Share Package

—

Unipot

by

Sergei Haller
(Sergei.Haller@math.uni-giessen.de)

Arbeitsgruppe Algebra
Mathematisches Institut, Justus Liebig Universität Giessen
Arndtstr. 2, 35392 Gießen, Germany

Contents

Preface

`Unipot` is a share package for GAP4 [GAP4]. This package is the content of my diploma thesis [SH2000].

Let U be a unipotent subgroup of a Chevalley group of Type $L(K)$. Then it is generated by the elements $x_r(t)$ for all $r \in \Phi^+, t \in K$. The roots of the underlying root system Φ are ordered according to the height function. Each element of U is a product of the root elements $x_r(t)$. By the Theorem 5.3.3 from [Carter72] each element of U can be uniquely written as a product of root elements with roots in increasing order. This unique form is called the canonical form.

The main purpose of this package is to compute the canonical form of an element of the group U. For we have implemented the unipotent subgroups of Chevalley groups and their elements as GAP objects and installed some operations for them. One method for the operation `Comm` uses the Chevalley's commutator formula, which we have implemented, too.

1 ► `Root Systems`

We are using the root systems and the structure constants available in GAP via the simple Lie algebras. We also are using the ordering of roots available in GAP.

Note that the structure constants in GAP4.1 are not generated corresponding to a Chevalley basis, so computations in the groups of type B_l may produce an error and computations in groups of types B_l, C_l and F_4 may lead to wrong results. In the groups of other types we haven't seen any wrong results but can not guarantee that all results are correct.

In the revision 4.2 of GAP the structure constants are generated corresponding to a Chevalley basis, so that they meet all our assumtions.

Therefore the share package requires the revision 4.2 of GAP.

2 ► `Future of 'unipot'`

In one of the future versions of the share package `unipot` we plan to implement some other features. Here is a small list of them:

- GAP4.2 provides special root system objects. We should use them.
- Provide some root systems in common notations (like Carter or Bourbaki).
- Allow the user to provide his own table of structure constants.
- Provide whole Chevalley groups as GAP objects
- Provide root subgroups
- The elements of Chevalley groups should act on the underlying simple Lie algebra as automorphisms
- There are many known properties of the Chevalley groups and their unipotent subgroups like simplicity, central series, etc. Implement them.

3 ► `Citation`

If you use `unipot` to solve a problem or publish some result that was partly obtained using `unipot`, I would appreciate it if you would cite `unipot`, just as you would cite another paper that you used. (Below is a sample citation.) Again I would appreciate if you could inform me about such a paper.

Specifically, please refer to:

```
[Hal00] Sergei Haller. Unipot --- a system for computing with elements
        of unipotent subgroups of Chevalley groups, Version 1.1.
        Justus-Liebig Universitaet Giessen, Germany, July 2000.
        (ftp://ftp-pclabor.hrz.uni-giessen.de/SHadow/unipot/)
```

(Should the reference style require full addresses please use: “Arbeitsgruppe Algebra, Mathematisches Institut, Justus-Liebig Universität Gießen, Arndtstr. 2, 35392 Gießen, Germany”)

1 The Share Package Unipot

This chapter describes the share package `unipot`. This share package provides the ability to compute with elements of unipotent subgroups of Chevalley groups.

In this chapter we will refer to unipotent subgroups of Chevalley groups as "unipotent subgroups" and to elements of unipotent subgroups as "unipotent elements".

1.1 General functionality

In this section we will describe the general functionality provided by this package.

1 ▶ `UnipotChevInfo()` I

`UnipotChevInfo` is an `InfoClass` used in this package. `InfoLevel` of this `InfoClass` is set to 1 by default.

1.2 Unipotent subgroups of Chevalley groups

In this section we will describe the functionality for unipotent subgroups provided by this package.

1 ▶ `IsUnipotChevSubGr` C

Category for unipotent subgroups.

