1

A Computer Implementation of Whitehead's Algorithm

Michael Lau
St. Olaf College
Northfield, MN

Advisor: Professor Dennis Garity
NSF sponsored REU Program
Oregon State University

August 24, 2010

Abstract

A computer implementation of Whitehead's Algorithm is given for $F(a, b, c)$, and is then applied to give a classification of minimal words of length at most 6. The notion of CP-equivalence is introduced in conjunction with this classification, and some properties of Whitehead automorphisms and equivalence are investigated.

[*]The extensive help of Professor Dennis Garity in developing the ideas found in this paper is gratefully acknowledged.

1 Whitehead's Algorithm

In 1936, J.H.C. Whitehead used topological techniques to prove that if two words of a finitely generated free group are equivalent under an automorphism of the group, then they are equivalent under a finite sequence of automorphisms of a special kind, the so-called Whitehead automorphisms [3],[4]. Moreover, for any pair of equivalent words w and w', where w' has the minimum length occurring in its equivalence class, the words obtained at each step in this transformation from w to w' are of strictly decreasing length until the minimum length is attained, after which the length remains constant.

Some Notation.

- $\overline{x_k}$ will denote the inverse of the letter x_k.
- $F(x_1, \ldots, x_n)$ will denote the free group of rank n, having $x_1, \ldots, x_n$ as its generators, with only the trivial relation $x_k\overline{x_k} = \overline{x_k}x_k = 1$.
- $w \sim w'$ will indicate the equivalence of two words $w, w' \in F = F(x_1, \ldots, x_n)$ under some automorphism of F.
- $|w|$ will be defined as the length of a word w.[1]

In the following two definitions, we will use L to denote the set $\{x_1, x_2, \ldots, x_n, \overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$.

Definition 1.1 *If $F = F(x_1, \ldots, x_n)$, then the* **Whitehead Type I automorphisms** *of F are the members of $AutF \cap SymL$–that is, those permutations ϕ acting on L and preserving inverses (i.e. $\overline{\phi(x_k)} = \phi(\overline{x_k})$).*

Definition 1.2 *If $F = F(x_1, \ldots, x_n)$, then given any $x \in L$ and $A \subset L$ such that $x \in A$ and $\overline{x} \notin A$, the pair (A, x) will be called a* **Whitehead Type II automorphism***, where (A, x) is the automorphism determined by*

[1]Keep in mind that the length of a word is the number of letters in the word after it has been **reduced**–i.e. after pairs of consecutive letters $x_k\overline{x_k}$ and $\overline{x_k}x_k$ have been removed until no such pair remains. By convention, the empty word is reduced.

the following actions on each $y \in L$:

$$y(A,x) = \begin{cases} yx & \text{if} \quad y \in A, \overline{y} \notin A, y \notin \{x, \overline{x}\} \\ \overline{x}y & \text{if} \quad y \notin A, \overline{y} \in A, y \notin \{x, \overline{x}\} \\ \overline{x}yx & \text{if} \quad y, \overline{y} \in A \\ y & \text{otherwise} \end{cases}$$

Example 1.1 $\rho = (x_1\ \overline{x_3}\ \overline{x_2})(\overline{x_1}\ x_3\ x_2)$ is a Whitehead Type I automorphism (written in cyclic notation).

Example 1.2 If $A = \{x_1, x_2, \overline{x_1}, \overline{x_3}\}$, then $x_2x_1x_3x_2x_3(A, x_2) = x_2\overline{x_2}x_1x_2\overline{x_2}x_3x_2\overline{x_2}x_3 = x_1x_3x_3$.

Whitehead's Theorem 1.1 *If $w, w' \in F = F(x_1, \ldots, x_n)$ such that $w \sim w'$ and w' is of minimum length for their equivalence class, then there is some sequence $\alpha_1, \alpha_2, \ldots, \alpha_m$ of Whitehead automorphisms such that $w\alpha_1\alpha_2 \cdots \alpha_m = w'$ and for all positive integers $k \leq m \quad |w\alpha_1\alpha_2 \cdots \alpha_k| \leq |w\alpha_1\alpha_2 \cdots \alpha_{k-1}|$ with strict inequality unless $w\alpha_1\alpha_2 \cdots \alpha_{k-1}$ is also of minimum length.*

As was mentioned above, this result was discovered and proven by Whitehead [3],[4]. Successively simpler proofs have been given by Rapaport[2] in 1958, and by Higgins and Lyndon[1] in 1974.

The theorem has several immediate consequences, of which we will concern ourselves with only the simplest and most obvious–the existence of an easily constructible algorithm for determining whether or not two words of a finitely generated free group are equivalent under an automorphism of the group. Moreover, since our ultimate goal in developing a computer implementation of this algorithm is to provide a useful tool for generating and testing hypotheses related to such equivalence or nonequivalence, we will limit ourselves to studying applications of the algorithm to free groups of rank 3, in order to study a nontrivial automorphism group while avoiding the complexity and long computer computation times associated with free groups of higher rank. The specific value of 3 was chosen to benefit fellow researchers studying free homotopy classes of self-intersecting curves on the twice-punctured torus, since they consider equivalence questions regarding elements of $F(a, b, c)$ under automorphisms of the group when demonstrating distinctness of homotopy classes.[2]

[2]Such questions arise since homotopic curves are equivalent under homeomorphisms of the torus, which in turn induce automorphisms of the first homology group, which is, in

Whitehead's Algorithm
Suppose $w, v \in F = F(x_1, \ldots, x_n)$.

