Kan

A package for Induced Category Actions

Version 1.27

20/10/2016

Anne Heyworth
Chris Wensley

Chris Wensley Email: c.d.wensley@bangor.ac.uk
Homepage: http://pages.bangor.ac.uk/~mas023/
Address: School of Computer Science, Bangor University,
Dean Street, Bangor, Gwynedd, LL57 1UT, U.K.

Abstract

The Kan package was originally implemented in 1996 using the GAP 3 language, to compute induced actions of categories, when the first author was studying for a Ph.D. in Bangor.

This reduced version only provides functions for the computation of normal forms of representatives of double cosets of finitely presented groups.

Kan became an accepted GAP package in May 2015.

Bug reports, suggestions and comments are, of course, welcome. Please submit an issue at `https://github.com/gap-packages/kan/issues/` or send an email to the second author at `c.d.wensley@bangor.ac.uk`.

Copyright

Acknowledgements

This documentation was prepared with the GAPDoc package of Frank Lübeck and Max Neunhöffer.

The procedure used to mount new releases on GitHub uses the packages GitHubPagesForGAP and ReleaseTools of Max Horn.

Contents

Chapter 1

Introduction

The Kan package started out as part of Anne Heyworth's thesis [Hey99], and was designed to compute induced actions of categories (see also [BH00]).

This version of Kan only provides functions for the computation of normal forms of representatives of double cosets of finitely presented groups, and is made available in support of the paper [BGHW06]. Existing methods for computing double cosets in GAP are described in [Lin91].

The package is loaded into GAP with the command

Example
```
gap> LoadPackage( "kan" );
```

The package may be obtained as a compressed tar file `kan-1.27.tar.gz` by ftp from one of the following sites:

- any GAP archive, e.g. `http://www.gap-system.org/Packages/packages.html`;
- the package GitHub repository: `https://gap-packages.github.io/kan`.

Some of the functions in the automata package are used to compute word acceptors and regular expressions for the languages they accept.

The kbmag package is also used to compute a word acceptor of a group `G` when `G` has no finite rewriting system. If kbmag is not available (the user is not on a UNIX system, or the C-programs have not been compiled), the file `dckbmag.gi` will not be read, so methods for the functions detailed in section 2.4.1 will not be available.

Once the package is loaded, it is possible to check the installation is correct by running a test file of the manual examples with the following command. (The test file itself is `tst/fulltest.tst` or `tst/parttest.tst`, depending whether or not kbmag is available.)

Example
```
gap> ReadPackage( "kan", "tst/testall.g" );
#I  Testing /Applications/gap/my-dev/pkg/kan/tst/fulltest.tst
#I  No errors detected while testing package kan
true
```

The information parameter `InfoKan` takes default value `0`. When raised to a higher value, additional information is printed out.

Once the package is loaded, the manual `doc/manual.pdf` can be found in the documentation folder. The `html` versions, with or without MathJax, may be rebuilt as follows.

```
Example
gap> InfoLevel( InfoKan );
0
gap> ReadPackage( "kan, "makedoc.g" );
```

Please send bug reports, suggestions and other comments to the second author, or use the GitHub issue tracker at `https://github.com/gap-packages/kan/issues/new`.

Additional information can be found on the *Computational Higher-dimensional Discrete Algebra* website at `http://pages.bangor.ac.uk/~mas023/chda/`.

Chapter 2

Double Coset Rewriting Systems

The Kan package provides functions for the computation of normal forms for double coset representatives of finitely presented groups. The first version of the package was released to support the paper [BGHW06], which describes the algorithms used in this package.

2.1 Rewriting Systems

2.1.1 KnuthBendixRewritingSystem

▷ `KnuthBendixRewritingSystem(`*`grp, gensorder, ordering, alph`*`)` (operation)
▷ `ReducedConfluentRewritingSystem(`*`grp, gensorder, ordering, limit`*`)` (operation)
▷ `DisplayRwsRules(`*`rws`*`)` (operation)

Methods for `KnuthBendixRewritingSystem` and `ReducedConfluentRewritingSystem` are supplied which apply to a finitely presented group. These start by calling `IsomorphismFpMonoid` and then work with the resulting monoid. The parameter `gensorder` will normally be `"shortlex"` or `"wreath"`, while `ordering` is an integer list for reordering the generators, and `alph` is an alphabet string used when printing words. A *partial* rewriting system may be obtained by giving a `limit` to the number of rules calculated. As usual, A,B denote the inverses of a,b.