2 ▶ `UnipotChevSubGr(` *type*, *n*, *F* `)` F

`UnipotChevSubGr` returns the unipotent subgroup U of the Chevalley group of type *type*, rank n over the ring F.

type must be one of "A", "B", "C", "D", "E", "F", "G"

For the types A to D, n must be a positive integer.

For the type E, n must be one of 6, 7, 8.

For the type F, n must be 4.

For the type G, n must be 2.

```
gap> U_G2 := UnipotChevSubGr("G", 2, Rationals);
<Unipotent subgroup of a Chevalley group of type G2 over Rationals>

gap> U_E3 := UnipotChevSubGr("E", 3, Rationals);
Error <n> must be one of 6, 7, 8 for type E  at
Error( "<n> must be one of 6, 7, 8 for type E " );
UnipotChevFamily( type, n, F ) called from
<function>( <arguments> ) called from read-eval-loop
Entering break read-eval-print loop, you can 'quit;' to quit to outer loop,
or you can return to continue
brk>
```

3 ▶ `PrintObj(` *U* `)` M
▶ `ViewObj(` *U* `)` M

Special methods for unipotent subgroups. (see GAP Reference Manual, section "ref:view and print" for general information on `View` and `Print`)

```
gap> Print(U_G2);
UnipotChevSubGr( "G", 2, Rationals )gap> View(U_G2);
<Unipotent subgroup of a Chevalley group of type G2 over Rationals>
```

4 ▶ `One( U )` M
▶ `OneOp( U )` M

Special methods for unipotent subgroups. Return the identity of U.

5 ▶ `Size( U )` M

`Size` returns the size of a unipotent subgroup. This is a special method for unipotent subgroups.

Size can be computed using the result in Carter [Carter72], Theorem 5.3.3 (ii).

6 ▶ `RootSystem( U )` M

This method is similar to the method `RootSystem` for semisimple Lie algebras (see GAP4.1 Reference Manual, section 58.7 for further information). `RootSystem` calculates the root system of the unipotent subgroup U. The output is a record with the following components:

- `fundroots` A set of fundamental roots
- `posroots` The set of positive roots of the root system. The positive roots are listed *according to increasing height*.

```
gap> RootSystem(U_G2);
rec( posroots := [ [ 2, -1 ], [ -3, 2 ], [ -1, 1 ], [ 1, 0 ], [ 3, -1 ], [ 0, 1 ] ],
    fundroots := [ [ 2, -1 ], [ -3, 2 ] ] )
gap>
```

1.3 Elements of unipotent subgroups of Chevalley groups

In this section we will describe the functionality for unipotent elements provided by this package.

1 ▶ `IsUnipotChevElem` C

Category for elements of a unipotent subgroup.

2 ▶ `IsUnipotChevRepByRootNumbers` R
▶ `IsUnipotChevRepByFundamentalCoeffs` R
▶ `IsUnipotChevRepByRoots` R

`IsUnipotChevRepByRootNumbers`, `IsUnipotChevRepByFundamentalCoeffs` and `IsUnipotChevRepByRoots` are different representations for unipotent elements.

Roots of elements with representation `IsUnipotChevRepByRootNumbers` are represented by their numbers (positions) in `RootSystem(`U`).posroots`.

Roots of elements with representation `IsUnipotChevRepByFundamentalCoeffs` are represented by coefficients of linear combinations of fundamental roots `RootSystem(`U`).fundroots`.

Roots of elements with representation `IsUnipotChevRepByRoots` are represented by roots themself.

(See 1.3.3, 1.3.4 and 1.3.5 for examples)

3 ▶ `UnipotChevElemByRootNumbers( U, list )` O
▶ `UnipotChevElemByRootNumbers( U, r, x )` O
▶ `UnipotChevElemByRN( U, list )` O
▶ `UnipotChevElemByRN( U, r, x )` O

`UnipotChevElemByRootNumbers` returns an element of a unipotent subgroup U with representation `IsUnipotChevRepByRootNumbers` (see 1.3.2).

list should be a list of records with components r and x representing the number of the root in `RootSystem(`U`).posroots` and a ring element, respectively.