1. Find mimimum length words w^* and v^* equivalent to w and to v, respectively. Since there are only finitely many Whitehead automorphisms of $F(x_1, \ldots, x_n)$, this may be done as follows:

   Check to see if w or v is equivalent , under a single Whitehead automorphism, to a word of strictly lesser length. If one of them is, then replace it with this shorter word and repeat this process. If not, then w and v are of minimum length.

2. Make a list of all the minimum length words equivalent to w by executing the following steps:

   (i) Using the minimum length word $w^* \sim w$ found in Step 1, make a list consisting of w^* and all the same-length words equivalent to w^* under a single Whitehead automorphism.

   (ii) Make a new list by appending (to the old list created in Step 2(i)) all the same-length words equivalent to a word on the old list via a single Whitehead automorphism. If this new list contains no words not already found on the old list, then clearly there are no other minimum length words that are equivalent to w under a finite sequence of Whitehead automorphisms. If, on the contrary, it contains new words, then reapply this step, using the "new list" in place of the "old list". Note that since there are only finitely many words of a given length in F, Step 2 must eventually terminate.

3. If v^* is on the list produced in Step 2, then clearly $w \sim v$. Otherwise, it follows directly from Whitehead's Theorem that $w \not\sim v$.

this special case, $F(a, b, c)$. This means that if the words corresponding to two curves are nonequivalent under $AutF(a, b, c)$, then the two curves must belong to distinct homotopy classes.

2 A Maple V Implementation of Whitehead's Algorithm

Now that we have an algorithmic interpretation of Whitehead's Theorem, we are almost ready to present a series of programs (written for Maple V, release 4) that will perform various functions related to the execution of the Whitehead Algorithm. However, we will first need to make a few observations in order to reduce computer computation time later:

Lemma 2.1 *Whitehead Type I automorphisms never change the length of a word.*

Proof Since Whitehead Type I automorphisms are permutations, they cannot increase the number of letters in a word. They preserve inverses while permuting letters, which means that they cannot introduce new adjacent $x\overline{x}$ or $\overline{x}x$ pairs, so they cannot decrease word length. ■

Lemma 2.2 *If a letter x and its inverse $\overline{x}$ are not present in a word w, then no Whitehead Type II automorphisms of the form (A, x) will reduce the length of w.*

Proof By definition, (A, x) acts on letters $y \in F(x_1, \cdots, x_n)$ by mapping y to y, $\overline{x}y$, yx, or $\overline{x}yx$. Since we have assumed that x is not present in w, the result of such mappings is merely the possible introuction of additional letters (occurrences of x and/or $\overline{x}$) without the possibility of new cancellations (since there were no previously existing occurrences of x or $\overline{x}$ with which to cancel). ■

Lemma 2.3 *If w and v are equivalent words in $F(x_1, \cdots, x_n)$ and v is of minimum length, then $\exists \alpha_1, \alpha_2, \ldots, \alpha_s$ such that*
(i) $w\alpha_1\alpha_2 \cdots \alpha_s = v$
(ii) $\alpha_1, \alpha_2, \ldots, \alpha_{s-1}$ *are Whitehead Type II automorphisms*
(iii) α_s *is a (possibly trivial) Whitehead Type I automorphism*
(iv) $|w\alpha_1\alpha_2 \cdots \alpha_k| \leq |w\alpha_1\alpha_2 \cdots \alpha_{k-1}|$ *with strict inequality unless* $w\alpha_1\alpha_2 \cdots \alpha_{k-1}$ *is of minimum length.*

Proof By Whitehead's Theorem, there exist Whitehead automorphisms $\beta_1, \ldots, \beta_r$ such that $w\beta_1 \cdots \beta_r = v$ and $|w\beta_1 \cdots \beta_k| \leq |w\beta_1 \cdots \beta_{k-1}|$ for all

$1 \leq k \leq r$ with strict inequality unless $w\beta_1 \cdots \beta_{k-1}$ is also of minimum length.

Note that for all $y \in F(x_1, \cdots, x_n)$, Whitehead Type I automorphisms ρ, and Whitehead Type II automorphisms (A, x),

$$y\rho(A,x) = y(A\rho^{-1}, x\rho^{-1})\rho, \text{ where } A\rho^{-1} = \{z\rho^{-1} : z \in A\}. \tag{1}$$

This follows from the facts that $y\rho = x$ iff $y = x\rho^{-1}$, $y\rho \in A$ iff $y \in A\rho^{-1}$, and $x\rho^{-1}\rho = x$.

Applying this observation to Whitehead's result, inducting on the position of a Whitehead Type I automorphism $\rho \in \{\beta_1, \ldots, \beta_r\}$, then inducting on the number of Whitehead Type I automorphisms in $\{\beta_1, \ldots, \beta_r\}$ gives a sequence of Whitehead automorphisms $\gamma_1, \ldots, \gamma_r$ such that $w\gamma_1 \cdots \gamma_r = v$ and there are no Whitehead Type I automorphisms occurring before a Whitehead Type II automorphism in this sequence. Since the set of all Whitehead Type I automorphisms clearly forms a group under function composition[3],the composition of all the Whitehead Type I automorphisms in $\gamma_1 \ldots, \gamma_r$ is clearly a single Whitehead Type I automorphism, which proves (i), (ii), and (iii).