In the example the generators are by default ordered $[A,a,B,b]$, so the list `L1` is used to specify the order `[a,A,b,B]` to be used with the shortlex ordering. Specifying a limit `0` means that no limit is prescribed.

Example
```
gap> G1 := FreeGroup( 2 );;
gap> L1 := [2,1,4,3];;
gap> order := "shortlex";;
gap> alph1 := "AaBb";;
gap> rws1 := ReducedConfluentRewritingSystem( G1, L1, order, 0, alph1 );
Rewriting System for Monoid( [ f1, f1^-1, f2, f2^-1 ], ... ) with rules
[ [ f1*f1^-1, <identity ...> ], [ f1^-1*f1, <identity ...> ],
  [ f2*f2^-1, <identity ...> ], [ f2^-1*f2, <identity ...> ] ]
gap> DisplayRwsRules( rws1 );;
[ [ Aa, id ], [ aA, id ], [ Bb, id ], [ bB, id ] ]
```

2.2 Example 1 – free product of two cyclic groups

2.2.1 DoubleCosetRewritingSystem

▷ DoubleCosetRewritingSystem(*grp, genH, genK, rws*) (function)
▷ IsDoubleCosetRewritingSystem(*dcrws*) (property)

A *double coset rewriting system* for the double cosets $H\backslash G/K$ requires as data a finitely presented group $G =$grp; generators genH, genK for subgroups H,K; and a rewriting system rws for G.

A simple example is given by taking G to be the free group on two generators a,b, a cyclic subgroup H with generator a^6, and a second cyclic subgroup K with generator a^4. (Similar examples using different powers of a are easily constructed.) Since gcd(6,4)=2, we have $Ha^2K = HK$, so the double cosets have representatives $[HK, HaK, Ha^iba^jK, Ha^ibwba^jK]$ where $i \in [0..5]$, $j \in [0..3]$, and w is any word in a,b.

In the example the free group G is converted to a four generator monoid with relations defining the inverse of each generator, [[Aa,id],[aA,id],[Bb,id],[bB,id]], where id is the empty word. The initial rules for the double coset rewriting system comprise those of G plus those given by the generators of H,K, which are $[[Ha^6,H],[a^4K,K]]$. In the complete rewrite system new rules involving H or K may arise, and there may also be rules involving both H and K.

For this example,

- there are two H-rules, $[[Ha^4,HA^2],[HA^3,Ha^3]]$,
- there are two K-rules, $[[a^3K,AK],[A^2K,a^2K]]$,
- and there are two H-K-rules, $[[Ha^2K,HK],[HAK,HaK]]$.

Here is how the computation may be performed.

```
Example
gap> genG1 := GeneratorsOfGroup( G1 );;
gap> genH1 := [ genG1[1]^6 ];;
gap> genK1 := [ genG1[1]^4 ];;
gap> dcrws1 := DoubleCosetRewritingSystem( G1, genH1, genK1, rws1 );;
gap> IsDoubleCosetRewritingSystem( dcrws1 );
true
gap> DisplayRwsRules( dcrws1 );;
G-rules:
[ [ Aa, id ], [ aA, id ], [ Bb, id ], [ bB, id ] ]
H-rules:
[ [ Haaaa, HAA ],
  [ HAAA, Haaa ] ]
K-rules:
[ [ aaaK, AK ],
  [ AAK, aaK ] ]
H-K-rules:
[ [ HaaK, HK ],
  [ HAK, HaK ] ]
```

2.2.2 WordAcceptorOfReducedRws

▷ `WordAcceptorOfReducedRws(`*rws*`)` (attribute)
▷ `WordAcceptorOfDoubleCosetRws(`*rws*`)` (attribute)
▷ `IsWordAcceptorOfDoubleCosetRws(`*aut*`)` (property)

Using functions from the automata package, we may

- compute a *word acceptor* for the rewriting system of G;
- compute a *word acceptor* for the double coset rewriting system;
- test a list of words to see whether they are recognised by the automaton;
- obtain a rational expression for the language of the automaton.

