InfiniteGeneration.nb

T. Church
August 2009

This notebook is a supplement to the paper "Infinite generation of the kernels of the Magnus and Burau representations", by Thomas Church and Benson Farb. In that paper, a homomorphism Phi is defined on the kernel of the Burau representation. This notebook implements the computation of Phi for a particular element found by Bigelow.

We are very grateful to W. Goldman for making his notebook FreeGroupAutos.nb available; it was this notebook that gave us the idea that these calculations might be possible, and was also an excellent introduction to programming in Mathematica.

To initialize the methods and run the program, select Evaluation > Evaluate Notebook. The whole computation takes approximately 30 seconds on a midrange personal computer. The final answer is not formatted nicely, but it can still easily be compared with the element displayed in Appendix A of "Infinite generation...".

Functions: operations in free groups

```
Off[General::"spell1"]; Off[General::"spell"];
Off[Syntax::"tsntxi"]; Off[Solve::"svars"]
```

Words in the free group F_m are represented by lists of nonzero integers from - m to m. A positive number i represents x_i; a negative number - i represents x_i^{-1}. These lists are always maintained in reduced form, meaning that i and -i are never adjacent.

- Conc takes in a sequence of words and concatenates them, reducing if cancellation occurs.

```
Conc[a_List] := a
Conc[{}, {}] := {}
Conc[{}, a_] := a
Conc[a_, {}] := a
Conc[a_List, b_List] :=
 If[Last[a] + First[b] == 0, Conc[Drop[a, -1], Drop[b, 1]], Join[a, b]] /;
  Length[a] > 0 && Length[b] > 0
Conc[a_List, b_List, c__List] := Conc[Conc[a, b], c]
```

- BInverse computes the inverse of a word, taking advantage of the fact that all our words are "braid words", i.e. a conjugate of a generator, so we only need invert the middle term.

```
BInverse[{}] := {}
BInverse[a_List] :=
 ReplacePart[a, (Length[a] + 1) / 2 → -a[[(Length[a] + 1) / 2]]] /; OddQ[Length[a]]
```

Automorphisms are represented by a list of n words, which are the images of x_1, ... x_n respectively. No attempt is made to check that the endomorphism defined is in fact an isomorphism.

- ApplyAuto applies an automorphism to a word.

```
ApplyAuto[A_List, {}] := {}
ApplyAuto[A_List, w_List ] :=
 Apply[Conc,
  Map[
   (If[# > 0,
      A[[#]],
      BInverse[A[[-#]]]]) &
   , w]]
```

ComposeAuto composes two or more automorphisms.

```
ComposeAuto[A1_, A2_] := Map[ApplyAuto[A1, #] &, A2]
ComposeAuto[A1_, A2_, X__] := ComposeAuto[A1, ComposeAuto[A2, X]]
```

Functions: computing Phi

The next two functions together take a word lying in K (the kernel of the natural map F_n -> Z) and puts it in terms of the basis x_{i,k}=x_1^k x_i x_1^{-k-1} for i != 1. This is encoded as {True, i, k} for x_{i,k} and {False, i, k} for x_{i,k}^{-1}.

- Knormalform iterates through the list, rewriting each term as it goes. The accumulator acc holds the current value of k.

```
Knormalform[{l_List, acc_Integer}, m_Integer] := With[{pos = m > 0},
  {If[Abs[m] == 1, l,
    Append[l, {pos, Abs[m], acc - If[pos, 0, 1]}]]
   , acc + If[pos, 1, -1]}]
```

- heckle applies Knormalform to each term of the word in turn.

```
heckle[l_List] := First[Fold[Knormalform, {{}, 0}, l]]
```

We encode elements of K^ab as arrays indexed by i and by k. Since array indices must be positive but k can be any integer, we encode it using an invertible function Z -> N.

```
Code[k_] := If[k > 0, 2 * k, 1 - 2 * k]
Decode[m_] := If[EvenQ[m], m / 2, (1 - m) / 2]
```

- AddTerm takes an element of K^ab and a single basis element and adds them.

```
AddTerm[sa_SparseArray, {isPos_, i_, k_}] :=
 ReplacePart[sa, {i, Code[k]} → sa[[i, Code[k]]] + If[isPos, 1, -1]]
```

- AddUp takes a word in the basis x_ {i, k} and computes its class in K^ab by adding up the basis elements.

```
AddUp[path_List] := Fold[AddTerm, SparseArray[{6, 100} → 0], path]
```

For any free group K, given an element x of [K, K], its class in [K, K]/[K, [K, K]] = /\^ 2 K^ab may be computed as follows. Write x in terms of a basis for K, and let its first term by y. Find the first occurence of y^{-1} and write x = yzy^{-1}w. Now x and zw differ by [y,z], which corresponds to [y] /\ [z] in /\^2 K^ab. Thus we have x = [y] /\ [z] + zw, and zw has fewer terms than x; thus this algorithm terminates.