To prove (iv), assume $\alpha_1, \ldots, \alpha_n$ are a sequence of Whitehead automorphisms given by Whitehead's Theorem. Suppose $\rho = \alpha_j$ is Whitehead Type I and $(A, x) = \alpha_{j+1}$ is Whitehead Type II. Let $\alpha'_{j+1} = (A\rho^{-1}, x\rho^{-1})$. By Lemma 2.1, $|w\alpha_1 \cdots \alpha_j| = |w\alpha_1 \cdots \alpha_{j-1}|$, so it follows from Whitehead's Theorem that $w\alpha_1 \cdots \alpha_{j-1}$ has minimal length. It is now sufficient to show that (a) $|w\alpha_1 \cdots \alpha_{j-1}\alpha'_{j+1}| = |w\alpha_1 \cdots \alpha_{j-1}|$ and (b) $|w\alpha_1 \cdots \alpha_{j-1}\alpha'_{j+1}\alpha_j| \leq |w\alpha_1 \cdots \alpha_{j-1}\alpha'_{j+1}|$ with strict inequality unless $w\alpha_1 \cdots \alpha_{j-1}\alpha'_{j+1}$ has minimal length. Note that $|w\alpha_1 \cdots \alpha_{j-1}\alpha'_{j+1}| = |w\alpha_1 \cdots \alpha_{j-1}\alpha'_{j+1}\alpha_j| = |w\alpha_1 \cdots \alpha_{j+1}| = |w\alpha_1 \cdots \alpha_{j-1}|$. (The first equality is from Lemma 2.1, the second comes from Equation (1), and the third follows from monotonicity and the fact that $w\alpha_1 \cdots \alpha_{j-1}$ has minimal length.) But what we have just shown (besides proving (a)) is that $w\alpha_1 \cdots \alpha_{j-1}\alpha'_{j+1}$ has minimal length. Then we may complete our proof of (iv) merely by noting that $|w\alpha_1 \cdots \alpha_{j-1}\alpha'_{j+1}\alpha_j| = |w\alpha_1 \cdots \alpha_{j-1}\alpha'_{j+1}|$. ■

These lemmas may now be combined with Whitehead's Algorithm to construct the following collection of Maple programs:

[3]Adopting the notation of Definition 1.1, *AutF* and *SymL* can easily be embedded as subgroups of *SymF*. Then since the intersection of subgroups is itself a group, the Whitehead Type I automorphisms form a group.

2.1 DWH("Do WHitehead automorphism")

DWH takes 3 arguments, a word wd, a Whitehead set A and a "distinguished" element x. (x and $\overline{x}$ should not be in A). It returns the result of applying the Whitehead Type II automorphism $(A \cup \{x\}, x)$ to wd.

kill takes 2 arguments, a word wd and a positive integer k. It returns the word formed by deleting the kth and $(k+1)$st letters of wd.

```
> kill :=proc(wd,k)
> if length(wd)>k
> then wdn :=cat(substring(wd,1..k-1),substring(wd,k+2..length(wd)));
> else wdn:=wd;fi;
> wdn;
> end:
```

inv("INVerse") takes 1 argument, a letter from the set $\{a, b, c, A, B, C\}$ and returns the inverse of that letter where X denotes the inverse of x for all x in $\{a, b, c\}$.

```
> inv:=proc(y) x:=y;
> if x='a' then X:='A';
> elif x='b' then X:='B';
> elif x='c' then X:='C';
> elif x='A' then X:='a';
> elif x='B' then X:='b';
> elif x='C' then X:='c';
> else ERROR('dontusebadwords');
> fi;
> end:
```

CFC("Check For Cancellations") takes 1 argument, a word wd on 6 letters a, b, c, A, B, C where X denotes the inverse of x for all x in $\{a, b, c\}$. It returns the word formed by performing all allowable cancellations on wd. If the empty word is the result, it returns "1".

```
> CFC :=proc(wd::string)
> w:=wd;
> l:=length(w); k:=1;
> if l>1
> then while k < l
>      do
>        if substring(w,k)=inv(substring(w,k+1))
>        then w :=kill(w,k); k:=max(k-1,1); l:=l-2;
>        else k:=k+1;
>        fi;
>      od;
> fi;
> if l=0 then w:=`1` else w; fi;
> end:
```

```
> DWH :=proc(wd,set,x)
> w2:=wd; w3:=w2;
> l:=length(w2); m:=1;
> for k from 1 to l do
> if member(substring(w2,k..k),{x,inv(x)})
> then m:=m+1 elif member(substring(w2,k..k),set)
> then if member(inv(substring(w2,k..k)),set)
> then w3:=cat(substring(w3,1..m-1),inv(x),substring(w2,k..k),
> x,substring(w2,k+1..length(w2))); m:=m+3;
> else w3:=cat(substring(w3,1..m),x,substring(w2,k+1..length(w2)));
> m:=m+2;
> fi;
> elif member(inv(substring(w2,k..k)),set)
> then w3:=cat(substring(w3,1..m-1),inv(x),substring(w2,k..length(w2)));
> m:=m+2;
> else m:=m+1;
> fi;
> od; w3;
> end:
```

2.2 minwd (”Minimum Word”)

minwd[4] takes two arguments, a word wd in the free group $F(a,b,c)$ with the convention that X denotes the inverse of x for all x in $\{a,b,c\}$, and an empty list [ ]. To avoid complications with Maple ”evaluating” the word it is given, it is best to enclose the argument in a pair of single quotation marks ’and’. It returns a word that is Whitehead equivalent to wd and is of minimal length, and the list of Whitehead automorphisms needed to get the minimal word. It uses all three of the lemmas to reduce computation time.

