1 Efficiency of the Algorithm

The worst case performance of this algorithm is worse than any finite tower of exponents (cite something). The more relevant question is what is the typical performance.

Definition 1.1 (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 1.2 (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 $\mathrm{Dehn}(n)$ 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 $\mathrm{Dehn}(n) \leq n$. 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.

2 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 if 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 occurence 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 2.1 (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.

3 HNN Extensions

Definition 3.1 (HNN Extension)**.** Given a group G and 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 a multiplicative group isomorphic to $\mathbb{Z}$, 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 3.2 (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 or 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 3.3 (Britton's Lemma)**.** Let $w \in G * \langle t \rangle$. If w is in HNN normal form then w and w contains a t, then $w \neq 1$ cite (On Britton's theorem Charles Miller)

Corollary 3.3.1. If a word w meets the same conditions as in the statement of Britton's Lemma, then w cannot be written as a t free word.

Proof of Corollary 3.3.1 Suppose $w = g$ with $g \in G$, then $g^{-1}w$ also meets the conditions in Theorem 3.3, and therefore $gw^{-1} \neq 1$, contradicting $w = g$.

The HNN normalization process given a word $w \in G * \langle t \rangle$ replaces any occurences of ta with $\phi(a)t$ when $a \in A$, and any occurance 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, 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 equality if two words $w, v \in G * \langle t \rangle$ are equal after mapping into the HNN extension. You can check is wv^{-1} maps to 1 in the HNN extension by applying the rewriting procedure to wv^{-1}. In order to compute this algorithm, it is necessary to also have an algorithm for checking equality of elements of G and to be able to check whether an element of G is in either of the subgroups A or B, and to be able to compute ϕ.

4 The Proof Certificate

An element of a group G is equal to 1 in the quotient by the normal closure of a relation 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 $\text{Eval}(p) = w$ can be seen as a witness that w is in the kernel of the quotient map.

Definition 4.1. (Eval) $\text{Eval}(r)$ is a map $F(G) \to G$, sending a basis element $g \in G$ to grg^{-1}.

Definition 4.2 (P functor)**.** Define a group structure on this set of pairs. For any $g \in G$ define an automorphism $MulFree(g)$ of $F(G)$, by sending a basis element $h \in G$ to gh. This defines a left action of G on $F(G)$. The group structure is given by the semidirect product. Define the group $P(G)$ to be

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

This group has multiplication given by $(a, b)(a', b') = (aMulFree(b)(a'), bb')$

Definition 4.2.1 (lhs and rhs)**.** Define two group homomorphisms from P into G. rhs is the obvious map sending (a, b) to b. lhs is 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 4.2.2. (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 4.2.3. (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 $b = c \bmod r$, then 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 $\text{Trans}(p, q) = (p_1q_1, q_2)$.

Definition 4.2.4. (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 4.2.5. (ChangeRel) Given a certificate p of the congruence $a \equiv b \bmod r$, then it is possible to make a certificate of the congruence $a \equiv b \bmod grg^{-1}$ for any $g \in G$. Let for any $g \in G$ let $\phi(g) : F(G) \to F(G)$ be the map sending $h \in G$ to hg. The $\text{ChangeRel}(g, (p_1, p_2))$ is defined to be $(\phi(g)(p_1), p_2)$ for $g \in G$ and $(p_1, p_2) \in P(G)$.

5 Magnus' Method

I describe an algorithm to check whether an element w of a one relator group is in the subgroup generated by a set of letters. It also writes w as an element in terms as a word using only those letters.

5.1 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 $\mathbb{Z}/n\mathbb{Z}$.

5.2 Case 1: Letter with exponent sum zero

There are two cases to consider, the first case is when there is a letter t with exponent sum equal to zero in r.

For this case apply the map AddSubscripts(t) (Definition 5.2) 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 2.1) of the relation $r' \in F(S \times \mathbb{Z})$ is less then the length of r. If $t \notin T$ and the exponent sum of t in w is not zero, then w can not be written as a word using letters in 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 w is a word in 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, this approach does not work because 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.

We can assume r is *cyclically reduced* (ref Freiheitsatz), meaning the first and last letter of r are different, and conjugate r if this is not the case.

Pick $x \in S$ such that $x \neq t$ and if $t \in T$ then $x \notin T$. x must also be a letter in r. We can assume that the first letter of r is x, since if it is not it can be 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 $\{i \in S\backslash\{t\} \times \mathbb{Z} \mid i \neq x \vee a \leq i \leq b\}$.

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

The isomorphism α from $F(S)$ 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 $r' = 1$ by this map so α is well defined on the quotient.

β sends $s_i \in S'$ to t^ist^{-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 so α is an isomorphism.

We then apply the HNN normalization procedure, described in detail in Section 5.5. We chose x and t such that either $x \notin T$ or $t \notin T$. In either case if 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 occurence 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 $g \in F(S')/r'$, it is enough to check that RemoveSubscript(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 $T' := \{s_0 \in S' | s \in T\}$.

5.3 Case 2: No Letter with exponent sum zero

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

Then define the map ψ on $F(S)$ defined for $s \in S$ by

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

The map ψ can be descended 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.