In our case, we represent /\^2 K^ab by an array, so that the entry in position {i_1,k_1,i_2,k_2} is the coefficient of x_{i_1,k_1} /\ x_{i_2,k_2}. This array is not skew-symmetric and thus each term appears twice; we apply skew-symmetry later in wedgenormalform.

```
reduce[{}, s_SparseArray] := s
reduce[{
   {bool_, i_Integer, k_Integer},
   {bool2_, i2_Integer, k2_Integer}
  }, s_SparseArray] := s /; bool2 == Not[bool] && i2 == i && k2 == k
reduce[{
   {bool_, i_Integer, k_Integer},
   {bool2_, i2_Integer, k2_Integer},
   b__},
  s_SparseArray] := reduce[{b}, s] /; bool2 == Not[bool] && i2 == i && k2 == k
reduce[{
   {bool_, i_Integer, k_Integer},
   a__,
   {bool2_, i2_Integer, k2_Integer},
   b___},
  s_SparseArray] :=
 reduce[{a, b},
   ReplacePart[s, {i, Code[k]} →
     If[bool,
      s[[i, Code[k]]] + AddUp[{a}],
      s[[i, Code[k]]] - AddUp[{a}]]]] /; bool2 == Not[bool] && i2 == i && k2 == k
```

- unwrap is used below to flatten rules and decode the index k

```
unwrap[Rule[{a_, b_, c_, d_}, n_]] := {a, Decode[b], c, Decode[d], n}
```

- Psi reduces a word in K, written in terms of the basis x_{i, k}, to its class in /\^ 2 K^{ab}. It returns a list of terms of the form {i_1,k_1,i_2,k_2,n}, which represents n times x_{i_1,k_1} /\ x_{i_2,k_2}.

```
Psi[h_List] := Map[unwrap, ArrayRules[
    reduce[h, SparseArray[{6, 100, 6, 100} → 0]]]]
```

- wedgenormalform puts terms into the normal form x_ {i_ 1, k_ 1}/\ x_ {i_ 2, k_ 2} with (i_ 1, k_ 1) < (i_ 2, k_ 2).

```
wedgenormalform[{a_, b_, c_, d_, k_}] :=
 Switch[Order[{a, b}, {c, d}],
  1, {a, b, c, d, k},
  -1, {c, d, a, b, -k},
  0, {}]

negate[{a_, b_, c_, d_, k_}] := {a, b, c, d, -k}
```

- sortbysecond is a custom sorting order for terms in /\^ 2 K^{ab}.

```
sortbysecond[a_, b_] :=
 If[Order[a[[{3, 4, 1, 2}]], b[[{3, 4, 1, 2}]]] == -1, False, True]
```

- combine takes two words x and y in K, formatted like the output of Psi, and returns xy^{-1} as an ordered list of terms in normal form, possibly with repetitions.

```
combine[l1_, l2_] :=
 Sort[
  DeleteCases[
   Map[wedgenormalform, Join[l1, Map[negate, l2]]]
   , {}], sortbysecond]
```

- eliminateDuplicates iterates through such a list and combines repeated terms. It "throws" an exception if the list is not sorted.

```
eliminateDuplicates[{}, acc_] := acc
eliminateDuplicates[{a_}, acc_] := Append[acc, a]
eliminateDuplicates[{a_, b_, y___}, acc_List] :=
 Switch[Order[a[[{3, 4, 1, 2}]], b[[{3, 4, 1, 2}]]],
  1, eliminateDuplicates[{b, y}, Append[acc, a]],
  -1, exception[a, b],
  0, eliminateDuplicates[{ReplacePart[a, {5} → a[[5]] + b[[5]]], y}, acc]]
```

- clean combines two lists as above, combines repeated terms, and removes those terms whose coefficient is 0.

```
clean[l1_, l2_] :=
 DeleteCases[eliminateDuplicates[combine[l1, l2], {}], {_, _, _, _, 0}]
```

