices (over $\mathbb{Z}$, $\mathbb{Q}$, number fields, function fields, for example) is a relatively new field of research, in which significant advances (both theoretical and practical) have been made recently by Flannery, Detinko, O'Brien, Hulpke, et al.

The input is a finite set of matrices that generate a group G.

> **Finiteness** of G can be decided quickly, as can nilpotency.
>
> **The Tits alternative** can be decided for G.

But it is unknown whether it is possible to decide whether G is free, even on the given generators.

Computing in finitely presented groups: coset enumeration

Suppose now that the group G is defined by a presentation $G = \langle X \mid R \rangle$ with X and R finite. We let $\Sigma := X \cup X^{-1}$. So elements of G are represented by words $w \in \Sigma^*$.

Computing in finitely presented groups: coset enumeration

Suppose now that the group G is defined by a presentation $G = \langle X \mid R \rangle$ with X and R finite. We let $\Sigma := X \cup X^{-1}$. So elements of G are represented by words $w \in \Sigma^*$.

The **coset enumeration** procedure was formulated by Todd and Coxeter in the 1930s and implemented first in 1953.

The input is X, R, and a finite set $Y \subset \Sigma^*$ generating a subgroup $H = \langle Y \rangle$ of G.

Computing in finitely presented groups: coset enumeration

Suppose now that the group G is defined by a presentation $G = \langle X \mid R \rangle$ with X and R finite. We let $\Sigma := X \cup X^{-1}$. So elements of G are represented by words $w \in \Sigma^*$.

The **coset enumeration** procedure was formulated by Todd and Coxeter in the 1930s and implemented first in 1953.

The input is X, R, and a finite set $Y \subset \Sigma^*$ generating a subgroup $H = \langle Y \rangle$ of G.

If $|G : H|$ is finite, then it, together with the associated action of G on the cosets of H is computed. A presentation of H can also be computed.

So we are effectively computing a **finite quotient** of G, namely the image of this action.

Computing in finitely presented groups: coset enumeration

Suppose now that the group G is defined by a presentation $G = \langle X \mid R \rangle$ with X and R finite. We let $\Sigma := X \cup X^{-1}$. So elements of G are represented by words $w \in \Sigma^*$.

The **coset enumeration** procedure was formulated by Todd and Coxeter in the 1930s and implemented first in 1953.

The input is X, R, and a finite set $Y \subset \Sigma^*$ generating a subgroup $H = \langle Y \rangle$ of G.

If $|G : H|$ is finite, then it, together with the associated action of G on the cosets of H is computed. A presentation of H can also be computed.

So we are effectively computing a **finite quotient** of G, namely the image of this action.

Related applications include finding all subgroups of G up to a specified index, so we can systematically enumerate finite quotients of G.

Other quotient algorithms

Other quotient algorithms include algorithms to compute:

- the largest abelian quotient (Smith Normal Form);
- finite p-quotients for a specified prime p;
- nilpotent quotients;
- polycyclic quotients;
- (solvable quotients).

Other quotient algorithms

Other quotient algorithms include algorithms to compute:

- the largest abelian quotient (Smith Normal Form);
- finite p-quotients for a specified prime p;
- nilpotent quotients;
- polycyclic quotients;
- (solvable quotients).

But, unless the group is virtually polycyclic, none of the above techniques enables computations with G itself rather than in a proper quotient of G.

Unfortunately the most natural problems involving G itself, including the **Dehn Problems**, have all been proved to be theoretically unsolvable.

The Dehn Problems

These are decision problems that were formulated by Dehn in 1911. They are basically theoretical questions, but Dehn described implementable algorithms for their solution in some cases.

The Dehn Problems

These are decision problems that were formulated by Dehn in 1911. They are basically theoretical questions, but Dehn described implementable algorithms for their solution in some cases.

- **The Word Problem** WP(G): given $w \in \Sigma^*$, is $w =_G 1$?
- **The Conjugacy Problem**: given $v, w \in \Sigma^*$, does there exists $c \in \Sigma^*$ with $w =_G c^{-1}vc$?
- **The Isomorphism problem**: given another finitely presented group $G' = \langle X' \mid R' \rangle$, is $G \cong G'$?
  (This is the most difficult, both theoretically and practically.)

The Dehn Problems

These are decision problems that were formulated by Dehn in 1911. They are basically theoretical questions, but Dehn described implementable algorithms for their solution in some cases.