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

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

$\overline{\psi_1} : F(S)/r \to F(r)/\psi_1(r)$ is an isomorphism, the inverse given by sending y to yt^{α}. $\overline{\psi_2} : F(S)/r$ to $F(r)/\psi_2(r)$ is also injective. This is proven constructively in Section 5.13. So $\overline{\psi} : F(S)/r$ 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 the subgroup generated by T. 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 $\psi(T)$, you can first write it as a word in $w' \in \langle T \rangle$ 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 5.2.

If $t \in T$ then the subgroup generated by $\psi(T)$ 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', you can first write it as a word in $w' inT$ if possible, and then check that for every occurence of t^k in w', k is a multiple of α.

If $t \notin T$, then the subgroup generated by $\psi(T)$ is T.

5.4 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 5.1 (ChangeSubscript)**.** Define a homomorphism *ChangeSubscript* from $\mathbb{Z}$ to the automorphism group of $F(S \times \mathbb{Z})$. If $(x, n) \in S \times mathbbZ$ is a basis element of the free group, then AddBasis$(m)(x, n) = (x, m + n)$.

Definition 5.2 (AddSubscripts)**.** There is a homomorphism $AddSubscripts(t)$ 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 replace occurence 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 5.3 (RemoveSubscripts)**.** *RemoveSubscripts* send 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 of Addsubscripts(r) is of the form $(r', 0_{\mathbb{Z}})$, then RemoveSubscripts(r') $= r$.

5.5 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 $F(S') * \langle t' \rangle$. $\langle t' \rangle$ is a cyclic multiplicative group isomorphic to $\mathbb{Z}$ and generated by t'.

Definition 5.4. 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 this map does map into $F(S' \times \langle t' \rangle)$.

Then apply the HNN normalization procedure. For this particular HNN extension ϕ is *Change-Subscript* For each occurence of $t'w$, we can use *Solve* to check whether w is equal to a word $a \in A$ in the quotient $F(S')/r'$, and if it is equal to some word a, rewrite $t'w$ to Changesubscript$(1)(a)t'$. Similarly, For each occurence of wt'^{-1}, use *Solve* to check whether w is equal to a word $b \in B$ in the quotient $F(S')/r'$, and if it it equal to some word b rewrite t'^{-1} to ChangeSubscript$(-1)(b)t'^{-1}$.

5.5.1 Computing Proof Certificates

To compute proof certificates a slightly modification of the procedure described in Section 5.5 is used.

First define a modification of Definition 5.4, from $F(S)$ into the binary coproduct $P(F(S \times \mathbb{Z})) * \langle t' \rangle$.

Definition 5.5. 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 5.6. The map Z sends $t'\in\langle t'\rangle$ to $\mathrm{Refl}(t)\in P(F(S))$. It sends $p\in P(F(S\times\mathbb{Z}))$ to $P(\mathrm{RemoveSubscripts})(p)\in P(F(S))$

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 5.5. We also want $\mathrm{lhs}(Z(n))$ to be equal to w, so we end up with a certificate that w is equal to some normalized word.

Definition 5.7. (conjP) Let $(p,a)\in P(F(S\times\mathbb{Z}))$ and $k\in\mathbb{Z}$. Define *ConjP* to map into $P(F(S\times\mathbb{Z}))$

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

conjP has the property that $\mathrm{lhs}(Z(\mathrm{conjP}(k,p)))=t^k\mathrm{lhs}(Z(p))t^{-k}$, 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 5.5 replaced each occurence of wt'^{-1} with $t'^{-1}\mathrm{ChangeSubscript}(-1)(a)$, where $a\in A$ was a word equal to $w\in F(S')$ in the quotient $F(S')/r'$.

To compute the proof certificate suppose there is an occurence of pt'^{-1} with $p\in P(F(S\times\mathbb{Z}))$. Use *Solve* to check whether $\mathrm{rhs}(p)$ is equal to a word $a\in A$. Suppose $q\in P(F(S'))$ is the certificate of this congruence. Then substitute pt'^{-1} with $t'^{-1}\mathrm{ConjP}(1,\mathrm{Trans}(p,q))$.

Similarly replace for every occurence of pt', with $\mathrm{rhs}(p)$ equal to some word $b\in B$, and q a certificate of this congruence, replace pt' with $t'\mathrm{ConjP}(-1,\mathrm{Trans}(p,q))$.

5.5.2 Performance

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

Example 5.7.1. As an example, suppose $r'=x_1y_1x_0{y_0}^{-2}$, and $w=t^nx_1y_1ty_0^{-1}x_0^{-1}$, where $n>0$. Then S' is the set $\{x_0,x_1\}\cup\{y_i|i\in\mathbb{Z}\}$, A is the subgroup generated by $S'\backslash x_1$ and B the subgroup generated by $S'\backslash x_0$. Suppose I first make the substitution $ty_0^{-1}x_0^{-1}$ to $y_1^{-1}x_1^{-1}t$, then w becomes $t^nx_1y_1y_1^{-1}x_1^{-1}t=t^{n+1}$. This is in HNN normal form.

