1 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 1.1 (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.

2 HNN Extensions

Definition 2.1 (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 $\{tat^{-1}\phi(a^{-1}) | a \in A\}$

Theorem 2.2 (Britton's Lemma)**.** Let $w = g_0 t^{k_1} g_1 t^{k_2} g_2 \cdots t^{k_n} g_n$ be a word in the HNN extension. If for every i, $k_i \neq 0$, $k_i > 0$ implies $g_i \notin A$ and $k_i < 0$ implies $g_i \notin B$, and w contains a t, then $w \neq 1$ cite (On Britton's theorem Charles Miller)

Corollary 2.2.1. 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 2.2.1 Suppose $w = g$ with $g \in G$, then $g^{-1}w$ also meets the conditions in Theorem 2.2, and therefore $gw^{-1} \neq 1$, contradicting $w = g$.

The HNN normalization process 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 $b \in B$. This produces a word of the form in Theorem 2.2.

3 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 3.1. (Eval) $\text{Eval}(r)$ is a map $F(G) \to G$, sending a basis element $g \in G$ to grg^{-1}.

Definition 3.2 (P functor)**.** We 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 3.2.1 (lhs and rhs)**.** We 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 $\mathrm{Eval}(a)b$. Since $\mathrm{Eval}(a)$ is in the kernel of the quotient map, for any $p \in P(G)$, $\mathrm{lhs}(p)$ and $\mathrm{rhs}(p)$ are equal in the quotient by r. Therefore an element p of $P(G)$ can be regarded as a certificate of the congruence $\mathrm{lhs}(p) \equiv \mathrm{rhs}(p) \bmod r$.

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 3.2.2. (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 3.2.3. (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 $\mathrm{Trans}(p, q)$ such that $\mathrm{Trans}(p, q)$ is a certificate of the congruence $a = c \bmod r$. If $p = (p_1, p_2)$, and $q = (q_1, q_2)$, then $\mathrm{Trans}(p, q) = (p_1q_1, q_2)$.

Definition 3.2.4. (Refl) Given $a \in G$, $(1, a)$ is a certificate of the congruence $a = a \bmod r$. Call this $\mathrm{Refl}(a)$.

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

Definition 3.2.5. (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 $\mathrm{ChangeRel}(g, (p_1, p_2))$ is defined to be $(\phi(g)(p_1), p_2)$ for $g \in G$ and $(p_1, p_2) \in P(G)$.

4 Magnus' Method

We 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.

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

4.2 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 4.2) 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 1.1) 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.

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. Cyclically conjugate r, so 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 $t^i s t^{-i}$ 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 so α is an isomorphism.

We then apply the HNN normalization procedure, described in detail in Section 4.6. 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 $A \cap B$, it is always possible to push t to the right. 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 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 $T' := \{s_0 \in S' | s \in T\}$.

4.3 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 $y \in T$ but $t \notin T$, then $y \cup \{t\}$ does not contain every letter in r. 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 $F(S)/r$ 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.

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

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

ψ_1 is an isomorphism as a map $F(S)/r$ to $F(r)/\psi_1(r)$, the inverse given by sending y to yt^{α}. ψ_2 is injective as a map $F(S)/r$ to $F(r)/\psi_2(r)$, this is proven constructively in Section 4.11. So ψ is injective as a map $F(S)/r$ to $F(r)/\psi(r)$.

The exponent sum of t in $\psi(r)$ is 0, so the problem can be solved using the method described in Section 4.2. The image of the subgroup generated by T under ψ might not be the subgroup generated by a set of letters, but it is contained in such a subgroup.

4.4 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 4.1 (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 $\text{AddBasis}(m)(x, n) = (x, m + n)$.

Definition 4.2 (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 4.3 (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$(r') = r$.

4.5 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 $\mathbb{Z}/n\mathbb{Z}$. However computing the appropriate proof term requires some explanation.

To compute the proof term, we write a function that takes an unnormalized word $w \in F(S)$, and a normalized proof word with proof $p \in P(F(S))$, and returns a word $q \in P(F(S))$, such that $\mathrm{lhs}(q) = w\mathrm{lhs}(p)$ and $\mathrm{rhs}(q)$ is a normalization of $w\mathrm{rhs}(p)$. By normalized, we mean that there is no occurence of a^k where $k \neq 0$ is a multiple of n.

We can normalize the word a^k where k is a multiple of n to $([1]^(n/k), 1) \in P(F(S))$, where $[1] \in F(F(S))$, is the basis element corresponding to $1 \in F(S)$.

4.6 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 4.4. Define a map on a basis element i as follows

$$\begin{cases} (i, 0) \in F(S') & i \neq t \\ t' & i = t \end{cases} \tag{5}$$

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 gt'^{-1}, we can use *Solve* to check whether g is equal to a word $a \in A$ in the quotient $F(S')/r'$, and if it is equal to some word a, rewrite gt'^{-1} to $t'^{-1}\mathrm{Changesubscript}(-1)(a)$. Similarly, For each occurence of gt', use *Solve* to check whether g is equal to a word $b \in B$ in the quotient $F(S')/r'$, and if it it equal to some word b rewrite gt' to $t'\mathrm{ChangeSubscript}(1)(b)$.