- **The Word Problem** WP(G): given $w \in \Sigma^*$, is $w =_G 1$?
- **The Conjugacy Problem**: given $v, w \in \Sigma^*$, does there exists $c \in \Sigma^*$ with $w =_G c^{-1}vc$?
- **The Isomorphism problem**: given another finitely presented group $G' = \langle X' \mid R' \rangle$, is $G \cong G'$?
  (This is the most difficult, both theoretically and practically.)

To this, we can add

- **The Generalized Word Problem** GWP(G, H): given finite $Y \subset \Sigma^*$ generating $H = \langle Y \rangle$ and $w \in \Sigma^*$, is $w \in H$?

The word problem

Although they are undecidable in general, they are solvable in certain classes of groups.

The word problem

Although they are undecidable in general, they are solvable in certain classes of groups.

Effective implementations of word problem solutions include:

- **collection** in polycyclic groups;
- putting words into normal form using **finite state automata (**fsa**)** in **automatic groups**

The word problem

Although they are undecidable in general, they are solvable in certain classes of groups.

Effective implementations of word problem solutions include:

- **collection** in polycyclic groups;
- putting words into normal form using **finite state automata (**fsa**)** in **automatic groups**

The class of automatic groups includes: free groups, hyperbolic groups, Coxeter groups, braid groups, mapping class groups, many types of Artin groups, and virtually abelian groups, but **not** other polycyclic groups.

The word problem

Although they are undecidable in general, they are solvable in certain classes of groups.

Effective implementations of word problem solutions include:

- **collection** in polycyclic groups;
- putting words into normal form using **finite state automata (**fsa**)** in **automatic groups**

The class of automatic groups includes: free groups, hyperbolic groups, Coxeter groups, braid groups, mapping class groups, many types of Artin groups, and virtually abelian groups, but **not** other polycyclic groups.

The normal forms most frequently used for automatic groups are **shortlex**: order Σ and then, for $v, w \in \Sigma^*$, defined $v < w$ if either $\ell(v) < \ell(w)$; or $\ell(v) = \ell(w)$ and $v <_{\text{lex}} w$.

Straight line programs

A **straight line program** (**SLP**) is a method of representing certain words over an alphabet Σ in a compressed form. This is achieved by extending the alphabet by introducing new 'letters' $w_1, w_2, w_3, \ldots$, where each w_k is defined as a word over the alphabet $\Sigma \cup \{w_1, w_2, \ldots, w_{k-1}\}$.

Straight line programs

A **straight line program** (**SLP**) is a method of representing certain words over an alphabet Σ in a compressed form. This is achieved by extending the alphabet by introducing new 'letters' $w_1, w_2, w_3, \ldots$, where each w_k is defined as a word over the alphabet $\Sigma \cup \{w_1, w_2, \ldots, w_{k-1}\}$.

Example

Let $\Sigma = \{a, b\}$ and define

$$w_1 := b,\ w_2 := a,\ w_i := w_{i-1}w_{i-2} \text{ for } i \geq 2.$$

So, if we rewrite w_i for $i > 2$ as words over Σ, we get

$$w_3 = ab, w_4 = aba, w_5 = aba^2b, w_6 = aba^2(ba)^2, w_7 = aba^2ba(ba^2)^2b,$$

and the length of w_n grows exponentially with n.

Formally, an SLP can be defined to be a **context-free grammar** $\mathcal{G} = (V, S, P)$ over an alphabet Σ that generates a unique word $\rho(\mathcal{G})$.

It has a set V of **variables** (the symbols w_i in the notation above) including a **start variable** S, and a set P of **productions**, which specify definitions of the w_i.

For each variable $A \in V$, there is a single production of the form $A \to (\Sigma \cup V)^*$. The requirement that $\mathcal{G}$ generates a unique word implies that we can order the variables in V such that S is the largest in the ordering, and any variables occurring in the right hand side of the production $A \to (\Sigma \cup V)^*$ must be less than A.

Formally, an SLP can be defined to be a **context-free grammar** $\mathcal{G} = (V, S, P)$ over an alphabet Σ that generates a unique word $\rho(\mathcal{G})$.

It has a set V of **variables** (the symbols w_i in the notation above) including a **start variable** S, and a set P of **productions**, which specify definitions of the w_i.

For each variable $A \in V$, there is a single production of the form $A \to (\Sigma \cup V)^*$. The requirement that $\mathcal{G}$ generates a unique word implies that we can order the variables in V such that S is the largest in the ordering, and any variables occurring in the right hand side of the production $A \to (\Sigma \cup V)^*$ must be less than A.