Computation: computing Bigelow's automorphism

We hard-code the Artin generators for B_ 6.

```
id = {{1}, {2}, {3}, {4}, {5}, {6}};
S1 = {{1, 2, -1}, {1}, {3}, {4}, {5}, {6}};
S2 = {{1}, {2, 3, -2}, {2}, {4}, {5}, {6}};
S3 = {{1}, {2}, {3, 4, -3}, {3}, {5}, {6}};
S4 = {{1}, {2}, {3}, {4, 5, -4}, {4}, {6}};
S5 = {{1}, {2}, {3}, {4}, {5, 6, -5}, {5}};
S1i = {{2}, {-2, 1, 2}, {3}, {4}, {5}, {6}};
S2i = {{1}, {3}, {-3, 2, 3}, {4}, {5}, {6}};
S3i = {{1}, {2}, {4}, {-4, 3, 4}, {5}, {6}};
S4i = {{1}, {2}, {3}, {5}, {-5, 4, 5}, {6}};
S5i = {{1}, {2}, {3}, {4}, {6}, {-6, 5, 6}};
```

Bigelow' s word is [W_ 1^{-1} S_ 3^{-1} W_ 1, W_ 2^{-1} S_ 3^{-1} W_ 2].

```
W1 = ComposeAuto[S4, S5i, S2i, S1] ;
W1i = ComposeAuto[S1i, S2, S5, S4i] ;
W2 = ComposeAuto[S4i, S5, S5, S2, S1i, S1i];
W2i = ComposeAuto[S1, S1, S2i, S5i, S5i, S4];
C1 = ComposeAuto[W1i, S3, W1];
C1i = ComposeAuto[W1i, S3i, W1];
C2 = ComposeAuto[W2i, S3, W2];
C2i = ComposeAuto[W2i, S3i, W2];

Bigelow = ComposeAuto[C1i, C2i, C1, C2];
```

```
V = Map[Length, Bigelow]
```

```
{6975, 9841, 833, 833, 9841, 6143}
```

Computation: simplifying and computing Phi

We use the following simplification : if a and b are generators (for us they will be x_1 and x_3) so that f (a) = gag^{-1} and f (b) = hbh^{-1}, where g and h both lie in [K, K], then we have f(aB)bA=gaGhBHbA. This is g - tg + th - h = (1-t) [g] - [h]. Thus we first compute clean(g,h).

```
g = Bigelow[[2]][[1 ;; 4920]];
```

```
h = Bigelow[[1]][[1 ;; 3487]];
```

We can check that g and h in fact lie in [K, K] by computing AddUp[heckle[g]] and AddUp[heckle[h]]; we get SparseArray[<0>,{6,100}], representing 0 in K^{ab}.

```
$IterationLimit = Infinity;
GminusH = clean[Psi[heckle[g]], Psi[heckle[h]]];
```

- timesT gives the diagonal action of t on /\^ 2 K^{ab} by multiplication.

```
timesT[{a_, b_, c_, d_, k_}] := {a, b + 1, c, d + 1, k}
```

Now the final answer Phi (Big) (y_ 2) = (1 - t)[g] - [h]:

```
answer = clean[GminusH, Map[timesT, GminusH]]
```

```
{{2, -3, 2, -2, -1}, {2, -3, 2, -1, 1}, {2, -3, 2, 0, -1}, {2, -2, 2, 0, -1},
 {2, -1, 2, 0, 1}, {2, -2, 2, 1, 1}, {2, -1, 2, 1, 1}, {2, 0, 2, 2, -2}, {2, 1, 2, 3, 1},
 {2, 2, 2, 3, 1}, {2, 3, 2, 4, -1}, {2, -3, 3, -4, 1}, {2, -2, 3, -4, -1},
```

