A Lean tactic to solve the word problem in One Relator Groups

Christopher Hughes

February 4, 2021

1 Introduction

This paper describes an implementation of Magnus' method for solving the word problem in one-relator groups [MS73][Put20]. This implementation is used as part of a Lean tactic to prove equalities in groups given a single equality as a hypothesis; for example, it might prove that $abab^2 = 1$ implies $ab = ba$.

In order to be used as part of a Lean tactic, the implementation of the algorithm must not only decide the problem, but also produce a proof certificate that can be converted into a Lean proof to be checked by Lean's kernel. The Lean tactic can prove that two words in a one relator group are equal, but not that they are unequal.

This paper starts by defining and stating a few important results about both HNN extensions and free groups. Next, the proof certificate is described, and then the algorithm is described along with how to generate the proof certificates.

2 Introduction to Lean

Lean [dMKA+18] is a proof assistant; it is a language capable of expressing mathematical propositions and definitions, and also a language for writing formal proofs of these propositions which can then be checked by Lean's kernel. A formal proof checked by Lean's kernel provides an extremely high level of confidence in a checked proof, at the expense of requiring a lot of time for a user to write a proof in a completely formal language.

As well as being a proof assistant, Lean is also a programming language. The definitions that are written in Lean are also executable programs, and Lean can be used to prove correctness of programs written in Lean as well as pure mathematical proofs. Lean can be used as a programming language to write Lean tactics, described in the next section.

There is a large library of formal mathematics in Lean, called mathlib [mC20]. As of January 2021 this contains 470000 lines of code.

2.1 Terms and Types

Lean is based on type theory, every term has a type, including types themselves. Lean makes use of the Curry-Howard correspondence, every proposition is a type, and the inhabitants of

the proposition can be thought of as proofs of the proposition.

2.1.1 Curry Howard Correspondence

2.2 Tactics

Because of the difficulty of writing a formal proof in Lean directly, it is possible to write tactics in Lean. The tactic framework is a domain specific language for writing Lean expressions, which include Lean proofs.

When using tactics to write a Lean proof, a tactic state is displayed to a user. This tactic state can be manipulated using tactics. An example of a tactic state is displayed below.

Example 2.0.1. The code below is a statement of the theorem that if G is a group, and $a, b \in G$ are such that $ab = b^2a$, then for any natural number n, $a^nba^{-n} = b^{2^n}$. Between the `begin ... end`, the user can write tactics to produce a proof of the proposition.

```
example {G : Type*} [group G] (a b : G) (h : a * b = b ^ 2 * a) (n : ℕ) :
  a ^ n * b * a ^ -↑n = b ^ 2 ^ n :=
begin

end
```

If the user types the above code then the following tactic state is displayed to the user.

```
1 goal
G : Type u_1,
_inst_1 : group G,
a b : G,
h : a * b = b ^ 2 * a,
n : ℕ
⊢ a ^ n * b * a ^ -↑n = b ^ 2 ^ n
```

The turnstile ⊢ indicates the current goal, the proposition the user is currently trying to prove. The preceding lines indicate the hypotheses and variables. `G : Type u_1` indicates that `G` is a type in an arbitrary universe `u_1`. `_inst_1` is a group structure on the Type `G`. A term of type `group G`, is a tuple containing a binary operation on `G`, an identity, and an inverse function, as well as proofs that these satisfy the group axioms. `a` and `b` are elements of `G`, `h` is the hypothesis that `a * b = b ^ 2 * a`, and `n` is a natural number.

If the user writes a tactic, then the tactic state will change. If the user wanted to prove this by induction on `n` they could write

```
induction n with k ih
```

The tactic state after this line will now be

```
2 goals
case nat.zero
G : Type u_1,
_inst_1 : group G,
a b : G,
```

```
h : a * b = b ^ 2 * a
⊢ a ^ 0 * b * a ^ -↑0 = b ^ 2 ^ 0

case nat.succ
G : Type u_1,
_inst_1 : group G,
a b : G,
h : a * b = b ^ 2 * a,
k : ℕ,
ih : a ^ k * b * a ^ -↑k = b ^ 2 ^ k
⊢ a ^ k.succ * b * a ^ -↑(k.succ) = b ^ 2 ^ k.succ
```

The tactic split the goal into two cases, in the first case, labelled `case nat.zero`, `n` is zero, and in the second case, labelled `case nat.succ n` is the successor of `k`. In the second case, an induction hypothesis `ih` has been added.

There are many other tactics that allow different manipulations of a tactic state.

3 Free Group

The free group is implemented in Lean as the set of reduced words. An element of the free group over a type S of letters is a list of pairs $S \times \mathbb{Z}$, the letter and the exponent. A list of the exponent part of every element of the list is non zero, and no two adjacent elements of the list have the same letter. The free group is the set of reduced lists.

Multiplication of elements of the free group is implemented by appending the lists whilst replacing any adjacent occurrences of (s, m) and (s, n) with $(s, m + n)$, and removing any occurrence of $(s, 0)$. Inversion is given by reversing the list and negating the exponent part of every pair. The identity is given by the empty list.