For the example above, a grammar $\mathcal{G}$ over $\Sigma = \{a, b\}$ with $\rho(\mathcal{G}) = w_6$ could be defined by $V = \{w_1, w_2, w_3, w_4, w_5, S\}$, and

$$P = \{S \to w_5 w_4, w_5 \to w_4 w_3, w_4 \to w_3 w_2, w_3 \to w_2 w_1, w_2 \to a, w_1 \to b\}$$

Straight line programs in finite group computations

The Schreier-Sims algorithm for computing a **BSGS** of a finite permutation or matrix group G makes use of SLPs.

The group G is generally defined by a generating set X and, with $\Sigma := X \cup X^{-1}$ as before, the algorithm extends X to a strong generating set (if necessary) by introducing new strong generators $w_1, w_2, \ldots, w_k$, where each w_i is defined as a word over $\Sigma \cup \{w_1^{\pm 1}, \ldots, w_{k-1}^{\pm 1}\}$.

Straight line programs in finite group computations

The Schreier-Sims algorithm for computing a **BSGS** of a finite permutation or matrix group G makes use of SLPs.

The group G is generally defined by a generating set X and, with $\Sigma := X \cup X^{-1}$ as before, the algorithm extends X to a strong generating set (if necessary) by introducing new strong generators $w_1, w_2, \ldots, w_k$, where each w_i is defined as a word over $\Sigma \cup \{w_1^{\pm 1}, \ldots, w_{k-1}^{\pm 1}\}$.

It is a very difficult problem in large groups (such as the Rubik Cube group) to write an arbitrary element of G a word over Σ, but easy to write it as a word over the strong generators and their inverses. This is particularly useful when defining images of elements $g \in G$ under homomorphisms.

The more recent **Composition Tree** algorithm for large matrix groups works in the same way, but with the additional feature that “nice” generating sets for the finite simple groups, such as transvections for the classical groups, have been chosen in advance, and the program attempts to find these nice generators of the given group, and to define them as SLPs in its original generators.

The more recent **Composition Tree** algorithm for large matrix groups works in the same way, but with the additional feature that "nice" generating sets for the finite simple groups, such as transvections for the classical groups, have been chosen in advance, and the program attempts to find these nice generators of the given group, and to define them as SLPs in its original generators.

The program starts by identifying the isomorphism classes of the composition factors of the input group G as abstract simple groups S. Then, for each nonabelian composition factor, it looks for elements that correspond to these nice generators of S, and uses them to set up an effective isomorphism between that factor and a **standard copy** of S.

For example, the standard copy of A_n is $\mathrm{Alt}(n)$ in its natural representation, and the standard copy of $\mathrm{PSL}(d,q)$ is the group $\mathrm{SL}(d,q)$ modulo its scalar subgroup.

Polynomial-time computation with SLPs

Let $\mathcal{G} = (V, S, P)$ be an SLP over Σ, and let $w = \rho(\mathcal{G})$ be the word in Σ^* defined by $\mathcal{G}$.

Polynomial-time computation with SLPs

Let $\mathcal{G} = (V, S, P)$ be an SLP over Σ, and let $w = \rho(\mathcal{G})$ be the word in Σ^* defined by $\mathcal{G}$.

It is not difficult to show that the length $|w|$ of w, which may be exponentially larger than the size of $\mathcal{G}$ can be computed in polynomial-time as a binary or decimal number, by using the fact that, for each production $A \to \alpha_1\alpha_2\cdots\alpha_k$, with $\alpha_i \in V \cup \{\Sigma\}$ we have $|\rho(A)| = \sum_{i=1}^{k} |\rho(\alpha_i)|$.

We can also compute in polynomial time the i-th letter in w for any i with $1 \le i \le |w|$ and we can define SLPs that define $\mathcal{G}_{ij}$ with $\rho(\mathcal{G}_{ij})$ equal to the subword $w[i : j]$ of w between its i-th and j-th letters.

Polynomial-time computation with SLPs

Let $\mathcal{G} = (V, S, P)$ be an SLP over Σ, and let $w = \rho(\mathcal{G})$ be the word in Σ^* defined by $\mathcal{G}$.

It is not difficult to show that the length $|w|$ of w, which may be exponentially larger than the size of $\mathcal{G}$ can be computed in polynomial-time as a binary or decimal number, by using the fact that, for each production $A \to \alpha_1\alpha_2 \cdots \alpha_k$, with $\alpha_i \in V \cup \{\Sigma\}$ we have $|\rho(A)| = \sum_{i=1}^{k} |\rho(\alpha_i)|$.