This is a slight deviation from the procedure decscribed in Section 2, the rewriting rules are applied in the opposite direction, and therefore the words are not quite in the normal form described in Theorem 2.2, but are in a reversed version of this normal form. The rewriting rules are applied from left to right; wherever a rewriting rule is applied, everything to the left of where that rule is applied should already be in HNN normal form. The reasons for these are to generate shorter certificates and are described in Section 4.6.1.

4.6.1 Computing Proof Certificates

To compute proof certificates a slightly modification of the procedure described in Section 4.6 is used.

First define a modification of Definition 4.4, from $F(S)$ into the binary coproduct $P(F(S \times \mathbb{Z})) * \langle t' \rangle$.

Definition 4.5. 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 4.6. 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 4.6. 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 4.7. (conjP) Let $(p, a) \in P(F(S \times \mathbb{Z}))$ and $k \in \mathbb{Z}$. Define *ConjP* to map into $P(F(S \times \mathbb{Z}))$

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

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 4.6 replaced each occurence 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 proof certificate suppose there is an occurence of pt'^{-1} with $p \in P(F(S \times \mathbb{Z}))$. Use *Solve* to check whether $\text{rhs}(p)$ 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 $t'^{-1}\text{ConjP}(1, \text{Trans}(p, q))$.

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

4.6.2 Proof Lengths

It is important to apply the rewriting procedure from left to right. This will usually produce shorter proof certificates. Consider normalizing $t'^{-1}wt'vt'^{-1}$ with $w, v \in F(S')$. Suppose we normalize right to left. $t'v$ is normalized to pt' with $p \in P(F(S \times \mathbb{Z}))$. After that substitution the new word $t'^{-1}wp$, with $wp \in P(F(S \times \mathbb{Z}))$. Suppose $\text{rhs}(wp)$ is normalized to $q \in P(F(S \times \mathbb{Z}))$. Then the final normalized word is $\text{conjP}(1, \text{Trans (wp, q)})t'^{-1}$.

The proof part of $\text{Trans}(wp, q)$ is $\text{MulFree}(w)(\text{Left}(p))\text{Left}(q)$.

Normalizing the other way, we first normalize $t'^{-1}w$ to $p_2t'^{-1}$ with $p_2 \in P(F(S \times \mathbb{Z}))$. Then the new word is p_2vt^{-1}. Suppose $\text{rhs}p_2v$ is normalized to $q_2 \in P(F(S \times \mathbb{Z}))$. Then the final normalized word is $t'^{-1}\text{conjP}(-1, \text{Trans}(p_2v, q_2))$.

The proof part of $\mathrm{Trans}(p_2 v, q)$ is $\mathrm{Left}(p_2)\mathrm{Left}(q)$. There is no use of *MulFree* since $\mathrm{Left}(v) = 1$. On average using *MulFree* will make words longer, and so the left to right normalization produces shorter proofs. The left to right normalization was found to produce shorter proofs in practice.

4.7 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 $\psi_2(a) = \psi_2(b) \bmod \psi_2(r)$, 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 4.8. 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 4.9. (PowProofAux) *PowProofAux* is a map $F(S) \to F(S \cup \{t'\})$, where t' is some letter not in S. *PowProofAux* 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 $\mathrm{PowProofAux}(w)$ is either not a multiple of n, or it is zero.

Definition 4.10. (PowProof) *PowProof* is a map $F(S) \to F(S)$. $\mathrm{PowProof}(w)$ is defined to be $\mathrm{PowProofAux}(w)$, but with every occurence of t' replaced with 1.

Theorem 4.11. 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 4.11.1. 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 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 t'.

Proof of Lemma 4.11.1. $\prod_{i=0}^{a} s_i^{k_i}$ 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 4.11
If $\mathrm{Eval}(\psi_2(r))(p)$ is in the image of ψ_2, then $\mathrm{Eval}(r)(\mathrm{PowProofAux}(p))$ has the property that

for every occurence of t'^k, k is a multiple of n. If $p' := \text{PowProofAux}(p)$, then $\text{Eval}(r)(p')$ can be written as a product of the form in Lemma 4.11.1. If $r' = \prod_i u_i^{l_i}$, then to write $\text{Eval}(r)(p')$ in this form, send $\left(\prod_i \left[\prod_{j=1}^a s_{ij}^{k_j}\right], \prod_i v_i^{m_i}\right) \in P(F(S \cup \{t'\}))$, to

$$\left(\prod_i \left(\prod_{j=1}^{a} {s_{ij}}^{k_j}\right) \left(\prod_j u_j^{l_j}\right) \left(\prod_{j=1}^{a} {s_{i(a-j)}}^{-k_{a-j}}\right)\right) \prod_i v_i^{m_i} \tag{8}$$

If all the nested products in Equation 8 are appended into one long product, then the product has the form in Lemma 4.11.1. Therefore when the word is reduced it will not contain t'. This means that deleting all occurences of t' will in p' will not change $\text{Eval}(r)(p')$, and therefore $\text{Eval}(r)(PowProof(p)) = \text{Eval}(r)(p')$.