Definition 3.1 (Length)**.** The length of a word w in the free group is the sum of the absolute values of the exponent parts of each element of the corresponding reduced list.

Definition 3.2 (Cyclically Reduced)**.** A word w in the free group is cyclically reduced if it cannot be made shorter by conjugating.

We state the following theorem *Freiheitsatz*. This theorem is an important part of the correctness of Magnus' method. The proof is omitted.

Theorem 3.3 (Freiheitsatz)**.** Suppose $F(S)$ is the free group over a set S and r is a a cyclically reduced word, and $T \subset S$ is a set of letters such that r cannot be written using any letters in T. Then T is a basis for a free subgroup of $F(S)$. [MS73]

4 HNN Extensions

Magnus' method makes use of isomorphisms between one-relator groups and HNN extensions. In this section we define an HNN extension of a group relative to an an isomorphism between two subgroups.

Definition 4.1 (HNN Extension)**.** Given a group G, subgroups A and B of G, and an isomorphism $\phi : A \to B$, we can define the *HNN extension* relative to ϕ of G. Let $\langle t \rangle$ be an infinite cyclic group generated by t. The HNN extension is the coproduct of G and $\langle t \rangle$ quotiented by the normal closure of the set $\{tat^{-1}\phi(a^{-1})|a \in A\}$

Definition 4.2 (HNN normal form)**.** Let $w = g_0 t^{k_1} g_1 t^{k_2} g_2 \cdots t^{k_n} g_n \in G * \langle t \rangle$. Then w is in *HNN normal form* if for every i, $k_i \neq 0$, $k_i > 0$ implies $g_i \notin A$ and $k_i < 0$ implies $g_i \notin B$.

Note that the HNN normal form is not unique; two words $w, v \in G * \langle t \rangle$ that are equal after mapping into the HNN extension and both in normal form might not be equal as elements of $G * \langle t \rangle$. However, if $w \in G * \langle t \rangle$ maps to 1 in the HNN extension then the following lemma tells us that the unique HNN normal form for w is 1.

Theorem 4.3 (Britton's Lemma)**.** Let $w \in G * \langle t \rangle$. If w is in HNN normal form and w contains a t, then $w \neq 1$ [Mil68]

Corollary 4.3.1. If a word w meets the conditions of Britton's Lemma, then w cannot be written as a t free word.

Proof. Suppose $w = g$ with $g \in G$; then $g^{-1}w$ also meets the conditions in Theorem 4.3, and therefore $gw^{-1} \neq 1$, contradicting $w = g$. □

Given a word $w \in G * \langle t \rangle$, the HNN normalization process replaces any occurrences of ta with $\phi(a)t$ when $a \in A$, and any occurrence of $t^{-1}b$ with $\phi^{-1}(b)t^{-1}$ when $b \in B$. Applying this rewriting procedure will always produce a word w' in HNN normal form and such that w and w' are equal after quotienting by the defining relations of the HNN extension, $\{tat^{-1}\phi(a^{-1})|a \in A\}$.

The HNN normalization process describes an algorithm for deciding whether two words $w, v \in G * \langle t \rangle$ are equal after mapping into the HNN extension, by applying the normalization procedure to wv^{-1}. In order to compute this algorithm, it is also necessary to have an algorithm for checking equality of elements in G, for checking whether an element of G is in either of the subgroups A or B, and for computing ϕ.

5 The Proof Certificate

An element of a group G is equal to 1 in the quotient by the normal closure of a relator r if and only if it can be written as a product of conjugates of r and r^{-1}. More precisely, there is a group homomorphism $Eval : F(G) \to G$ from the free group over G into G that sends a basis element of $F(G)$, $g \in G$, to $grg^{-1} \in G$. The image of this map is exactly the kernel of the quotient map. Therefore an element p of $F(G)$ such that $\mathrm{Eval}(p) = w$ can be seen as a witness that w is in the kernel of the quotient map.

Definition 5.1. (Eval) $\mathrm{Eval}(r)$ is a map $F(G) \to G$, sending a basis element $g \in G$ to grg^{-1}.

The certificate is a pair of a normalised word w' and $p \in F(G)$ such that $w = \mathrm{Eval}(p)w'$.

Definition 5.2 (P functor)**.** For any $g \in G$ define an automorphism $\mathrm{MulFree}(g)$ of $F(G)$ by sending a basis element $h \in G$ to gh. This defines a left action of G on $F(G)$. Define the group $P(G)$ to be

$$P(G) := F(G) \rtimes_{MulFree} G \tag{1}$$

This group has multiplication given by $(a, b)(a', b') = (a\mathrm{MulFree}(b)(a'), bb')$

Definition 5.2.1 (lhs and rhs)**.** Define two group homomorphisms from P into G: let rhs be the obvious map sending (a, b) to b and let lhs be the map sending (a, b) to $\text{Eval}(a)b$. Since $\text{Eval}(a)$ is in the kernel of the quotient map, for any $p \in P(G)$, $\text{lhs}(p)$ and $\text{rhs}(p)$ are equal in the quotient by r. Therefore an element p of $P(G)$ can be regarded as a certificate of the congruence $\text{lhs}(p) \equiv \text{rhs}(p) \bmod r$.