We can also compute in polynomial time the i-th letter in w for any i with $1 \leq i \leq |w|$ and we can define SLPs that define $\mathcal{G}_{ij}$ with $\rho(\mathcal{G}_{ij})$ equal to the subword $w[i:j]$ of w between its i-th and j-th letters.

A more difficult result due to Plandowski, which is essential for the applications to the compressed word problem, is that, given two SLPs $\mathcal{G}_1$ and $\mathcal{G}_2$ over the same alphabet Σ, we can decide in polynomial time whether $\rho(\mathcal{G}_1) = \rho(\mathcal{G}_2)$. (More generally, we can find their longest common prefix.)

The compressed word and conjugacy problems

In the **compressed word problem** CWP(G) for a group G, the input word w is given in compressed form as an SLP.

The compressed word and conjugacy problems

In the **compressed word problem** CWP(G) for a group G, the input word w is given in compressed form as an SLP.

Of course CWP(G) is solvable if and only if WP(G) is solvable, because compressed words can be expanded to normal words.

But the expanded word can be exponentially longer than the input compressed word, so we might expect CWP(G) to be more difficult than WP(G) in terms of complexity.

The compressed word and conjugacy problems

In the **compressed word problem** CWP(G) for a group G, the input word w is given in compressed form as an SLP.

Of course CWP(G) is solvable if and only if WP(G) is solvable, because compressed words can be expanded to normal words.

But the expanded word can be exponentially longer than the input compressed word, so we might expect CWP(G) to be more difficult than WP(G) in terms of complexity.

Indeed, there are groups, such as the Grigorchuk group and certain wreath products $H \wr \mathbb{Z}$, in which the space complexity of CWP(G) is provably larger than that of WP(G) (PSPACE-complete and LOGSPACE, respectively), and the time complexity is also conjectured to be higher.

On the other hand, it has been proved by Marcus Lohrey and others that CWP(G) is solvable in polynomial time in the following classes of finitely generated groups:

1. free groups;
2. nilpotent groups;
3. right-angled Artin groups;
4. Coxeter groups.

On the other hand, it has been proved by Marcus Lohrey and others that CWP(G) is solvable in polynomial time in the following classes of finitely generated groups:

1. free groups;
2. nilpotent groups;
3. right-angled Artin groups;
4. Coxeter groups.

More recently, Lohrey, Schleimer and Holt have shown that the compressed word and conjugacy problems are solvable in polynomial-time in hyperbolic groups.

This result has been extended by Holt and Rees to groups that are hyperbolic relative to a collection of free abelian subgroups.

On the other hand, it has been proved by Marcus Lohrey and others that CWP(G) is solvable in polynomial time in the following classes of finitely generated groups:

1. free groups;
2. nilpotent groups;
3. right-angled Artin groups;
4. Coxeter groups.

More recently, Lohrey, Schleimer and Holt have shown that the compressed word and conjugacy problems are solvable in polynomial-time in hyperbolic groups.

This result has been extended by Holt and Rees to groups that are hyperbolic relative to a collection of free abelian subgroups.

The results proving the solvability of CWP(G) in polynomial time are theoretical and, except possibly for the case of free groups, not suitable for effective implementation, because the constants involved are too large. The algorithms typically involve testing all words up to some bounded but moderately large length.

Solving the word problem in automorphism groups

If $\mathrm{Aut}(G)$ is finitely generated by a set Φ, then we can use the compressed word problem in G to solve the word problem in $\mathrm{Aut}(G)$

Suppose that $\alpha = \varphi_1\varphi_2\cdots\varphi_n$ with each $\varphi_i \in \Phi$, and we wish to decide whether $\alpha = 1_{\mathrm{Aut}(G)}$.

Solving the word problem in automorphism groups

If $\mathrm{Aut}(G)$ is finitely generated by a set Φ, then we can use the compressed word problem in G to solve the word problem in $\mathrm{Aut}(G)$

Suppose that $\alpha = \varphi_1\varphi_2\cdots\varphi_n$ with each $\varphi_i \in \Phi$, and we wish to decide whether $\alpha = 1_{\mathrm{Aut}(G)}$.

To do this, for each $x \in \Sigma$, we define an SLP $\mathcal{G}_x$ over Σ with variables $A_{a,k}$ for each $a \in \Sigma$ and $0 \le k \le n$ and start variable $A_{x,n}$.