The second variant of `UnipotChevElemByRootNumbers` is an abbreviation for the first one if *list* contains only one record.

`UnipotChevElemByRN` is a synonym for `UnipotChevElemByRootNumbers`.

```
gap> IsIdenticalObj( UnipotChevElemByRN, UnipotChevElemByRootNumbers );
true
gap> y := UnipotChevElemByRootNumbers(U_G2, [rec(r:=1, x:=2), rec(r:=5, x:=7)]);
x_{1}( 2 ) * x_{5}( 7 )
gap> x := UnipotChevElemByRootNumbers(U_G2, 1, 2);
x_{1}( 2 )
```

In this example we create two elements: $x_{r_1}(2) \cdot x_{r_5}(7)$ and $x_{r_1}(2)$, where $r_i, i = 1, \ldots, 6$ are the positive roots in `RootSystem(`U`).posroots` and $x_{r_i}(t), i = 1, \ldots, 6$ the corresponding root elements.

4 ► `UnipotChevElemByFundamentalCoeffs(` U`,` *list* `)` O
► `UnipotChevElemByFundamentalCoeffs(` U`,` *coeffs*`,` x `)` O
► `UnipotChevElemByFC(` U`,` *list* `)` O
► `UnipotChevElemByFC(` U`,` *coeffs*`,` x `)` O

`UnipotChevElemByFundamentalCoeffs` returns an element of a unipotent subgroup U with representation `IsUnipotChevRepByFundamentalCoeffs` (see 1.3.2).

list should be a list of records with components *coeffs* and x representing a root in `RootSystem(`U`).posroots` as coefficients of a linear combination of fundamental roots `RootSystem(`U`).fundroots` and a ring element, respectively.

The second variant of `UnipotChevElemByFundamentalCoeffs` is an abbreviation for the first one if *list* contains only one record.

`UnipotChevElemByFC` is a synonym for `UnipotChevElemByFundamentalCoeffs`.

```
gap> y1 := UnipotChevElemByFundamentalCoeffs( U_G2,
>      [ rec( coeffs := [ 1, 0 ], x := 2 ),
>        rec( coeffs := [ 3, 1 ], x := 7 ) ] );
x_{[ 1, 0 ]}( 2 ) * x_{[ 3, 1 ]}( 7 )
gap> x1 := UnipotChevElemByFundamentalCoeffs( U_G2, [ 1, 0 ], 2 );
x_{[ 1, 0 ]}( 2 )
```

In this example we create the same two elements as in 1.3.3: $x_{[1,0]}(2) \cdot x_{[3,1]}(7)$ and $x_{[1,0]}(2)$, where $[1,0] = 1r_1+0r_2 = r_1$ and $[3,1] = 3r_1+1r_2 = r_5$ are the first and the fifth positive roots of `RootSystem(`U`).posroots` respectively.

5 ► `UnipotChevElemByRoots(` U`,` *list* `)` O
► `UnipotChevElemByRoots(` U`,` r`,` x `)` O
► `UnipotChevElemByR(` U`,` *list* `)` O
► `UnipotChevElemByR(` U`,` r`,` x `)` O

`UnipotChevElemByRoots` returns an element of a unipotent subgroup U with representation `IsUnipotChevRepByRoots` (see 1.3.2).

list should be a list of records with components r and x representing the root in `RootSystem(`U`).posroots` and a ring element, respectively.

The second variant of `UnipotChevElemByRoots` is an abbreviation for the first one if *list* contains only one record.

UnipotChevElemByR is a synonym for UnipotChevElemByRoots.

```
gap> y2 := UnipotChevElemByRoots( U_G2,
>      [ rec( r := [ 2, -1 ], x := 2 ),
>        rec( r := [ 3, -1 ], x := 7 ) ] );
x_{[ 2, -1 ]}( 2 ) * x_{[ 3, -1 ]}( 7 )
gap> x2 := UnipotChevElemByRoots( U_G2, [ 2, -1 ], 2 );
x_{[ 2, -1 ]}( 2 )
```

In this example we create again the two elements as in previous examples: $x_{[2,-1]}(2) \cdot x_{[3,-1]}(7)$ and $x_{[2,-1]}(2)$, where $[2,-1] = r_1$ and $[3,-1] = r_5$ are the first and the fifth positive roots of RootSystem(U).posroots respectively.