{2, −3, 3, −3, −1}, {2, −1, 3, −3, 1}, {2, −2, 3, −2, 1}, {2, −1, 3, −2, −1},
{2, −3, 3, −1, 1}, {2, 0, 3, −1, −1}, {2, 1, 3, −1, 1}, {2, 2, 3, −1, −1},
{2, −2, 3, 0, −2}, {2, 3, 3, 0, 1}, {3, −1, 3, 0, 1}, {2, −1, 3, 1, 2},
{3, −1, 3, 1, −1}, {2, 0, 3, 2, −2}, {2, 4, 3, 2, −1}, {3, −1, 3, 2, 1},
{2, 1, 3, 3, 1}, {2, 4, 3, 3, 1}, {3, 0, 3, 3, −1}, {3, 1, 3, 3, 1}, {3, 2, 3, 3, −1},
{2, −3, 4, −3, 1}, {2, −2, 4, −3, −1}, {2, −3, 4, −2, −1}, {2, −1, 4, −2, 1},
{2, −2, 4, −1, 1}, {2, −1, 4, −1, −1}, {2, −3, 4, 0, 1}, {2, 0, 4, 0, −1},
{2, 1, 4, 0, 1}, {2, 2, 4, 0, −1}, {3, 0, 4, 0, −1}, {3, 1, 4, 0, 1}, {3, 2, 4, 0, −1},
{2, −2, 4, 1, −2}, {2, 3, 4, 1, 1}, {3, −1, 4, 1, 1}, {3, 3, 4, 1, 1}, {4, 0, 4, 1, 1},
{2, −1, 4, 2, 2}, {3, −1, 4, 2, −1}, {3, 3, 4, 2, −1}, {4, 0, 4, 2, −1},
{2, 0, 4, 3, −2}, {2, 4, 4, 3, −1}, {3, −1, 4, 3, 1}, {3, 3, 4, 3, 1}, {4, 0, 4, 3, 1},
{2, 1, 4, 4, 1}, {2, 4, 4, 4, 1}, {3, 0, 4, 4, −1}, {3, 1, 4, 4, 1}, {3, 2, 4, 4, −1},
{4, 1, 4, 4, −1}, {4, 2, 4, 4, 1}, {4, 3, 4, 4, −1}, {2, −3, 5, −3, 1},
{2, −2, 5, −3, −1}, {2, −3, 5, −2, 1}, {2, −2, 5, −2, −1}, {2, 0, 5, −2, 1},
{3, −4, 5, −2, −1}, {3, −3, 5, −2, 1}, {3, −1, 5, −2, −1}, {4, −3, 5, −2, −1},
{4, −2, 5, −2, 1}, {4, 0, 5, −2, −1}, {5, −3, 5, −2, −1}, {2, −3, 5, −1, −2},
{2, −1, 5, −1, 1}, {2, 0, 5, −1, 1}, {2, 1, 5, −1, −1}, {3, −4, 5, −1, 1},
{3, −2, 5, −1, −1}, {3, 0, 5, −1, 2}, {4, −3, 5, −1, 1}, {4, −1, 5, −1, −1},
{4, 1, 5, −1, 2}, {5, −3, 5, −1, 1}, {2, −3, 5, 0, 1}, {2, −2, 5, 0, 2},
{2, −1, 5, 0, −2}, {2, 0, 5, 0, −1}, {2, 2, 5, 0, −1}, {3, −3, 5, 0, −1},
{3, −2, 5, 0, 1}, {3, 0, 5, 0, −1}, {3, 1, 5, 0, −1}, {3, 2, 5, 0, −1},
{4, −2, 5, 0, −1}, {4, −1, 5, 0, 1}, {4, 1, 5, 0, −1}, {4, 2, 5, 0, −1},
{4, 3, 5, 0, −1}, {5, −1, 5, 0, 1}, {2, −3, 5, 1, −1}, {2, −1, 5, 1, −1},
{2, 0, 5, 1, 1}, {2, 1, 5, 1, 1}, {2, 3, 5, 1, 1}, {3, −1, 5, 1, 1}, {3, 0, 5, 1, −1},
{3, 1, 5, 1, 2}, {3, 3, 5, 1, 1}, {4, 0, 5, 1, 1}, {4, 1, 5, 1, −1}, {4, 2, 5, 1, 2},
{4, 4, 5, 1, 1}, {5, −2, 5, 1, −1}, {5, 0, 5, 1, −1}, {2, −2, 5, 2, 1},
{2, −1, 5, 2, −1}, {2, 0, 5, 2, 2}, {2, 2, 5, 2, −1}, {2, 3, 5, 2, 1},
{3, −1, 5, 2, −1}, {3, 2, 5, 2, −1}, {4, 0, 5, 2, −1}, {4, 3, 5, 2, −1},
{5, −1, 5, 2, 1}, {5, 0, 5, 2, −2}, {5, 1, 5, 2, 1}, {2, 1, 5, 3, −1},
{2, 2, 5, 3, −1}, {2, 4, 5, 3, −1}, {3, 0, 5, 3, 1}, {4, 1, 5, 3, 1}, {5, 1, 5, 3, 1},
{5, 2, 5, 3, 1}, {2, 2, 5, 4, 1}, {2, 3, 5, 4, 1}, {3, 1, 5, 4, −1}, {3, 3, 5, 4, 1},
{4, 2, 5, 4, −1}, {4, 4, 5, 4, 1}, {5, 2, 5, 4, −1}, {5, 3, 5, 4, −1},
{2, 3, 5, 5, −1}, {3, 2, 5, 5, 1}, {3, 3, 5, 5, −1}, {4, 3, 5, 5, 1}, {4, 4, 5, 5, −1},
{5, 3, 5, 5, 1}, {2, −3, 6, −3, −1}, {2, −2, 6, −3, 1}, {5, −2, 6, −3, −1},
{5, −1, 6, −3, 1}, {2, −3, 6, −2, 1}, {2, −1, 6, −2, −1}, {5, −2, 6, −2, 1},
{5, 0, 6, −2, −1}, {2, −3, 6, −1, 1}, {2, −2, 6, −1, −1}, {2, 0, 6, −1, 1},
{3, −4, 6, −1, −1}, {3, −3, 6, −1, 1}, {3, −1, 6, −1, −1}, {4, −3, 6, −1, −1},