oldminwd takes one argument, a word wd in $F(a,b,c)$. To avoid complications with Maple ”evaluating” the word it is given, it is best to enclose the argument in a pair of single quotation marks ’and’. It returns a word that is Whitehead equivalent to wd and is of minimal length.

```
Whitehead sets
aset
> aset:=combinat[powerset]({'b','c','B','C'}):
> aset:=aset minus {{}};

Aset
> Aset:=combinat[powerset]({'b','c','B','C'}):
> Aset:=Aset minus {{}};

bset
> bset:=combinat[powerset]({'a','c','A','C'}):
> bset:=bset minus {{}};

Bset
> Bset:=combinat[powerset]({'a','c','A','C'}):
> Bset:=Bset minus {{}};

cset
> cset:=combinat[powerset]({'b','a','B','A'}):
```

[4]The idea and code for using “Whitehead sets” to improve the efficiency of minwd (via Lemma 2.2) was supplied by Professor Garity.

```
> cset:=cset minus {{}};

Cset
> Cset:=combinat[powerset]({'b','a','B','A'}):
> Cset:=Cset minus {{}};


> minwd:=proc(wd::string,lst::list)
> wold:=CFC(wd); WHlist:=lst; wnew:=wold;
> k:=1;
> while k > 0 do
>   j:=0;
>   if SearchText(`a`,wold)>0 or SearchText(`A`,wold)>0
>   then for i from 1 to 15 do
>        wnew:=CFC(DWH(wold, aset[i],`a`));
>          if length(wnew)<length(wold)
>          then wold:=wnew; WHlist:=[op(WHlist),[aset[i],a]];
>          j:=1;
>          fi;
>        od;
>        for i1 from 1 to 15 do
>        wnew:=CFC(DWH(wold, Aset[i1],`A`));
>          if length(wnew)<length(wold)
>          then wold:=wnew; WHlist:=[op(WHlist),[Aset[i1],A]];
>          j:=1;
>          fi;
>        od;
>   fi;
>   if SearchText(`b`,wold)>0 or SearchText(`B`,wold)>0
>   then for i2 from 1 to 15 do
>        wnew:=CFC(DWH(wold, bset[i2],`b`));
>          if length(wnew)<length(wold)
>          then wold:=wnew; WHlist:=[op(WHlist),[bset[i2],b]];
>          j:=1;
>          fi;
>        od;
>        for i3 from 1 to 15 do
>        wnew:=CFC(DWH(wold, Bset[i3],‘B‘));
>          if length(wnew)<length(wold)
>          then wold:=wnew; WHlist:=[op(WHlist),[Bset[i3],B]];
>          j:=1;
>          fi;
>        od;
>   fi;
>   if SearchText(‘c‘,wold)>0 or SearchText(‘C‘,wold)>0
>   then for i4 from 1 to 15 do
>        wnew:=CFC(DWH(wold, cset[i4],‘c‘));
>          if length(wnew)<length(wold)
>          then wold:=wnew; WHlist:=[op(WHlist),[cset[i4],c]];
>          j:=1;
>          fi;
>        od;
>        for i5 from 1 to 15 do
>        wnew:=CFC(DWH(wold, Cset[i5],‘C‘));
>          if length(wnew)<length(wold)
>          then wold:=wnew; WHlist:=[op(WHlist),[Cset[i5],C]];
>          j:=1;
>          fi;
>        od;
>   fi;
>   if j=1 then k:=1; else k:=0; fi;
> od;
> ouput:=[wold,WHlist];
> end:


> oldminwd:=proc(wd::string)wold:=CFC(wd); wnew:=wold;
> k:=1;
> while k > 0 do
>   j:=0;
>   if SearchText(‘a‘,wold)>0 or SearchText(‘A‘,wold)>0
>   then for i from 1 to 15 do
>        wnew:=CFC(DWH(wold, aset[i],‘a‘));
>          if length(wnew)<length(wold)
>          then wold:=wnew;
>          j:=1;
>          fi;
>        od;
>        for i1 from 1 to 15 do
>        wnew:=CFC(DWH(wold, Aset[i1],‘A‘));
>          if length(wnew)<length(wold)
>          then wold:=wnew;          j:=1;
>          fi;
>        od;
>   fi;
>   if SearchText(‘b‘,wold)>0 or SearchText(‘B‘,wold)>0
>   then for i2 from 1 to 15 do
>        wnew:=CFC(DWH(wold, bset[i2],‘b‘));
>          if length(wnew)<length(wold)
>          then wold:=wnew;
>          j:=1;
>          fi;
>        od;
>              for i3 from 1 to 15 do
>        wnew:=CFC(DWH(wold, Bset[i3],‘B‘));
>          if length(wnew)<length(wold)
>          then wold:=wnew;
>          j:=1;
>          fi;
>        od;
>   fi;
>   if SearchText(‘c‘,wold)>0 or SearchText(‘C‘,wold)>0
>   then for i4 from 1 to 15 do
>        wnew:=CFC(DWH(wold, cset[i4],‘c‘));
>          if length(wnew)<length(wold)
>          then wold:=wnew;
>          j:=1;
>          fi;
>        od;
>        for i5 from 1 to 15 do
>        wnew:=CFC(DWH(wold, Cset[i5],‘C‘));
>           if length(wnew)<length(wold)
>           then wold:=wnew;
>           j:=1;
>           fi;
>         od;
>    fi;
>    if j=1 then k:=1; else k:=0; fi;
> od;
> ouput:=wold;
> end:
```

DPS(Do PermutationS) takes 1 argument, a word wd from the free group $F(a, b, c)$ where X denotes the inverse of x for all x in $\{a, b, c\}$. It returns a set containing the results of performing each Whitehead Type 1 automorphism (permutations of letters in wd) on wd.