Example

```
gap> waG1 := WordAcceptorOfReducedRws( rws1 );
Automaton("det",6,"aAbB",[ [ 1, 4, 1, 4, 4, 4 ], [ 1, 3, 3, 1, 3, 3 ], [ 1, 2,\
 2, 2, 1, 2 ], [ 1, 1, 5, 5, 5, 5 ] ],[ 6 ],[ 2, 3, 4, 5, 6 ]);;
gap> wadc1 := WordAcceptorOfDoubleCosetRws( dcrws1 );
< deterministic automaton on 6 letters with 15 states >
gap> Print( wadc1 );
Automaton("det",15,"HKaAbB",[ [ 2, 2, 2, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ],\
 [ 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 2, 1, 1, 2, 2 ], [ 2, 2, 13, 2, 10, 5, 2, 13,\
 2, 7, 11, 11, 12, 2, 2 ], [ 2, 2, 9, 2, 2, 14, 2, 9, 15, 2, 2, 2, 2, 7, 15 ],\
 [ 2, 2, 2, 2, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8 ], [ 2, 2, 3, 2, 3, 3, 3, 2, 3,\
 3, 3, 3, 3, 3 ] ],[ 4 ],[ 1 ]);;
gap> words1 := [ "HK","HaK","HbK","HAK","HaaK","HbbK","HabK","HbaK","HbaabK"];;
gap> valid1 := List( words1, w -> IsRecognizedByAutomaton( wadc1, w ) );
[ true, true, true, false, false, true, true, true, true ]
gap> lang1 := FAtoRatExp( wadc1 );
((H(aaaUAA)BUH(a(aBUB)UABUB))(a(a(aa*BUB)UB)UA(AA*BUB)UB)*(a(a(aa*bUb)Ub)UA(AA\
*bUb))UH(aaaUAA)bUH(a(abUb)UAbUb))((a(a(aa*BUB)UB)UA(AA*BUB))(a(a(aa*BUB)UB)UA\
(AA*BUB)UB)*(a(a(aa*bUb)Ub)UA(AA*bUb))Ua(a(aa*bUb)Ub)UA(AA*bUb)Ub)*((a(a(aa*BU\
B)UB)UA(AA*BUB))(a(a(aa*BUB)UB)UA(AA*BUB)UB)*(a(aKUK)UAKUK)Ua(aKUK)UAKUK)U(H(a\
aaUAA)BUH(a(aBUB)UABUB))(a(a(aa*BUB)UB)UA(AA*BUB)UB)*(a(aKUK)UAKUK)UH(aKUK)
```

2.3 Example 2 – the trefoil group

2.3.1 PartialDoubleCosetRewritingSystem

▷ `PartialDoubleCosetRewritingSystem(`*grp, Hgens, Kgens, rws, limit*`)` (operation)
▷ `WordAcceptorOfPartialDoubleCosetRws(`*grp, prws*`)` (attribute)

It may happen that, even when G has a finite rewriting system, the double coset rewriting system is infinite. This is the case with the trefoil group T with generators $[x,y]$ and relator $[x^3 = y^2]$ when the wreath product ordering is used with $X > x > Y > y$. The group itself has a rewriting system with just 6 rules.