Because both lhs and rhs are group homomorphisms, if $p \in P(G)$ is a certificate of the congruence $a \equiv b \bmod r$ and q is a certificate of the congruence $c \equiv d \bmod r$, then pq is a certificate of the congruence $ac \equiv bd \bmod r$. Similarly, p^{-1} is a certificate of the congruence $a^{-1} \equiv b^{-1} \bmod r$.

Definition 5.2.2. (P is functorial). Given a homomorphism $f : G \to H$, functoriality of the free group gives a natural map $F(f) : F(G) \to F(H)$. Define the map $P(f) : P(G) \to P(H)$ to send $(p, b) \in P(G)$ to $(F(f)(p), f(b)) \in P(H)$. Given a certificate of the congruence $a \equiv b \bmod r$, this map returns a certificate of the congruence $f(a) \equiv f(b) \bmod f(r)$.

Definition 5.2.3. (Trans) Given $p, q \in P(G)$ such that p is a certificate of the congruence $a = b \bmod r$ and q is a certificate of the congruence $b = c \bmod r$, it is possible to define $\text{Trans}(p, q)$ such that $\text{Trans}(p, q)$ is a certificate of the congruence $a = c \bmod r$. If $p = (p_1, p_2)$ and $q = (q_1, q_2)$, then $\text{Trans}(p, q) = (p_1 q_1, q_2)$.

Definition 5.2.4. (Refl) Given $a \in G$, $(1, a)$ is a certificate of the congruence $a = a \bmod r$. Call this $\text{Refl}(a)$.

It is also possible to define *Symm* such that $\text{lhs}(\text{Symm}(p)) = \text{rhs}(p)$ and vice versa, but this is not used in the algorithm.

Definition 5.2.5. (ChangeRel) Given a certificate p of the congruence $a \equiv b \bmod r$, it is possible to make a certificate of the congruence $a \equiv b \bmod grg^{-1}$ for any $g \in G$. For any $g \in G$, let $\phi(g) : F(G) \to F(G)$ be the map sending $h \in G$ to hg. Then $\text{ChangeRel}(g, (p_1, p_2))$ is defined to be $(\phi(g)(p_1), p_2)$ for $g \in G$ and $(p_1, p_2) \in P(G)$.

6 Adding and Removing Subscripts

Given a letter t in the free group over a set S, we can define a map into a semidirect product.

Definition 6.1 (ChangeSubscript)**.** Define a homomorphism ChangeSubscript from $\mathbb{Z}$ to the automorphism group of $F(S \times \mathbb{Z})$. If $(x, n) \in S \times \mathbb{Z}$ is a basis element of the free group, then $\text{ChangeSubscript}(m)(x, n) = (x, m + n)$.

Definition 6.2 (AddSubscripts)**.** There is a homomorphism $\text{AddSubscripts}(t)$ from $F(S)$ into $F(S \times \mathbb{Z}) \rtimes_{\text{ChangeSubscript}} \mathbb{Z}$ sending a basis element $s \in S$ to $(s, 0) \in F(S \times \mathbb{Z})$ when $s \neq t$ and sending t to $(1, 1_{\mathbb{Z}}) \in F(S \times \mathbb{Z}) \rtimes \mathbb{Z}$. Loosely, this map replaces occurrences of $t^n a t^{-n}$ with a_t.

The map AddSubscripts is only used during the algorithm on words w when the sum of the exponents of t in w is zero, meaning the result will always be of the form $(w', 0_{\mathbb{Z}})$.

Definition 6.3 (RemoveSubscripts)**.** RemoveSubscripts sends a basis element of $F(S \times \mathbb{Z})$, $(s, n) \in S \times \mathbb{Z}$, to $t^n s t^{-n}$.

RemoveSubscripts is a group homomorphism and if r is a word such that $\text{AddSubscripts}(r)$ is of the form $(r', 0_{\mathbb{Z}})$, then $\text{RemoveSubscripts}(r') = r$.

7 Magnus' Method

Given an element $w \in F(S)$ of a free group, a relation r in the free group, and a subgroup of the free group generated by a set of letters T, we write $\overline{w}$ for the corresponding element in $F(S)/r$ and $\overline{T}$ for the image of the subgroup generated by T in $F(S)/r$.

The algorithm decides whether $\overline{w} \in F(s)/r$ is in $\overline{T}$, and returns an element w' such that $\overline{w'} = \overline{w}$ and $w' \in T$.

We describe the implementation of a function *Solve* whose arguments are a word w in the free group $F(S)$, a relator $r \in F(S)$, and a subset T of S. If there is a word $w' \in F(S)$ such that $w' \in T$ and $\overline{w} \in F(S)/r$ is equal to $\overline{W'}$, then it returns an element p of $P(F(S))$ such that $\mathrm{lhs}(p) = w$ and $\mathrm{rhs}(p) = w'$. It terminates without returning anything if there is no such word.

Without loss of generality we can assume r is cyclically reduced and conjugate r if this is not the case. We can use ChangeRel, to make the correct proof certificate after conjugating r.