transpose takes 3 arguments, a word wd and 2 letters x and y. It returns the word formed by replacing every x occurring in wd with a y and every y occurring in wd with an x.

```
> transpose :=proc(wd,x,y) l :=length(wd); w1:=wd;
> for k from 1 to l do
> if substring(w1,k..k)=x
>    then w1:=cat(substring(w1,1..k-1),y,substring(w1,k+1..l));
> elif substring(w1,k..k)=y
>    then w1:=cat(substring(w1,1..k-1),x,substring(w1,k+1..l));
> else;
> fi;
> od;
> w1;
> end:
```

II(interchange inverses) takes 2 arguments, a word wd and a letter x. It returns the word formed by replacing every x occurring in wd with the inverse of x.

```
> II:=proc(wd, x)
> w9:=transpose(wd,x,inv(x));
> end:
```

P1,P2,P3,P4, and P5 ("Permutations 1 through 5") each take 1 argument, a word wd in the free group $F(a, b, c)$ (where X denotes the inverse of x for all x in $\{a, b, c\}$), and return the word formed by permuting the letters of wd according to the permutations (in cyclic notation)

- (a b)(A B) for P1,
- (a c)(A C) for P2,
- (b c)(B C) for P3,
- (a b c)(A B C) for P4,
- (a c b)(A C B) for P5.

```
>P1:=proc(wd) w1:=wd; w1:=transpose(transpose(w1,'a','b'),'A','B'); end:
>P2:=proc(wd) w1:=wd; w1:=transpose(transpose(w1,a,c),A,C); end:
>P3:=proc(wd) w1:=wd; w1:=transpose(transpose(w1,'b','c'),'B','C'); end:
>P4:=proc(wd)w1:=wd;
>w1:=transpose(transpose(transpose(transpose(w1,'a','b'),'A','B'),'a','c'),
> 'A','C'); end:
>P5:=proc(wd) w1:=wd; w1:=transpose(transpose(transpose(transpose(w1,'a','c'),
> 'A','C'),'a','b'),'A','B');end:
```

```
> Permls:=combinat[powerset]({a,b,c});
```

```
> DPS:=proc(wd) ls:={wd};for k from 1 to 8 do if Permls[k]={} then ls:=ls
>  union {P1(wd)} union {P2(wd)} union {P3(wd)} union {P4(wd)} union {P5(wd)};
> else wd9:=wd;for m from 1 to nops(Permls[k]) do
> wd9:=II(wd9,Permls[k][m]); od; ls:=ls union {wd9} union {P1(wd9)} union
> {P2(wd9)} union {P3(wd9)} union {P4(wd9)} union {P5(wd9)};fi;od;ls; end:
```

2.3 EWL(Equivalent Word List)

EWL takes 1 argument, a reduced word wd from $F(a, b, c)$. It returns the list of all reduced words that are Whitehead equivalent to wd. It uses the results of Lemma 2.1 and Lemma 2.3 to reduce computation time.

EW1L(Equivalent via a Whitehead Type 1 automorphism (allowable permutation of letters)) takes 1 argument, a set of reduced words wdls from $F(a, b, c)$. It returns the set of all reduced words equivalent to a word from wdls under a Whitehead Type 1 automorphism.

```
> EW1L:=proc(ewl::set) wl:=ewl;
> for k from 1 to nops(ewl) do
> wl:=wl union DPS(ewl[k]);
> od; end:
```

EW2L(Equivalent via a single Whitehead Type 2 automorphism List) takes one argument, a reduced word wd from $F(a, b, c)$ and returns the set of all reduced words equivalent to wd under a single Whitehead Type 2 automorphism.

```
> EW2L:=proc(wd) global w5;w5:=0;
> w4:=wd; l:=length(w4); ewl:={};
> ps:=combinat[powerset](['a','b','c','A','B','C']);
>  for k from 1 to 64 do
>   if ps[k]=[] then ps:=ps; else
>     for i from 1 to nops(ps[k]) do
>       if member(inv(ps[k][i]),ps[k]) then ps:=ps;
>        else w5:=CFC(DWH(w4,ps[k],ps[k][i]));
>         if length(w5)<=l
>          then ewl:=ewl union {w5}; fi; fi; od;fi; od; ewl;end:
```

CWL(Complete Word List) takes 1 argument, a set of reduced words wdls from $F(a, b, c)$. It returns the list of all reduced words that are Whitehead equivalent to a word from wdls.

```
> CWL:=proc(wdls)
> w7:=wdls; w8:=wdls; j:=0; while j<2 do
> for k from 1 to nops(w7) do
> w8:=w8 union EW2L(w7[k]); od;
> if w8=w7 then p:=EW1L(w7); j:=9; else j:=0; w7:=w8;
> fi; od;p; end:

> EWL:=proc(wd)
> CWL({wd}); end:
```

2.4 EQW(EQuivalent Words)

EQW takes 2 arguments, each a (possibly unreduced) word from $F(a,b,c)$. It returns "equivalent" if the 2 words are equivalent under some automorphism of $F(a,b,c)$, and "not equivalent" if they are not equivalent under any automorphism of $F(a,b,c)$. Due to the limitations of Maple V's computational engine, it is sometimes necessary to use the functions "minwd" and "EWL" instead of "EQW" to make the determination of equivalence when words are of longer length.

```
> EQW:=proc(wd1,wd2)
> if member(oldminwd(wd2),EWL(oldminwd(wd1))) then 'equivalent';
> else 'not equivalent'; fi;
> end:
```

3 Applications to Minimal Words of Length at most 6

Using the programs constructed in Section 2, it is possible to write simple functions to find the minimal words in $F(a,b,c)$ (i.e. words of minimal length in their equivalence class) of length at most 6, and then partition them into equivalence classes. After making the trivial observation that Whitehead's Theorem guarantees that minimal words of different length are non-equivalent, the approach becomes straightforward. Unfortunately, programs relying on "brute force" to perform these tasks soon encounter prohibitively long run-times. Examples of straightforward "brute-force" programs (which we used for words of length at most 5) and a slightly more complex approach(used for length-6 words)[5] may be found in the Appendix.