Example

```
gap> FT := FreeGroup( 2 );;
gap> relsT := [ FT.1^3*FT.2^-2 ];;  T := FT/relsT;;
gap> genT := GeneratorsOfGroup( T );;  x := genT[1];  y := genT[2];
gap> alphT := "XxYy";;  ordT := [4,3,2,1];;  orderT := "wreath";;
gap> rwsT := ReducedConfluentRewritingSystem( T, ordT, orderT, 0, alphT );;
gap> DisplayRwsRules( rwsT );;
[ [ Yy, id ], [ yY, id ], [ xxx, yy ], [ yyx, xyy ], [ X, xxYY ], [ Yx, yxYY ]\
 ]
gap> accT := WordAcceptorOfReducedRws( rwsT );
< deterministic automaton on 4 letters with 7 states >
gap> Print( "accT = ", accT );
accT = Automaton("det",7,"yYxX",[ [ 1, 2, 1, 5, 2, 5, 5 ], [ 1, 1, 3, 3, 1, 3,\
 3 ], [ 1, 1, 1, 7, 7, 1, 6 ], [ 1, 1, 1, 1, 1, 1, 1 ] ],[ 4 ],[ 2, 3, 4, 5, 6\
, 7 ]);;
gap> langT := FAtoRatExp( accT );
(yxUx)((xyUy)x)*((xyUy)(yy*U@)Ux(YY*U@)UYY*U@)Uy(yy*U@)UYY*U@
gap> IsRecognizedByAutomaton( accT, "X" );
false
gap> IsRecognizedByAutomaton( accT, "yxyxyxYY" );
true
```

In versions of Kan, from about 1.01 up to 1.21, the complementary automaton and language were returned in the example above. This error has now been rectified.

In earlier versions of Kan (in 0.95 for example) a shorter rational expression for the language was obtained from Automata. In what follows, we check that the two expressions define the same language.

Example

```
gap> alph := AlphabetOfRatExpAsList( langT );;
gap> a1 := RatExpOnnLetters( alph, [ ], [1] );;   ## y
gap> a2 := RatExpOnnLetters( alph, [ ], [2] );;   ## Y
gap> a3 := RatExpOnnLetters( alph, [ ], [3] );;   ## x
gap> a4 := RatExpOnnLetters( alph, [ ], [4] );;   ## X
gap> s1 := RatExpOnnLetters( alph, "star", a1 );; ## y*
gap> s2 := RatExpOnnLetters( alph, "star", a2 );; ## Y*
gap> a1a3 := RatExpOnnLetters( alph, "product", [ a1, a3 ] );;  ## yx
gap> u1 := RatExpOnnLetters( alph, "union", [ a1a3, a3 ] );;    ## yxUx
gap> a3a1 := RatExpOnnLetters( alph, "product", [ a3, a1 ] );;  ## xy
gap> u2 := RatExpOnnLetters( alph, "union", [ a3a1, a1 ] );;    ## xyUy
gap> u2a3 := RatExpOnnLetters( alph, "product", [ u2, a3 ] );;  ## (xyUy)x
gap> su2a3 := RatExpOnnLetters( alph, "star", u2a3 );;          ## ((xyUy)x)*
gap> u2s1 := RatExpOnnLetters( alph, "product", [ u2, s1 ] );;  ## (xyUy)y*
gap> a3s2 := RatExpOnnLetters( alph, "product", [ a3, s2 ] );;  ## xY*
gap> u3 := RatExpOnnLetters( alph, "union", [u2s1,a3s2,s2] );;
gap> prod := RatExpOnnLetters( alph, "product", [u1,su2a3,u3] );;
gap> a1s1 := RatExpOnnLetters( alph, "product", [ a1, s1 ] );;  ## yy*
gap> r := RatExpOnnLetters( alph, "union", [ prod, a1s1, s2] );
(yxUx)((xyUy)x)*((xyUy)y*UxY*UY*)Uyy*UY*
gap> AreEqualLang( langT, r );
```

```
true
```

Taking subgroups H, K to be generated by x and y respectively, the double coset rewriting system has an infinite number of H-rules. It turns out that only a finite number of these are needed to produce the required automaton. The function `PartialDoubleCosetRewritingSystem` allows a limit to be specified on the number of rules to be computed. In the listing below a limit of 20 is used, but in fact 10 is sufficient.