7.1 Case 1: All letters in r are in T

For this case it is helpful to consider the group $F(S)$ as the coproduct of the subgroup generated by the letters in T and the subgroup generated by the rest of the letters: $F(S) \cong F(T) * F(S \backslash T)$.

Since every letter in r is also in T then $F(S)/r \cong F(T)/r * F(S \backslash T)$. An element of $F(S)$ can therefore be written in the form $w_0 v_0 w_1 v_1 \dots w_n v_n$, where $w_i \in F(T)$ and $v_i \in F(S \backslash T)$. The problem can be reduced to deciding whether an element of $w_i \in F(T)$ is equal to an element of the quotient. To decide the word problem whenever $\overline{w_i} = 1$, perform this substitution and then check whether the resulting word in $F(T) * F(S \backslash T)$ is in T.

7.2 Case 2: There is a letter in r that is not in T

7.2.1 Base Case

The base case is the case where the relation r is of the form a^n with $n \in \mathbb{Z}$, and a a letter in S. It is straightforward to decide the word problem in this group, since $F(S)/a^n$ is isomorphic to the binary coproduct of $F(S \backslash \{a\})$ and $\mathbb{Z}/n\mathbb{Z}$.

7.2.2 Case 2a: Letter with exponent sum zero

Apply the map AddSubscripts(t) (Definition 6.2) to r. Since the exponent sum of t is equal to zero, AddSubscripts(t)(r) is of the form $(r', 0_{\mathbb{Z}})$. The length (Definition 3.1) of the relation $r' \in F(S \times \mathbb{Z})$ is less than the length of r. If $t \notin T$ and the exponent sum of t in w is not zero, then $\overline{w} \notin \overline{T}$. If $t \in T$, then w can be written in the form $w' t^n$ where t has exponent sum zero in w', and w' is a word in T if and only if $\overline{w} \in \overline{T}$.

A naive approach would be to apply AddSubscripts(t) to w, and solve the word problem in $F(S \times \mathbb{Z})$ with respect to r'. However, the image of the normal closure of r' under AddSubscripts(t) restricted to $F(S \times \mathbb{Z})$ is not the normal closure of r'; it is the normal closure of the set of all relations of the form ChangeSubscript(n)(r') for every n.

Pick $x \in S$ such that $x \neq t$, x is a letter in r and such that $t \in T$ implies $x \notin T$. If this is not possible, then apply the procedure in Section 7.1. We can assume that the first letter of r is x, since otherwise r can be conjugated until the first letter is x. Let a and b be respectively the smallest and greatest subscript of x in r'. Let S' be the set $\{(i_1, i_2) \in S\backslash\{t\} \times \mathbb{Z} \mid i_1 \neq x \vee a \leq i_2 \leq b\}$.

Define two subsets of S' by $A := S'\backslash\{x_b\}$ and $B := S'\backslash\{x_a\}$. Then there is an isomorphism ϕ between A and B given by $\phi :=$ ChangeSubscript(1). We claim the group $F(S)/r$ is isomorphic to the HNN extension of $F(S')/r'$ relative to ϕ.

The homomorphism α from $F(S)/r$ to the HNN extension sends a letter $s \in S\backslash\{t\}$ to s_0 and the letter t to the stable letter t of the HNN extension. Since $ts_it^{-1} = s_{i+1}$ in the HNN extension for $s_i \in S'$, r is sent to r' by this map so that α is well defined on the quotient.

Now let β send $s_i \in S'$ to $t^i s t^{-i}$ and the stable letter t to t. Again, r' is sent to r by β, and $\beta(ts_it^{-1}) = t^{i+1}st^{-(i+1)} = \beta(\phi(s_i))$ so β preserves the defining relations of the HNN extension and it is well defined. It can be checked β is a two sided inverse to α, and thus α is an isomorphism.

We then apply the HNN normalization procedure, which will be described in detail in Section 7.3. We chose x and t such that either $x \notin T$ or $t \notin T$.

In either case, if $\overline{w}$ can be written as a word in T, then an HNN normal form of w will be of the form gt^n with $g \in F(S')/r'$. In the case $x \notin T$, then because any word in $F(S')$ not containing x_i must be in $A \cap B$, there can be no occurrence of tg with $g \notin A$ or $t^{-1}g$. If $t \notin T$, then it must be possible to write w without t, so in fact it can be normalized to $g \in F(S')/r'$. We can check whether any words in $F(S')/r'$ are in the subgroups generated by A or B using Magnus' method again for the shorter relation r', and rewrite these words using the letters in A or B when possible.

Once in the form gt^n with $g \in F(S')$, it is enough to check that $\overline{\text{RemoveSubscripts}(g)}$ can be written as a word in T. If $t \in T$ then this amounts to solving the word problem for r' and the set $T' := \{s_i \in S' | s \in T, i \in \mathbb{Z}\}$. If $t \notin T$, this amounts to checking that $n = 0$ and solving the word problem for r' and the set $T' := \{s_0 \in S' | s \in T\}$.

Case 2.b: No letter with exponent sum zero

If there is no letter t in r with exponent sum zero, then choose x and t such that $x \neq t$ and such that if $t \notin T$ then $x \notin T$. Let α be the exponent sum of t in r and let β be the exponent sum of x.