Though lists generated in this manner are complete, they do not easily lend themselves to analysis, since many of their equivalence classes are quite large.[6] For this reason, it is helpful to introduce a new notion of equivalence, by which we will further categorize elements within each equivalence class.

Definition 3.1 *Suppose $w, v \in F(a,b,c)$ with $w = w_1, \ldots, w_n$ for some $w_i \in \{a, b, c, \overline{a}, \overline{b}, \overline{c}\}$. We say that w and v are* **cyclically equivalent** *if $v = w_{k+1}w_{k+2}\cdots w_n w_1 \cdots w_k$ for some $k \leq n$.*

Definition 3.2 *Two words $w_1, w_2 \in F(a,b,c)$ are said to be* **CP-equivalent**[7] *if there exists a Whitehead Type I automorphism ψ such that $\psi(w_1)$ is cyclically equivalent to w_2.*

Definition 3.3 *Given a minimal-length word $w \in F(a,b,c)$, we say that w is in* **reduced-CP (R-CP) form** *if it is the first word of its CP-equivalence class to occur in the natural lexicographic ordering given by the ordered (first-to-last) set $\{a, b, c, \overline{a}, \overline{b}, \overline{c}\}$.*

Example 3.1 The words $abac\overline{b}\overline{c}$ and $\overline{b}\overline{c}abac$ are cyclically equivalent.

Example 3.2 The word $\overline{c}a\overline{b}\overline{a}\overline{b}\overline{c}$ is CP-equivalent to $aabc\overline{b}\overline{c}$. Moreover, $aabc\overline{b}\overline{c}$ is in R-CP form.

Notation $w \overset{CP}{\sim} v$ will indicate that w and v are CP-equivalent.

Remark 3.1 *CP-equivalence implies Whitehead equivalence. That is, if*

[5]In our "length-6 computations," we make use of the fact that we can generate all length-6 minimal words by applying Whitehead automorphisms to words whose first letter is "a," whose second letter is "a" or "b,"and contain no occurrences of $x\overline{x}$ or $\overline{x}x$.

[6]In fact one such equivalence class(for words of length 6) contains 1968 members.

[7]**C**yclically equivalent after Whitehead Type I automorphisms (**P**ermutations)

$w \overset{CP}{\sim} v$, *then* $w \sim v$.
Proof If $w = w_1 \cdots w_n \in F(x_1, \ldots, x_n)$ and $w_1 = x_j$ then $w(\{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_{j-2}}, \overline{x_{j-1}}, \overline{x_{j+1}}, \overline{x_{j+2}} \ldots, \overline{x_n}\}, x_j) = w_2 \cdots w_n w_1$. The result now follows by induction. ■

Remark 3.2 *R-CP words always begin with a string of one or more consecutive "a's". Moreover, this string is always maximal, in the sense that no longer strings of consecutive occurrences of a single letter can occur in the same CP-class.*
Proof If an R-CP word w did not begin with a maximal string of a's, a permutation (which could easily be chosen to be inverse-preserving) could be applied to any cyclically-equivalent word beginning with a maximal same-letter string to get a word CP-equivalent to w and occurring ahead of it in the lexicographic ordering. ■

We can now proceed with our classification.

Equivalence Classes of Minimal Words of Length at Most 6

Length	Class	Card. of Class	R-CP elements
1	1	6	a
2	1	6	aa
3	1	6	aaa
4	1	6	$aaaa$
	2	24	$ab\overline{a}\overline{b}$
	3	96	$aabb$ $aba\overline{b}$
5	1	6	$aaaaa$
	2	120	$aaba\overline{b}$
	3	120	$aab\overline{a}\overline{b}$
	4	240	$aaabb$ $aab\overline{a}b$

Length	Class	Card. of Class	R-CP elements
6	1	6	$aaaaaa$
	2	72	$aaabbb$
	3	72	$aabaa\overline{b}$
	4	72	$aab\overline{a}\overline{a}\overline{b}$
	5	144	$aaaba\overline{b}$
	6	144	$aaab\overline{a}\overline{b}$
	7	144	$aabba\overline{b}$
	8	144	$aabb\overline{a}b$
	9	144	$aabb\overline{a}\overline{b}$
	10	360	$aaaabb$
			$aaab\overline{a}b$
			$aab\overline{a}\overline{a}b$
	11	1968	$aabbcc$
			$aabcb\overline{c}$
			$aabccb$
			$aabc\overline{b}c$
			$aabc\overline{b}\overline{c}$
			$abacbc$
			$abacb\overline{c}$
			$abac\overline{b}\overline{c}$
			$abca\overline{b}\overline{c}$

The first column in the table gives the length of the minimal words we are considering, the second column is simply a numbering of the equivalence classes (under automorphisms of $F(a,b,c)$), the third gives the cardinality of the appropriate equivalence class, and the rightmost column gives the R-CP element of each CP-equivalence class in the appropriate Whitehead equivalence class.[8]