Example

```
gap> prwsT := PartialDoubleCosetRewritingSystem( T, [x], [y], rwsT, 20 );;
#I WARNING: reached supplied limit 20 on number of rules
gap> DisplayRwsRules( prwsT );
G-rules:
[ [ X, xxYY ], [ Yx, yxYY ], [ Yy, id ], [ yY, id ], [ xxx, yy ], [ yyx, xyy ]\
 ]
H-rules:
[ [ Hx, H ],
  [ HY, Hy ],
  [ Hyy, H ],
  [ Hyxyy, Hyx ],
  [ HyxY, Hyxy ],
  [ Hyxyxyy, Hyxyx ],
  [ Hyxxyy, Hyxx ],
  [ HyxxY, Hyxxy ],
  [ HyxyxY, Hyxyxy ],
  [ Hyxyxyxyy, Hyxyxyx ],
  [ Hyxyxxyy, Hyxyxx ],
  [ Hyxxyxyy, Hyxxyx ],
  [ HyxxyxYY, Hyxxyx ] ]
K-rules:
[ [ YK, K ],
  [ yK, K ] ]
```

This list of partial rules is then used by a modified word acceptor function.

Example

```
gap> paccT := WordAcceptorOfPartialDoubleCosetRws( T, prwsT );;
< deterministic automaton on 6 letters with 6 states >
gap> Print( paccT, "\n" );
Automaton("det",6,"HKyYxX",[ [ 2, 2, 2, 6, 2, 2 ], [ 2, 2, 1, 2, 2, 1 ], [ 2, \
2, 5, 2, 2, 5 ], [ 2, 2, 2, 2, 2, 2 ], [ 2, 2, 6, 2, 3, 2 ], [ 2, 2, 2, 2, 2, \
2 ] ],[ 4 ],[ 1 ]);;
gap> plangT := FAtoRatExp( paccT );
H(yx(yx)*x)*(yx(yx)*KUK)
gap> wordsT := ["HK", "HxK", "HyK", "HYK", "HyxK", "HyxxK", "HyyH", "HyxYK"];;
gap> validT := List( wordsT, w -> IsRecognizedByAutomaton( paccT, w ) );
[ true, false, false, false, true, true, false, false ]
```

2.4 Example 3 – an infinite rewriting system

2.4.1 KBMagRewritingSystem

▷ KBMagRewritingSystem(*fpgrp*) (attribute)
▷ KBMagWordAcceptor(*fpgrp*) (attribute)
▷ KBMagFSAtoAutomataDFA(*fsa, alph*) (operation)
▷ WordAcceptorByKBMag(*grp, alph*) (operation)
▷ WordAcceptorByKBMagOfDoubleCosetRws(*grp, dcrws*) (operation)

When the group G has an infinite rewriting system, the double coset rewriting system will also be infinite. In this case we may use the function `KBMagWordAcceptor` which calls KBMAG to compute a word acceptor for G, and `KBMagFSAtoAutomataDFA` to convert this to a deterministic automaton as used by the automata package. The resulting `dfa` forms part of the double coset automaton, together with sufficient H-rules, K-rules and H-K-rules found by the function `PartialDoubleCosetRewritingSystem`. (Note that these five attributes and operations will not be available if the kbmag package has not been loaded.)

In the following example we take a two generator group $G3$ with relators $[a^3, b^3, (a*b)^3]$, the normal forms of whose elements are some of the strings with a or a^{-1} alternating with b or b^{-1}. The automatic structure computed by KBMAG has a word acceptor with 17 states.