Then define the map ψ on $F(S)$ by

$$\psi(s) = \begin{cases} t^\beta & \text{if } s = t \\ xt^{-\alpha} & \text{if } s = x \\ s & \text{otherwise} \end{cases} \tag{2}$$

The map ψ descends to a map $\overline{\psi} : F(S)/r \to F(s)/\psi(r)$. The map ψ is equal to $\psi_1 \circ \psi_2$, where ψ_2 and ψ_1 are defined as follows:

$$\psi_1(s) = \begin{cases} xt^{-\alpha} & \text{if } s = x \\ s & \text{otherwise.} \end{cases} \tag{3}$$

$$\psi_2(s) = \begin{cases} t^{\beta} & \text{if } s = t \\ s & \text{otherwise} \end{cases} \tag{4}$$

We have that $\overline{\psi_1} : F(S)/r \to F(r)/\psi_1(r)$ is an isomorphism, with inverse given by sending x to xt^{α}. Meanwhile, $\overline{\psi_2} : F(S)/r$ to $F(r)/\psi_2(r)$ is also injective. This is proven constructively in Section 7.9. Hence $\overline{\psi} : F(S)/r$ to $F(r)/\psi(r)$ is injective.

The image of the subgroup generated by T under ψ might not be the subgroup generated by a set of letters, but it is always contained in T. By the Freiheitsatz, if $\overline{\psi(w)}$ can be written as a word w' using letters in T then this solution is unique. Therefore, to check if $\overline{\psi(w)}$ is in the subgroup generated by $\overline{\psi(T)}$, one can first write it as a word in $w' \in T$ if possible, and then check if w' is in $\psi(T)$. The exponent sum of t in $\psi(r)$ is 0, so the problem of checking if $\psi(w)$ can be written as a word in T can be solved using the method described in Section 7.2.2.

If $t \in T$ then $\psi(T)$ is generated by $T' := T \backslash \{t\} \cup t^{\beta}$. By the Freiheitsatz, if $\psi(w)$ can be written as a word w' using letters in T then this solution is unique. Therefore, to check if $\psi(w)$ is in the subgroup generated by T', one can first write it as a word in w' in T if possible, and then check that for every occurrence of t^k in w', k is a multiple of α.

If $t \notin T$, then $\psi(T) = T$.

7.3 HNN normalization

We first present a simplified version of the HNN normalization that does not compute the proof certificates, and then explain how to compute the certficates at the same time as normalization.

To compute the HNN normalized term, first compute the following isomorphism from $F(S)$ into the binary coproduct $F(S') * \langle t \rangle$, where $\langle t \rangle$ is an infinite cyclic group generated by t.

Definition 7.1. Define a map on a basis element i as follows

$$\begin{cases} i_0 \in S' & i \neq t \\ t & i = t \end{cases} \tag{5}$$

It is important that $a \leq 0 \leq b$, to ensure that the image of this map is contained in $F(S' \times \langle t \rangle)$.

Then apply the HNN normalization procedure. For this particular HNN extension ϕ is *Change-Subscript* (Definition 6.1). We work in the $F(S') * \langle t \rangle$, and apply the following rewriting rules.

For each occurrence of tw where there is an $a \in A$ such that $\overline{a} = \overline{w}$ replace tw with $\phi(a)t$.

For each occurrence of $t^{-1}w$ where there is an $b \in B$ such that $\overline{b} = \overline{w}$ replace tw with $\phi^{-1}(b)t^{-1}$.

We can use *Solve* to check whether there is such a and b with these properties.

7.3.1 Computing Proof Certificates

To compute proof certificates a slight modification of the procedure described in Section 7.3 is used.

First define a modification of Definition 7.1, from $F(S)$ into the binary coproduct $P(F(S \times \mathbb{Z})) * \langle t \rangle$.

Definition 7.2. Define a map on the basis as follows

$$\begin{cases} \text{Refl}(i, t^0) \in F(S' \times \langle t \rangle) & i \in S \text{ and } i \neq t \\ t & i = t \end{cases} \tag{6}$$

There is also a map Z from $P(F(S \times \mathbb{Z})) * \langle t \rangle$ into $P(F(S))$. This map is not computed as part of the algorithm, but is useful to define anyway.

Definition 7.3. The map Z sends $t' \in \langle t \rangle$ to $\text{Refl}(t) \in P(F(S))$. It sends $p \in P(F(S \times \mathbb{Z}))$ to $P(\text{RemoveSubscripts})(p) \in P(F(S))$

The aim is to define a normalization process into that turns a word $w \in F(S)$ into word $n \in P(F(S \times \mathbb{Z})) * \langle t \rangle$ such that after applying *rhs*, the same word is returned as in the normalization process described in Section 7.3. We also want $\text{lhs}(Z(n))$ to be equal to w, so we end up with a certificate that w is equal to some normalized word.