The information in the table suggests several questions. Why are there no minimal words that are not of the form $x^n \quad (x \in \{a,b,c,\overline{a},\overline{b},\overline{c}\})$ for $n < 4$? Why are there no "c's" and "$\overline{c}$'s" in R-CP words of length < 6? Is the number of occurrences of a letter x and its inverse $\overline{x}$ in a minimal word

[8]Observe that it is very easy to generate all minimal-length elements of a Whitehead equivalence class using only its R-CP elements–simply write the words cyclically equivalent to the R-CP element(if the R-CP element is of length n, then there are n such words), and then apply the Whitehead Type I permutations (48 of them for F(a,b,c)).

always constant (up to allowable permutations of letters) for all words in its Whitehead equivalence class? Can we generate R-CP elements recursively by appending "a's" to R-CP elements of shorter length? We will discuss these and further questions in the sections that follow.

4 Results

Definition 4.1 *Suppose $w \in F(x_1, \dots, x_n)$ and $x \in \{x_1 \dots, x_n\}$. The sum of the number of occurrences of x and the number of occurrences of $\overline{x}$ in w is called the* **weight** *of x in w.*[9]

Theorem 4.1 *If a letter x has weight 1 in a word w, then w has minimal length 1.*

Proof Suppose $w = w_1 w_2 \cdots w_n$ is a word in which a letter $x = w_k$ has weight 1. Apply the Whitehead automorphism $T_1 = (\{w_{k-1}, \overline{x}\}, w_{k-1})$ to w. This gives $wT_1 = w_1 w_2 \cdots w_{k-2} x w_{k+1} \cdots w_n$. A pair of simple inductions on k now completes the proof. ■

Corollary 4.1 *Suppose w is a word of length $n > 1$. Then w contains no more than $\left\lfloor \frac{n}{2} \right\rfloor$ letters of nonzero weight.*

Note that Corollary 4.1 settles the question of why all minimal words of length < 4 are powers of some letter, and why there are no "c's" and "$\overline{c}$'s" in R-CP words of length < 6.

Theorem 4.2 *(Rapaport)*[10] *Suppose that w is a minimal word in $F(x_1, \dots, x_n)$, and that $x_1, \dots, x_n$ have weights $k_1, \dots, k_n$, respectively, in w. Then the letters $x_1, \dots, x_n$ have the same weights in any minimal word v equivalent to w, up to permutation of subscripts.*

Proof By Whitehead's Theorem, there exists a sequence of level (i.e. length-preserving) Whitehead automorphisms taking w to v. Whitehead Type II

[9]The weight is undefined and assumed not to exist for inverses of generators.

[10]This appears as "Theorem 4" in [2].

automorphisms act by introducing occurrences of a "special letter" x or its inverse $\overline{x}$. In doing so, they sometimes cause cancellations of previously-existing occurrences of x or $\overline{x}$. No other cancellations can result. If an automorphism is length-preserving, then the introduction of each new x or $\overline{x}$ must be balanced by a cancellation of a previously-existing x or $\overline{x}$, thus preserving weights. Type I automorphisms are just inverse-preserving permutations, so they obviously preserve weights up to permutation of subscripts. ■

The contrapositive of Rapaport's Theorem is clearly a valuable criterion for non-equivalence, and could potentially be exploited to speed up some of the code presented in Section 2.

5 Questions for Further Research

It would be especially nice to find some recursive relationships (with respect to word length) between minimal words. The following unproven conjecture would be a step in this direction.

Conjecture 5.1 *Suppose $w, v \in F(a, b, c)$ such that w and v are nonequivalent words in R-CP form. Then aw and av are nonequivalent words in R-CP form.*

If proven, this conjecture would guarantee that the number of equivalence classes is monotonically increasing with respect to word length. Additionally, it would give a nice recursive criterion for non-equivalence. One might also hope that if $w \sim v$ and w and v are both R-CP, then $aw \sim av$, but unfortunately, the R-CP elements $aabb \sim aba\overline{b}$ provide a counterexample, since $aaabb \not\sim aaba\overline{b}$.

Another useful result would be a "nice" upper bound for the minimum number of Whitehead automorphisms required to map a given word w of length m to an equivalent and minimal word v of length n. The number of Whitehead automorphisms required to reduce w to minimal length is clearly at most $m - n$ by the strict monotonicity property in Whitehead's Theorem, but we know of no good upper bound for the number of length-preserving automorphisms that must follow. An obvious, though poor, upper bound is the number of minimal words in the equivalence class of v, but it seems intuitively implausible that such a large number of automorphisms should be required. One possible approach to this problem would be an analysis of the minimum number of Whitehead automorphisms required to map a given R-CP word of length n to a word in another given CP-equivalence class.[11] One could then use a series (of length at most n) of Whitehead automorphisms like the one described in the proof for Remark 3.1, followed by a single Type I automorphism to map any CP-class representative to any other word in its CP-class.

Finally, it is hoped that the computer-based classification of minimal words in $F(a, b, c)$ can be extended to words of longer length and free groups of larger rank. Corollary 4.1 gives the information required to decide how large a rank it is necessary to consider to avoid losing generality, and the notion of CP-equivalence gives an attractive and useful way of representing minimal words. It may be possible to improve efficiency by writing code to generate all R-CP words, and then use the programs from Section 2 to partition them according to equivalence class. If this is significantly faster, it may be possible to greatly expand the classification.

While the computer programs and applications given in this paper should not be regarded as ends in themselves, they illustrate the interplay between technology and mathematics, and it is hoped that the programs, data, and results presented will be of use to other researchers.