Example
```
gap> F3 := FreeGroup("a","b");;
gap> rels3 := [ F3.1^3, F3.2^3, (F3.1*F3.2)^3 ];;
gap> G3 := F3/rels3;;
gap> alph3 := "AaBb";;
gap> waG3 := WordAcceptorByKBMag( G3, alph3 );;
gap> Print( waG3, "\n");
Automaton("det",18,"aAbB",[ [ 2, 18, 18, 8, 10, 12, 13, 18, 18, 18, 18, 18, 18\
, 8, 17, 12, 18, 18 ], [ 3, 18, 18, 9, 11, 9, 12, 18, 18, 18, 18, 18, 18, 11, \
18, 11, 18, 18 ], [ 4, 6, 6, 18, 18, 18, 18, 18, 6, 12, 16, 18, 12, 18, 18, 18\
, 12, 18 ], [ 5, 5, 7, 18, 18, 18, 18, 14, 15, 5, 18, 18, 7, 18, 18, 18, 15, 1\
8 ] ],[ 1 ],[ 1 .. 17 ]);;
gap> langG3 := FAtoRatExp( waG3 );
((abUAb)AUbA)(bA)*(b(aU@)UB(aB)*(a(bU@)U@)U@)U(abUAb)(aU@)U((aBUB)(aB)*AUba(Ba\
)*BA)(bA)*(b(aU@)U@)U(aBUB)(aB)*(a(bU@)U@)Uba(Ba)*(BU@)UbUaUA(B(aB)*(a(bU@)UAU\
@)U@)U@
```

2.4.2 DCrules

▷ DCrules(*dcrws*) (operation)
▷ Hrules(*dcrws*) (attribute)
▷ Krules(*dcrws*) (attribute)
▷ HKrules(*dcrws*) (attribute)

We now take H to be generated by ab and K to be generated by ba. If we specify a limits of 50, 75, 100, 200 for the number of rules in a partial double coset rewrite system, we obtain lists of H-rules, K-rules and H-K-rules of increasing length. The numbers of states in the resulting automata also increase. We may deduce by hand (but not computationally – see [BGHW06]) three infinite sets of rules and a limit for the automata.

Example

```
gap> lim := 100;;
gap> genG3 := GeneratorsOfGroup( G3 );;
gap> a := genG3[1];;  b := genG3[2];;
gap> gH3 := [ a*b ];;  gK3 := [ b*a ];;
gap> rwsG3 := KnuthBendixRewritingSystem( G3, "shortlex", [2,1,4,3], alph3 );;
gap> dcrws3 := PartialDoubleCosetRewritingSystem( G3, gH3, gK3, rwsG3, lim );;
#I using PartialDoubleCosetRewritingSystem with limit 100
#I  51 rules, and 1039 pairs
#I WARNING: reached supplied limit 100 on number of rules
gap> Print( Length( Rules( dcrws3 ) ), " rules found.\n" );
101 rules found.
gap> dcaut3 := WordAcceptorByKBMagOfDoubleCosetRws( G3, dcrws3 );;
gap> Print( "Double Coset Minimalized automaton:\n", dcaut3 );
Double Coset Minimalized automaton:
Automaton("det",44,"HKaAbB",[ [ 2, 2, 2, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2\
, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2\
, 2, 2 ], [ 2, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, \
2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1 ], [ 2, 2, 2,\
 2, 3, 24, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 43, 2, 43, 2, 27, 2, 2, 2\
, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ], [ 2, 2, 2, 2, 44, 3, 29, 2\
, 8, 2, 10, 2, 12, 2, 14, 2, 16, 2, 18, 2, 20, 2, 22, 2, 2, 2, 2, 26, 2, 29, 2\
, 31, 2, 33, 2, 35, 2, 37, 2, 39, 2, 41, 2, 2 ], [ 2, 2, 2, 2, 21, 2, 2, 28, 2\
, 9, 2, 11, 2, 13, 2, 15, 2, 17, 2, 19, 2, 42, 2, 3, 2, 28, 3, 2, 7, 2, 30, 2,\
 32, 2, 34, 2, 36, 2, 38, 2, 40, 2, 2, 28 ], [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2\
, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 2, 25, 25, 2, 2, 2, 2, 2, 2, 2, 2, 2,\
 2, 2, 2, 2, 2, 2, 23, 6 ] ],[ 4 ],[ 1 ]);;
gap> dclang3 := FAtoRatExp( dcaut3 );;
gap> Print( "Double Coset language of acceptor:\n", dclang3, "\n" );
Double Coset language of acceptor:
(HbAbAbAbAbAbAbAbUHAb)(Ab)*(A(Ba(Ba)*bKUK)UK)UHbAbA(bA(bA(bA(bA(bAKUK)UK)UK)UK\
)UK)UH(A(B(aB)*(abUA)KUK)UaKUb(a(Ba)*BA(bA(bA(bA(bA(bA(bA(bA)*(bKUK)UK)UK)U\
K)UK)UK)UK)UK)UAK)UK)
gap> ok := DCrules(dcrws3);;
gap> alph3e := dcrws3!.alphabet;;
gap> Print("H-rules:\n");  DisplayAsString( Hrules(dcrws3), alph3e, true );
H-rules:
[ HB, Ha ]
[ HaB, Hb ]
[ Hab, H ]
[ HbAB, HAba ]
[ HbAbAB, HAbAba ]
[ HbAbAbAB, HAbAbAba ]
[ HbAbAbAbAB, HAbAbAbAba ]
[ HbAbAbAbAbAB, HAbAbAbAbAba ]
[ HbAbAbAbAbAbAB, HAbAbAbAbAbAba ]
[ HbAbAbAbAbAbAbAB, HAbAbAbAbAbAbAba ]
gap> Print("K-rules:\n");  DisplayAsString( Krules(dcrws3), alph3e, true );;
K-rules:
[ BK, aK ]
[ BaK, bK ]
[ baK, K ]
[ BAbK, abAK ]
[ BAbAbK, abAbAK ]
[ BAbAbAbK, abAbAbAK ]
[ BAbAbAbAbK, abAbAbAbAK ]
[ BAbAbAbAbAbK, abAbAbAbAbAK ]
[ BAbAbAbAbAbAbK, abAbAbAbAbAbAK ]
[ BAbAbAbAbAbAbAbK, abAbAbAbAbAbAbAK ]
gap> Print("HK-rules:\n");  DisplayAsString( HKrules(dcrws3), alph3e, true );;
HK-rules:
[ HbK, HAK ]
[ HbAbK, HAbAK ]
[ HbAbAbK, HAbAbAK ]
[ HbAbAbAbK, HAbAbAbAK ]
[ HbAbAbAbAbK, HAbAbAbAbAK ]
[ HbAbAbAbAbAbK, HAbAbAbAbAbAK ]
[ HbAbAbAbAbAbAbK, HAbAbAbAbAbAbAK ]
```