Definition 7.4. (conjP) Let $(p, a) \in P(F(S \times \mathbb{Z}))$ and $k \in \mathbb{Z}$. Define *ConjP* to map into $P(F(S \times \mathbb{Z}))$

$$\text{ConjP}(k, (p, a)) = (\text{MulFree}((t, 0)^k, p), \text{ChangeSubscript}(k, a)) \tag{7}$$

conjP has the property that $\text{lhs}(Z(\text{conjP}(k, p))) = t^k \text{lhs}(Z(p)) t^{-k}$, and similarly for *rhs*. Note that *conjP* maps into $P(F(S \times \mathbb{Z}))$ and not $P(F(S'))$, although *rhs* of every word computed will be in $F(S')$.

The procedure described in Section 7.3 replaced each occurrence of wt^{-1} with $t^{-1}\text{ChangeSubscript}(-1)(a)$, where $a \in A$ was a word equal to $w \in F(S')$ in the quotient $F(S')/r'$.

To compute the certificates apply the following rewriting procedure: for each occurrence of tp where $\overline{\text{rhs}(p)} = \overline{a}$ for some $a \in A$, and q is a certificate of this equality, replace tp with $\text{ConjP}(1, \text{Trans}(p, q))t$

Similarly, for each occurrence of $t^{-1}p$ where $\overline{\text{rhs}(p)} = \overline{b}$ for some $b \in B$ and q is a certificate of this equality, replace $t^{-1}p$ with $\text{ConjP}(-1, \text{Trans}(p, q))t^{-1}$

7.3.2 Performance

The order in which the rewriting rules are applied can have a big effect on the performance of the algorithm.

Example 7.4.1. Suppose $r' = x_1 {x_0}^{-2}$ and $w = t^n x_1 t x_0^{-1}$, where $n > 0$. Then S' is the set $\{x_0, x_1\}$, A is the subgroup generated by $S' \backslash x_1$ and B the subgroup generated by $S' \backslash x_0$.

Suppose we first make the substitution tx_0^{-1} to $x_1^{-1}t$; then w becomes $t^n x_1 x_1^{-1} t = t^{n+1}$. This is in HNN normal form.

Now consider trying the HNN normalization process from the left. For any $m \in \mathbb{Z}$, $x_1^m = x_0^{2m}$, so the HNN normalization proces will rewrite $tx_1^m t o x_1^{2m} t$. Therefore $t^n x_1$ will be rewritten to $x_1^{2^n} t^n$. Hence w gets rewritten to $x_1^{2^n} t^{n+1} x_0^{-1}$, which then will eventually be rewritten to t^{n+1}. The maximum length of w during the normalization process was greater than 2^n.

Applying one rewrite rule first might mean that another rewrite is unnecessary, or a call to *Solve* is given an easier problem.

Example 7.4.2. Consider the word $tw_0t^{-1}w_1$ with $w_0, w_1 \in F(S')$. In this situation it is best to start by attempting to prove $\overline{w_0} \in \overline{A}$ in the quotient. Applying the left hand rewrite first will put the word into HNN normal form straight away; it will not be necessary to check $\overline{w_1} \in \overline{B}$.

Rewriting starting on the right first might give a word such as $tw_0\phi^{-1}(b)t^{-1}$, where $\overline{b} = \overline{w_1}$. But since $\phi^{-1}(b) \in A$, checking whether $\overline{w_0\phi^{-1}(b)} \in \overline{A}$ is no easier than checking $\overline{w_0} \in \overline{A}$ has not become any easier. So in this example it is better to start rewriting on the left, and furthermore, if $\overline{w_0} \notin \overline{A}$ then it will not be possible to eliminate the t's, so the algorithm can fail straight away without attempting more rewrites.

Example 7.4.3. unknown Consider the word $tw_0t^{-1}w_1tw_2$. Here it is not possible to determine the best order without considering what particular words w_0, w_1, w_2 are. Starting in the middle gives the word $tw_0\phi(b)w_2$, and since w_2 might not be in B, the first problem may have become easier.

However starting on the left with the pair tw_0 might also be the best thing to do, since starting on the left would make the second problem easier, giving the word $\phi()bw_1tw_2$.

Example 7.4.4. Consider the word tw_0tw_1. Here it is best to apply the right hand rewrite first. Applying the left hand rewrite first will not make the right hand one any easier; the t's will not cancel, but applying the right hand one first could make the left hand problem easier. After applying the right hand rewrite, the word would become $tw_0\phi(a)t$, where $a \in A$ and $\overline{a} = \overline{w_1}$. It is possible that it is easier to check $\overline{w_0\phi(a)} \in \overline{A}$ than to check both $\overline{w_0} \in \overline{A}$ and $\overline{phi}(a) \in \overline{A}$.

Example 7.4.2 and Example 7.4.4 give an optimal normalization order for simple examples, with only two occurrences of a power of t. It is not possible to generalize this to more complicated examples, but some sensible heuristics are possible.

Consider a word $W := w_0t^{n_1}w_1 \dots t^{n_{k-1}}w_{k-1}t^{n_k}w_k$, and the following two sets

$$R^+ := \left\{ i | \forall j, \sum_0^i n_i \leq \sum_0^j n_j \right\}$$
$$R^- := \left\{ i | \forall j, \sum_0^i n_i \geq \sum_0^j n_j \right\}$$

Let I be the smaller of the two intervals $[\min R^+, \max R^+]$ and $[\min R^-, \max R^-]$.