{4, −2, 6, −1, 1}, {4, 0, 6, −1, −1}, {5, −3, 6, −1, −1}, {5, 1, 6, −1, 1},
{6, −3, 6, −1, 1}, {6, −2, 6, −1, −1}, {2, −3, 6, 0, −1}, {2, −2, 6, 0, −1},
{2, −1, 6, 0, 1}, {2, 0, 6, 0, 2}, {2, 1, 6, 0, −2}, {2, 2, 6, 0, 1}, {3, −3, 6, 0, 1},
{3, −2, 6, 0, −1}, {3, −1, 6, 0, −1}, {3, 0, 6, 0, 3}, {3, 1, 6, 0, −1},
{3, 2, 6, 0, 1}, {4, −2, 6, 0, 1}, {4, −1, 6, 0, −1}, {4, 0, 6, 0, −1},
{4, 1, 6, 0, 3}, {4, 2, 6, 0, −1}, {4, 3, 6, 0, 1}, {5, 0, 6, 0, −1}, {5, 1, 6, 0, 1},
{5, 2, 6, 0, −2}, {6, −2, 6, 0, −1}, {2, −3, 6, 1, 1}, {2, −2, 6, 1, 1},
{2, 0, 6, 1, −1}, {2, 1, 6, 1, −1}, {2, 3, 6, 1, −1}, {3, −1, 6, 1, −1},
{3, 0, 6, 1, 1}, {3, 1, 6, 1, −2}, {3, 3, 6, 1, −1}, {4, 0, 6, 1, −1}, {4, 1, 6, 1, 1},
{4, 2, 6, 1, −2}, {4, 4, 6, 1, −1}, {5, −2, 6, 1, 1}, {5, −1, 6, 1, 1},
{5, 2, 6, 1, 1}, {5, 3, 6, 1, 1}, {6, −1, 6, 1, 1}, {6, 0, 6, 1, 2}, {2, −2, 6, 2, −1},
{2, −1, 6, 2, −1}, {2, 2, 6, 2, 1}, {3, −1, 6, 2, 1}, {3, 2, 6, 2, 1}, {3, 3, 6, 2, 1},
{4, 0, 6, 2, 1}, {4, 3, 6, 2, 1}, {4, 4, 6, 2, 1}, {5, −1, 6, 2, −1}, {5, 1, 6, 2, 1},
{5, 2, 6, 2, −1}, {5, 4, 6, 2, −1}, {6, 0, 6, 2, −2}, {6, 1, 6, 2, −1}, {2, 0, 6, 3, 2},
{2, 4, 6, 3, 1}, {3, −1, 6, 3, −1}, {3, 3, 6, 3, −1}, {4, 0, 6, 3, −1},
{4, 4, 6, 3, −1}, {5, 0, 6, 3, −1}, {5, 2, 6, 3, −1}, {5, 5, 6, 3, 1}, {6, 0, 6, 3, 1},
{6, 2, 6, 3, 1}, {2, 1, 6, 4, −1}, {2, 2, 6, 4, −1}, {2, 4, 6, 4, −1},
{3, 0, 6, 4, 1}, {4, 1, 6, 4, 1}, {5, 1, 6, 4, 1}, {5, 2, 6, 4, 1}, {5, 4, 6, 4, 1},
{5, 5, 6, 4, −1}, {6, 1, 6, 4, −1}, {2, 3, 6, 5, 1}, {3, 2, 6, 5, −1}, {3, 3, 6, 5, 1},
{4, 3, 6, 5, −1}, {4, 4, 6, 5, 1}, {5, 3, 6, 5, −1}, {6, 3, 6, 5, 1}, {6, 4, 6, 5, −1}}