2.4.3 NextWord

▷ `NextWord(dcrws, word)` (operation)
▷ `WordToString(word, alph)` (operation)
▷ `DisplayAsString(word, alph)` (operation)
▷ `IdentityDoubleCoset(dcrws)` (operation)

These functions may be used to find normal forms of increasing length for double coset representatives.

Example

```
gap> len := 30;;
gap> L3 := 0*[1..len];;
gap> L3[1] := IdentityDoubleCoset( dcrws3 );;
gap> for i in [2..len] do
gap>     L3[i] := NextWord( dcrws3, L3[i-1] );
gap> od;
gap> ## List of 30 normal forms for double cosets:
gap> DisplayAsString( L3, alph3e, true );
[ HK, HAK, HaK, HAbK, HbAK, HABAK, HAbAK, HABabK, HAbAbK, HbAbAK, HbaBAK, HABa\
BAK, HAbAbAK, HABaBabK, HAbABabK, HAbAbAbK, HbAbAbAK, HbaBAbAK, HbaBaBAK, HABa\
BaBAK, HAbAbAbAK, HABaBaBabK, HAbABaBabK, HAbAbABabK, HAbAbAbAbK, HbAbAbAbAK, \
HbaBAbAbAK, HbaBaBAbAK, HbaBaBaBAK, HABaBaBaBAK ]
gap> w := NextWord( dcrws3, L3[30] );
m1*(m3*m6)^4*m3*m2
gap> s := WordToString( w, alph3e );
"HAbAbAbAbAK"
```

Chapter 3

Development History

3.1 Versions of the package

The first version of the package, written for GAP 3, formed part of Anne Heyworth's thesis [Hey99] in 1999, but was not made generally available.

Version 0.91 was prepared to run under GAP 4.4.6, in July 2005.