Within the interval I, let Q be the set of pairs $t^{n_i}w_i$ such that $\text{sgn}(n_i) \neq \text{sgn}(n_{i+1})$.

Then Q has the property that if there are no applicable rewrites in the set Q, then the word cannot be put into the form wt^n with $w \in F(S')$.

The rewriting procedure always chooses a rewrite within the set Q, prioritising pairs $w_i t^{n_i}$ where w_i has the least occurences of letters not in the set T, (likely to be the fastest problems to solve). This heuristic mitigates the exponential behaviour described in Example 7.4.1

7.4 Heuristics

A few heuristics are implemented when there is a simpler method than Magnus' method. They are implemented in the following order

- Check whether the word w is already written using letters in T. If $w \in T$, then trivially $\overline{w} \in \overline{T}$
- If there is a letter x in the relator r such that $x \notin T$, but x is in w then by the Freiheitsatz, $\overline{w} \notin \overline{T}$, so the algorithm can fail straight away.
- If w is not in the subgroup generated by r and T after abelianizing the free group, the algorithm can fail straight away.
- If the relation r has exactly one occurrence of a letter, say x, then the problem can be solved by rearranging the equation $r = 1$ to the form $x = v$, where v is a word not containing x, and making this substitution everywhere in w.

7.5 Injectivity

The correctness of the algorithm relies on the fact that the map ψ_2 is an injective map. Since ψ_2 is injective, if $p \in P(F(S))$ is a witness of the congruence $\psi_2(a) \equiv \psi_2(b) \bmod \psi_2(r)$, then there must exist a certificate q of the congruence $a = b \bmod r$. The question is how to compute this. The proof of this congruence given in [Put20] relies on the fact that the canonical maps into an amalgamated product of groups are injective. However the standard proof seems to rely on the law of the excluded middle, so it cannot be translated into an algorithm to compute q.

Suppose $p \in P(F(S))$ is a witness of the congruence $\psi_2(a) = \psi_2(b) \bmod \psi_2(r)$. It is not necessarily the case that k is a multiple of n in every occurrence of t^k in p. For example $p := ([t][tr^{-1}t^{-1}][t]^{-1}, 1) \in P(F(S))$ is a witness of the congruence $r = 1$. Both $\mathrm{lhs}(p)$ and $\mathrm{rhs}(p)$ are in the image of ψ_2 for $n = 2$, when r is in the image of ψ_2, but p is not in the image of $P(\psi_2)$. However where there are occurrences of t, they are all cancelled after lhs is applied, in fact one could remove every occurrence of t from p and still have a certificate of the same congruence.

Definition 7.5. Given a word $w \in F(S)$, define the set of partial exponent sums of a letter $t \in S$ to be the set of exponent sums of all the initial words of w. For example, the partial exponent sums of t in $t^n a t$ are the exponent sums of t in t^n, $t^n a$ and $t^n a t$.

Definition 7.6. h is a map $F(S) \to F(S \cup \{t'\})$, where t' is some letter not in S. h replaces every occurrence of t^k with $t'^a t^b$ in such a way that $a + nb = k$, and every partial exponent sum of t' in $h(w)$ is either not a multiple of n, or it is zero.

Definition 7.7. θ is a group homomorphism $F(S \cup \{t'\})$. Let $s \in S$. Then

$$\theta(s) = \begin{cases} t & \text{if } s = t' \\ t^n & \text{if } s = t \\ s & otherwise \end{cases} \tag{8}$$

θ and h satisfy $\theta \circ h = \mathrm{id}$. For any w in $F(S)$, $\theta(w) = \psi_2(w)$.

Definition 7.8. (PowProof) *PowProof* is a map $F(S) \to F(S)$. PowProof(w) is defined to be $h(w)$, but with every occurrence of t' replaced with 1.

Theorem 7.9. For any $p \in F(F(S))$ if
$\mathrm{Eval}(\psi_2(r))(p) = \psi_2(w)$, then $\mathrm{Eval}(r)(F(\mathrm{PowProof})(p)) = w$.

Lemma 7.9.1. Consider $\prod_{i=1}^{a} s_i^{k_i}$, as an element of the $F(S \cup \{t'\})$ with $s_i \in S \cup \{t'\}$ (Note that this is not necessarily a reduced word; k_i may be zero and s_i may be equal to s_{i+1}). Suppose every partial product $\prod_{i=1}^{b} s_i^{k_i}$, with $b \leq a$ has the property that if the exponent sum of t' is a multiple of n, then it is zero. Suppose also that $\prod_{i=1}^{b} s_i^{k_i}$ has the property that for every occurrence of t'^k in the reduced product, k is a multiple of n. Then the reduced word $\prod_{i=1}^{a} s_i^{k_i}$ can be written without an occurrence of t'.