Now consider trying the HNN normalization process from the left. For any $m\in\mathbb{Z}$, $(x_1y_1)^m=(x_0y_0)^{2m}$, so the HNN normalization proces will rewrite $t(x_1y_1)^m to (x_1y_1)^{2m}t$. Therefore $t^nx_1y_1$ will be rewritten to $(x_1y_1)^{2^n}t^n$. So w gets rewritten to $(x_1y_1)^{2^n}t^{n+1}y_0^{-1}x_0^{-1}$, which then will eventually be rewritten to t^{n+1}. The maximum length of w during the normalization process became 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 5.7.2. 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 5.7.3. 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 $\overline{phi}(a) \in \overline{A}$. In Example 5.7.1, $w_0\phi(a) = 1$, which is clearly much easier.

Example 5.7.2 and Example 5.7.3 give an optimal normalization order for simple examples. It is possible to generalize this to more complicated examples.

Theorem 5.8. Consider a word $W := w_0t^{n_1}w_1 \dots t^{n_{k-1}}w_{k-1}t^{n_k}w_k$, where $n_1 > 0$. Let $1 \leq a \leq k$ be such that for any b in the same range, $\sum_{i=1}^{a} n_i \geq \sum_{i=1}^{b} n_i$, and such that for any b such that $\sum_{i=1}^{a} n_i = \sum_{i=1}^{b} n_i$, then $a \leq b$. Then W cannot be put into the form gt^n with $g \in F(S')$ by the HNN rewriting rules unless $\overline{w_a} \in \overline{A}$.

There is a similar theorem when $n_1 < 0$.

Proof of Theorem 5.8. There are two ways that W may be normalized without $\overline{w_a} \in \overline{A}$. Either the problem is simplified from the left, and the t is cancelled by being rewritten on the left, or after some rewrites to the right of w_a, w_a ends up multiplied by $\phi(a)$, and where $\overline{w_a\phi(a)} \in \overline{A}$.

In order for the pair $t^{n_k}w_k$ to be simplified from the left, W must first be rewritten to $W' := w_0't^{n_1'}w_1' \dots t^{n_{a-1}'}w_{a-1}t^{n_a}w_a \dots t^{n_{k-1}'}w_{k-1}'t^{n_k'}w_k'$ where $n_{a-1}' < -n_a$. But then $\sum_{i=1}^{a-2} n_i' \geq \sum_{i=1}^{a} n_i'$, contradicting the condition on a (This condition is preserved by applying rewriting rules to pairs other than $t^{n_a}w_a$).

In order for the pair $t^{n_k}w_k$ to be simplified from the right, W must first be rewritten to $W' := w_0't^{n_1'}w_1' \dots t^{n_a}w_at^{n_{a+1}'}w_{a'+1} \dots t^{n_{k-1}'}w_{k-1}'t^{n_k'}w_k'$ where $n_{a+1} > 0$. But then $\sum_{i=1}^{a+1} n_i' > \sum_{i=1}^{a} n_i'$ again contradicting the condition on a.

Theorem 5.8 gives a precise rewriting order that avoids the undesirable inefficiencies in Example 5.7.1 and Example 5.7.2, the pair $t^{n_a}w_a$ that satisfies the condition in this theorem should be rewritten first. Rewriting using this order also has the advantage that provided the starting problem is true, *Solve* only recursively calls itself on problems that are also true, which is an obvious performance benefit, and this means that *Solve* can fail early as soon as it recursively calls itself on a problem that is not true.

5.6 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 relies on the fact that the canonical maps into an amalgamated product of groups are injective. However this proofs relies on the law of the excluded middle, so it cannot be translated into an algorithm to compute q. (Cite proof of amalgamated product).

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 occurence 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 occurences of t, they are all cancelled after lhs is applied, in fact you could remove every occurence of t from p and still have a certificate of the same congruence.

Definition 5.9. 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 $t^n at$ are the exponent sums of t in t^n, $t^n a$ and $t^n at$.

Definition 5.10. h is a map $F(S) \to F(S \cup \{t'\})$, where t' is some letter not in S. h replaces every occurence 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 5.11. θ is a group homomorphism $F(S \cup \{t'\})$. Let $s \in S$. Then

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

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

Definition 5.12. (PowProof) *PowProof* is a map $F(S) \to F(S)$. $\mathrm{PowProof}(w)$ is defined to be $h(w)$, but with every occurence of t' replaced with 1.

Theorem 5.13. 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 5.13.1. 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 $b \le a$ 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 occurence 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 occurence of t'.

Proof of Lemma 5.13.1. $\prod_{i=0}^{a} s_i^{k_i}$ 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 occurence 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 5.13
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 occurence 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 5.13.1. 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 5.13.1. Therefore when the word is reduced it will not contain t' by Lemma 5.13.1. This means that deleting all occurences 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 $\psi_2(\mathrm{Eval}(r)(F(\mathrm{PowProof})(p)) = \psi_2(\mathrm{Eval}(r)(F(h)(p))) = \theta(\mathrm{Eval}(r)(F(h)(p))) = \mathrm{Eval}(\psi_2(r))(p)$.