For each $a \in \Sigma$ and k with $0 \le k \le n$, we define $\phi_k(A_{a,k-1})$ to be the word in the variables $\{A_{y,k-1} : y \in \Sigma\}$ that corresponds to $\phi_k(a)$.

So, for example, if $\phi_k(a) = abba$ then

$$\phi_k(A_{a,k-1}) = A_{a,k-1}A_{b,k-1}A_{b,k-1}A_{a,k-1}.$$

The productions of each $\mathcal{G}_x$ are, for each $a \in A$,

$$\begin{array}{rcl} A_{a,0} & \to & a; \\ A_{a,k} & \to & \phi_k(A_{a,k-1}) \text{ for } 1 \leq k < n; \\ A_{a,n} & \to & a^{-1}\phi_n(A_{a,n-1}). \end{array}$$

Then $\alpha = 1_{\mathrm{Aut}(G)}$ if and only if $\alpha(x) = x$ for all $x \in X$, which is the case if and only $\rho(\mathcal{G}_x) =_G 1$ for all $x \in X$.

The productions of each $\mathcal{G}_x$ are, for each $a \in A$,

$$\begin{aligned} A_{a,0} &\rightarrow a; \\ A_{a,k} &\rightarrow \phi_k(A_{a,k-1}) \text{ for } 1 \leq k < n; \\ A_{a,n} &\rightarrow a^{-1}\phi_n(A_{a,n-1}). \end{aligned}$$

Then $\alpha = 1_{\mathrm{Aut}(G)}$ if and only if $\alpha(x) = x$ for all $x \in X$, which is the case if and only $\rho(\mathcal{G}_x) =_G 1$ for all $x \in X$.

If the compressed word problem in G is solvable in polynomial-time, and $\mathrm{Aut}(G)$ is finitely generated, then the ordinary word problem $\mathrm{WP}(\mathrm{Aut}(G))$ is solvable in polynomial-time.

In particular, if G is a hyperbolic group, then $\mathrm{WP}(\mathrm{Aut}(G))$ is solvable in polynomial time.

The compressed word problem in hyperbolic groups

Let $G = \langle \Sigma \rangle$ be hyperbolic, and let $\mathcal{G}$ be an SLP defining a compressed word in Σ^*.

The basic idea is to compute an SLP defining the shortlex normal form $\mathrm{nf}(\rho(A))$ for each variable A of $\mathcal{G}$, finishing with $\mathrm{nf}(\rho(\mathcal{G}))$, which is empty if and only if $\rho(\mathcal{G}) =_G 1$.

The compressed word problem in hyperbolic groups

Let $G = \langle \Sigma \rangle$ be hyperbolic, and let $\mathcal{G}$ be an SLP defining a compressed word in Σ^*.

The basic idea is to compute an SLP defining the shortlex normal form $\mathrm{nf}(\rho(A))$ for each variable A of $\mathcal{G}$, finishing with $\mathrm{nf}(\rho(\mathcal{G}))$, which is empty if and only if $\rho(\mathcal{G}) =_G 1$.

The principal step is to compute an SLP for $\mathrm{nf}(\rho(A))$ from SLPs defining $\mathrm{nf}(\rho(B))$ and $\mathrm{nf}(\rho(C))$, for each production $A \to BC$.

The compressed word problem in hyperbolic groups

Let $G = \langle \Sigma \rangle$ be hyperbolic, and let $\mathcal{G}$ be an SLP defining a compressed word in Σ^*.

The basic idea is to compute an SLP defining the shortlex normal form $\mathrm{nf}(\rho(A))$ for each variable A of $\mathcal{G}$, finishing with $\mathrm{nf}(\rho(\mathcal{G}))$, which is empty if and only if $\rho(\mathcal{G}) =_G 1$.

The principal step is to compute an SLP for $\mathrm{nf}(\rho(A))$ from SLPs defining $\mathrm{nf}(\rho(B))$ and $\mathrm{nf}(\rho(C))$, for each production $A \to BC$.

This can be done using the geometry of the hyperbolic triangle with sides labelled by $\mathrm{nf}(\rho(A))$, $\mathrm{nf}(\rho(B))$, and $\mathrm{nf}(\rho(C))$: the **meeting points** of the triangle can be located using standard SLP operations.

The compressed word problem in hyperbolic groups

Let $G = \langle \Sigma \rangle$ be hyperbolic, and let $\mathcal{G}$ be an SLP defining a compressed word in Σ^*.