Version 0.94 differed in two significant ways.

- The manual was produced using the GAPDoc package.
- The test file `kan/tst/kan_manual.tst` set the `AssertionLevel` to `0` to avoid recursion in the Automata package.

Version 0.95, of 9th October 2007, just fixed file protections and added a `CHANGES` file.

Version 0.96 was required because the Kan website moved with the rest of the Mathematics website at Bangor.

Version 0.97, of November 18th 2008, deleted temporary fixes which were no longer needed once version 1.12 of Automata became available.

Version 1.01, of August 2011, included minor changes required for GAP 4.5. In particular, the test file `kan_manual.tst` was replaced by the pair `fulltest.tst` and `parttest.tst`.

Version 1.11, of December 2014 was required when the Kan website moved yet again. At the same time a bitbucket repository for the package was started.

Kan became an accepted GAP package in May 2015.

3.2 What needs doing next?

There are too many items to list here, but some of the most important are as follows.

- Implement iterators and enumerators for double cosets.
- At present the methods for `DoubleCosetsNC` and `RightCosetsNC` in this package return automata, rather than lists of cosets or coset enumerators. This needs to be fixed.
- Provide methods for operations such as `DoubleCosetRepsAndSizes`.
- Convert the rest of the original GAP 3 version of Kan to GAP 4.

3.2.1 DoubleCosetsAutomaton

▷ DoubleCosetsAutomaton(*G, U, V*) (operation)
▷ RightCosetsAutomaton(*G, V*) (operation)

Alternative methods for DoubleCosetsNC(G,U,V) and RightCosetsNC(G,V) *should be* provided in the cases where the group G has a rewriting system or is known to be infinite. At present the functions RightCosetsAutomaton and DoubleCosetsAutomaton return minimized automata, and Iterators for these are not yet available.

Example
```
gap> F := FreeGroup(2);;
gap> rels := [ F.2^2, (F.1*F.2)^2 ];;
gap> G4 := F/rels;;
gap> genG4 := GeneratorsOfGroup( G4 );;
gap> a := genG4[1];  b := genG4[2];;
gap> U := Subgroup( G4, [a^2] );;
gap> V := Subgroup( G4, [b] );;
gap> dc4 := DoubleCosetsAutomaton( G4, U, V );;
gap> Print( dc4 );
Automaton("det",5,"HKaAbB",[ [ 2, 2, 2, 5, 2 ], [ 2, 2, 1, 2, 1 ], [ 2, 2, 2, \
2, 3 ], [ 2, 2, 2, 2, 2 ], [ 2, 2, 2, 2, 2 ], [ 2, 2, 2, 2, 2 ] ],[ 4 ],[ 1 ])\
;;
gap> rc4 := RightCosetsAutomaton( G4, V );;
gap> Print( rc4 );
Automaton("det",6,"HKaAbB",[ [ 2, 2, 2, 6, 2, 2 ], [ 2, 2, 1, 2, 1, 1 ], [ 2, \
2, 3, 2, 2, 3 ], [ 2, 2, 2, 2, 5, 5 ], [ 2, 2, 2, 2, 2, 2 ], [ 2, 2, 2, 2, 2, \
2 ] ],[ 4 ],[ 1 ]);;
```

References

[BGHW06] R. Brown, N. Ghani, A. Heyworth, and C. D. Wensley. String rewriting systems for double coset systems. *J. Symbolic Comput.*, 41:573–590, 2006. 4, 6, 12

[BH00] R. Brown and A. Heyworth. Using rewriting systems to compute left kan extensions and induced actions of categories. *J. Symbolic Comput.*, 29:5–31, 2000. 4

[Hey99] A. Heyworth. *Applications of Rewriting Systems and Groebner Bases to Computing Kan Extensions and Identities Among Relations*. PhD thesis, University of Wales, Bangor, 1999. `http://www.maths.bangor.ac.uk/research/ftp/theses/heyworth.ps.gz`. 4, 14

[Lin91] S. Linton. Double coset enumeration. *J. Symbolic Comput.*, 12:415–426, 1991. 4

Index