[11]We used the programs (particularly EW2L(page 15)) from Section 2, and applied this approach to the R-CP elements found in the table on pages 18-19. The greatest number of such automorphisms that was ever needed was 3, which occurred in the case of searching for a sequence of Whitehead automorphisms taking $aabbcc$ to any member of the CP-equivalence class of $aabc\bar{b}\bar{c}$. This was much smaller than 1968, the number of minimal words in their equivalence class.

Appendix

The following series of programs finds all minimal words of lengths 3,4, and 5 in $F(a, b, c)$.

```
> mk3list:=proc() ls:={'a','b','c','A','B','C'}; ls3:={};
>  i:=1; j:=1; k:=1; while i<7 do
> ls3:=ls3 union {cat(ls[i],ls[j],ls[k])};
> k:=k+1; if k=7 then k:=1; j:=j+1;fi; if j=7 then j:=1; i:=i+1;
> fi;od;ls3;end;
>ls3:=mk3list();
> min3ls:=proc()
> lst:={};
> for p from 1 to nops(ls3) do
> x:=oldminwd(ls3[p]);
> if length(x)=3 then lst:=lst union {x};
> fi;od;lst;
> end;

> mk4list:=proc() ls:={'a','b','c','A','B','C'}; ls4:={}; i:=1; j:=1;
> k:=1;l:=1; while i<7 do
> ls4:=ls4 union {cat(ls[i],ls[j],ls[k],ls[l])};
> l:=l+1; if l=7 then l:=1; k:=k+1;fi;if k=7 then k:=1; j:=j+1;fi;
> if j=7 then
> j:=1; i:=i+1; fi;od;ls4;end;
> ls4:=mk4list();
> min4ls:=proc()lst:={};
> for p from 1 to nops(ls4) do
> x:=oldminwd(ls4[p]);
> if length(x)=4 then lst:=lst union {x};
> fi;od;lst;
> end;

> mk5list:=proc() ls:={'a','b','c','A','B','C'}; ls5:={}; i:=1; j:=1;
> k:=1;l:=1; m:=1; while i<7 do
> ls5:=ls5 union {cat(ls[i],ls[j],ls[k],ls[l],ls[m])};
> m:=m+1; if m=7 then m:=1; l:=l+1;fi;if l=7 then l:=1; k:=k+1;fi;
> if k=7 then k:=1; j:=j+1;fi; if j=7 then j:=1; i:=i+1; fi;od;ls5;end;
> ls5:=mk5list();
> min5lsproc:=proc()
> lst:={};
> for p from 1 to nops(ls5) do
> x:=oldminwd(ls5[p]);
> if length(x)=5 then lst:=lst union {x}; fi;od;lst;end;
```

The next programs generate a list of some minimal words of length 6. While this list is not complete, all minimal words of length 6 can be generated by applying sequences of Whitehead automorphisms to words on this list.

```
> fun6lsproc:=proc()
> ls:={'a','A','b','B','c','C'};
> ls6:={};
> i:=1; j:=1; k:=1; l:=1; m:=1;
> while i<5 do
> ls6:=ls6 union {cat('a',ls[i],ls[j],ls[k],ls[l],ls[m])};
> m:=m+1;
> if m=2 then m:=3; fi;
> if frac(m/2)=0 then
>     if m=l+1 then m:=m+1; fi;
> elif m=m-1 then m:=m+1; fi;
> if m>6 then l:=l+1; m:=1;
>     if l=2 then m:=2; fi;
> fi;
>
> if frac(l/2)=0 then
>     if l=k+1 then l:=l+1; fi;
> elif l=k-1 then l:=l+1; fi;
> if l>6 then l:=1; k:=k+1;
>     if k=2 then l:=2; fi;
> fi;
>
> if frac(k/2)=0 then
>     if k=j+1 then k:=k+1; fi;
> elif k=j-1 then k:=k+1; fi;
> if k>6 then k:=1; j:=j+1;
>     if j=2 then k:=2; fi;
> fi;
>
> if frac(j/2)=0 then
>     if j=i+1 then j:=j+1; fi;
> elif j=i-1 then j:=j+1; fi;
> if j>6 then j:=1; i:=i+1; fi;
>
> if i=2 then i:=3; fi;
> od;
> ls6;
> end;
> pls6:=fun6lsproc();
> min6lsproc:=proc()
> lst:={};
> for p from 1 to nops(pls6) do
> x:=oldminwd(pls6[p]);
> if length(x)=6 then lst:=lst union{x};
> fi; od;lst;end;
```

SIEC(Split Into Equivalence Classes) takes one argument, a list ls of words from $F(a, b, c)$. It returns a list containing the distinct equivalence classes of minimal words equivalent the words in ls.

```
> SIEC:=proc(ls)
> ls1:=ls;lsn:={};
> while nops(ls1)>0  do
> x:=EWL(ls1[1]);lsn:={x} union lsn; ls1:=ls1 minus x;
> od; lsn;end;
```

References

[1] P.J. Higgins and R.C. Lyndon. Equivalence of elements under automorphisms of a free group. *J. London Math. Soc.*, pages 8: 254–258, 1974.

[2] E.S. Rapaport. On free groups and their automorphisms. *Acta. Math.*, pages 99:139–163, 1958.

[3] J.H.C. Whitehead. On certain sets of elements in a free group. *Proceedings of the London Math. Society*, pages 41:48–56, 1936.

[4] J.H.C. Whitehead. On equivalent sets of elements in a free group. *Ann. of Math.*, pages 37:782–800, 1936.