The basic idea is to compute an SLP defining the shortlex normal form $\mathrm{nf}(\rho(A))$ for each variable A of $\mathcal{G}$, finishing with $\mathrm{nf}(\rho(\mathcal{G}))$, which is empty if and only if $\rho(\mathcal{G}) =_G 1$.

The principal step is to compute an SLP for $\mathrm{nf}(\rho(A))$ from SLPs defining $\mathrm{nf}(\rho(B))$ and $\mathrm{nf}(\rho(C))$, for each production $A \to BC$.

This can be done using the geometry of the hyperbolic triangle with sides labelled by $\mathrm{nf}(\rho(A))$, $\mathrm{nf}(\rho(B))$, and $\mathrm{nf}(\rho(C))$: the **meeting points** of the triangle can be located using standard SLP operations.

But it turned out to be frustratingly difficult to do this while keeping the size of the SLP to define $\mathrm{nf}(\rho(\mathcal{G}))$ polynomially bounded.

Technical solutions were eventually found independently by Schleimer and Lohrey.

The compressed word problem in relatively hyperbolic groups

The Holt/Rees generalization to groups hyperbolic relative to free abelian subgroups makes heavy use of a significant paper of **Yago Antolin** and **Laura Ciobanu** on the geometry of relatively hyperbolic groups.

They proved in particular that these groups are shortlex automatic, when the parabolic subgroups are virtually abelian, so we initially hoped for a moderately straightforward generalization.

The compressed word problem in relatively hyperbolic groups

The Holt/Rees generalization to groups hyperbolic relative to free abelian subgroups makes heavy use of a significant paper of **Yago Antolin** and **Laura Ciobanu** on the geometry of relatively hyperbolic groups.

They proved in particular that these groups are shortlex automatic, when the parabolic subgroups are virtually abelian, so we initially hoped for a moderately straightforward generalization.

But there were serious problems with the geometric arguments, and in the end we have to use a different "automatic structure" using a normal in which normal form words are geodesic in the extended Cayley graph, which is a hyperbolic graph in which all elements of the parabolic subgroups are included as generators, and so they label edges of length one.

This new structure is only **asynchronously automatic**, but fortunately that turned out not to be serious obstacle.

The generalized word problem

Let $H = \langle Y \rangle \leq G$ with $Y \leq \Sigma^*$ and Y finite. Recall that the **generalized word problem** $\mathrm{GWP}(G, H)$ is the problem of deciding whether a given $w \in \Sigma^*$ lies in H.

The generalized word problem

Let $H = \langle Y \rangle \leq G$ with $Y \leq \Sigma^*$ and Y finite. Recall that the **generalized word problem** $\mathrm{GWP}(G, H)$ is the problem of deciding whether a given $w \in \Sigma^*$ lies in H.

The **Stallings Folding** method is a linear-time algorithm for solving this problem when G is (virtually) free. A fsa is constructed that accepts a *reduced* word in $w \in \Sigma^*$ if and only if $w \in H$.

In fact Stallings Folding is essentially the same as coset enumeration.

The generalized word problem

Let $H = \langle Y \rangle \leq G$ with $Y \leq \Sigma^*$ and Y finite. Recall that the **generalized word problem** $\mathrm{GWP}(G, H)$ is the problem of deciding whether a given $w \in \Sigma^*$ lies in H.

The **Stallings Folding** method is a linear-time algorithm for solving this problem when G is (virtually) free. A fsa is constructed that accepts a *reduced* word in $w \in \Sigma^*$ if and only if $w \in H$.

In fact Stallings Folding is essentially the same as coset enumeration.

In the **fully compressed generalized word problem** the generators Y of H and the input word w are all given as SLPs.

The generalized word problem

Let $H = \langle Y \rangle \leq G$ with $Y \leq \Sigma^*$ and Y finite. Recall that the **generalized word problem** $\mathrm{GWP}(G, H)$ is the problem of deciding whether a given $w \in \Sigma^*$ lies in H.

The **Stallings Folding** method is a linear-time algorithm for solving this problem when G is (virtually) free. A fsa is constructed that accepts a *reduced* word in $w \in \Sigma^*$ if and only if $w \in H$.

In fact Stallings Folding is essentially the same as coset enumeration.

In the **fully compressed generalized word problem** the generators Y of H and the input word w are all given as SLPs.

A challenging open problem is whether this problem is solvable in polynomial time for free groups G.

It has been proved recently by Marco Linton that it is solvable in polynomial time under the assumption $|Y| \leq k$ for some fixed constant k.

The End

Thank you for listening!