Proof of Lemma 7.9.1. $\prod_{i=0}^{a} s_i^{k_i}$ can be written as a reduced word $\prod_{i=1}^{c} u_i^{k'_i}$ such that k'_i is never equal to zero and $u_i \neq u_{i+1}$ for any i. The set of partial products of this $\prod_{i=1}^{c} u_i^{k'_i}$ is a subset of the set of partial products of $\prod_{i=1}^{a} s_i^{k_i}$, therefore the exponent sum of t' in every partial product of $\prod_{i=1}^{a} s_i^{k_i}$, is either 0 or not a multiple of n. However, by assumption every occurrence t'^k in $\prod_{i=0}^{a} s_i^{k_i}$, k, is a multiple of n, so the exponent sum of t' in every partial product is 0. So $\prod_{i=1}^{a} s_i^{k_i}$ does not contain t'.

Proof of Theorem 7.9
If $\mathrm{Eval}(\psi_2(r))(p)$ is in the image of ψ_2, then $\mathrm{Eval}(r)(F(h)(p))$ has the property that for every occurrence of t'^k, k is a multiple of n. If $p' := F(h)(p)$, then $\mathrm{Eval}(r)(p')$ can be written as a product of the form in Lemma 7.9.1. If $r' = \prod_i u_i^{l_i}$, then to write $\mathrm{Eval}(r)(p')$ in this form, send $\prod_i \left[\prod_{j=1}^{a} s_{ij}^{k_j}\right] \in P(F(S \cup \{t'\}))$, to

$$\prod_i \left(\left(\prod_{j=1}^{a} {s_{ij}}^{k_j} \right) \left(\prod_j u_j^{l_j} \right) \left(\prod_{j=1}^{a} {s_{i(a-j)}}^{-k_{a-j}} \right) \right) \tag{9}$$

If all the nested products in Equation 9 are appended into one long product, then the product has the form in Lemma 7.9.1. Therefore when the word is reduced it will not contain t' by Lemma 7.9.1. This means that deleting all occurrences of t' will in p' will not change $\mathrm{Eval}(r)(p')$, and therefore $\mathrm{Eval}(r)(F(\mathrm{PowProof})(p)) = \mathrm{Eval}(r)(F(h)(p))$. Applying ψ_2 to both sides gives $\psi_2(\mathrm{Eval}(r)(F(\mathrm{PowProof})(p)) = \psi_2(\mathrm{Eval}(r)(F(h)(p))) = \theta(\mathrm{Eval}(r)(F(h)(p))) = \mathrm{Eval}(\psi_2(r))(p)$.

8 Efficiency of the Algorithm

The worst case performance of this algorithm is worse than any finite tower of exponents [MUW11]. The more relevant question is what is the typical performance.

Definition 8.1 (Area of a Relation)**.** For a finitely presented group $G := \langle S|R\rangle$, if $w \in F(S)$ is equal to 1 in the quotient G, then we say it is a *relation*. The *area of a relation* is the smallest N such that w can be written in the form $\prod_{i=1}^{N} g_i r_i^{\epsilon_i} g_i^{-1}$, where $\epsilon_i = \pm 1$ and $r_i \in R$ for all i.

Definition 8.2 (Dehn Function)**.** For a finitely presented group $G := \langle S|R\rangle$, the Dehn function of the presentation $\mathrm{Dehn}(n) \in \mathbb{N}$ is defined as the largest area of a relation of length at most n.

The Dehn function puts a lower bound on the complexity of the one-relator algorithm. The area of a relation is by definition the length of the shortest certificate that the algorithm might produce, so the complexity of the algorithm is bounded above by the Dehn function of a relator. The group $\langle a, b | bab^{-1}aba^{-1}b^{-1} = a^2 \rangle$ is such that $\text{Dehn}(n)$ is worse than any finite tower of exponents. This means that the complexity of the one relator algorithm is also worse than any finite tower of exponents.

Not all groups have such a fast growing Dehn function. For example, if the relator is of the form r^k with $|k| \neq 1$ then $\text{Dehn}(n) \leq n$. Similarly, even in groups with a rapidly increasing Dehn function, there are words that do not have a large area as the worst case.

So, even though the worst case behaviour is very bad, there are still potentially many problems that the algorithm could solve in a practical amount of time. The aim of this implementation was to have good performance on relations with a small area. A typical Lean tactic state will usually be used on problems where the author knows the solution, but simply needs automation to write a formal proof of the solution. These relations will usually have a very small area. The aim of this implementation was that the algorithm should have good performance on relations with a small area, but makes no attempt to solve problems where the area of the relation is very large.

References

[dMKA+18] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer, The lean theorem prover (system description), Jun 2018.

[mC20] The mathlib Community, The lean mathematical library, Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (2020).

[Mil68] Charles F. Miller, On Britton's theorem A, Proceedings of the American Mathematical Society **19** (1968), no. 5, 1151–1154 (eng).

[MS73] James McCool and Paul E. Schupp, On one relator groups and hnn extensions, Journal of the Australian Mathematical Society **16** (1973), no. 2, 249–256.

[MUW11] Alexei Miasnikov, Alexander Ushakov, and Dong Wook Won, The word problem in the baumslag group with a non-elementary dehn function is polynomial time decidable, 2011.

[Put20] Andrew Putman, One relator groups.