6 ► UnipotChevElemByRootNumbers(x) O
► UnipotChevElemByFundamentalCoeffs(x) O
► UnipotChevElemByRoots(x) O

UnipotChevElemByRootNumbers is provided for converting elements to the representation IsUnipotChevRepByRootNumbers. If x has already the representation IsUnipotChevRepByRootNumbers, then x itself is returned. Otherwise a **new** element with representation IsUnipotChevRepByRootNumbers is generated.

UnipotChevElemByFundamentalCoeffs and UnipotChevElemByRoots do the same for the representations IsUnipotChevRepByFundamentalCoeffs and IsUnipotChevRepByRoots, respectively.

```
gap> x;
x_{1}( 2 )
gap> x1 := UnipotChevElemByFundamentalCoeffs( x );
x_{[ 1, 0 ]}( 2 )
gap> IsIdenticalObj(x, x1); x = x1;
false
true
gap> x2 := UnipotChevElemByFundamentalCoeffs( x1 );;
gap> IsIdenticalObj(x1, x2);
true
```

Note: If some attributes of x are known (e.g Inverse (see 1.3.13), CanonicalForm (see 1.3.7)), then they are "converted" to the new representation, too.

7 ► CanonicalForm(x) A

CanonicalForm returns the canonical form of x. For more information on the canonical form see Carter [Carter72], Theorem 5.3.3 (ii). It says:

Each element of a unipotent subgroup U of a Chevalley group with root system Φ is uniquely expressible in the form

$$\prod_{r_i \in \Phi^+} x_{r_i}(t_i),$$

where the product is taken over all positive roots in increasing order.

```
gap> z := UnipotChevElemByFC( U_G2,
>      [ rec( coeffs := [0,1], x := 3 ),
>        rec( coeffs := [1,0], x := 2 ) ] );
x_{[ 0, 1 ]}( 3 ) * x_{[ 1, 0 ]}( 2 )
gap> CanonicalForm(z);
x_{[ 1, 0 ]}( 2 ) * x_{[ 0, 1 ]}( 3 ) * x_{[ 1, 1 ]}( 6 ) *
x_{[ 2, 1 ]}( 12 ) * x_{[ 3, 1 ]}( 24 ) * x_{[ 3, 2 ]}( -72 )
```

8 ▶ `PrintObj( x )` M
▶ `ViewObj( x )` M

Special methods for unipotent elements. (see GAP Reference Manual, section "ref:view and print" for general information on `View` and `Print`)

```
gap> Print(x);
UnipotChevElemByRootNumbers( UnipotChevSubGr( "G", 2, Rationals ), [ rec(
      r := 1,
      x := 2 ) ] )gap> View(x);
x_{1}( 2 )

gap> Print(x1);
UnipotChevElemByFundamentalCoeffs( UnipotChevSubGr( "G", 2, Rationals ),
[ rec(
      coeffs := [ 1, 0 ],
      x := 2 ) ] )gap> View(x1);
x_{[ 1, 0 ]}( 2 )
```

9 ▶ `ShallowCopy( x )` M

This is a special method for unipotent elements.

`ShallowCopy` creates a copy of x. The returned object is **not identical** to x but it is **equal** to x w.r.t. the equality operator `=`. **Note** that `CanonicalForm` and `Inverse` of x (if known) are identical to `CanonicalForm` and `Inverse` of the returned object.

(See GAP Reference Manual, section "ref:duplication of objects" for further information on copyability)

10 ▶ x `=` y M

Special method for unipotent elements. If x and y are identical or are products of the **same** root elements then `true` is returned. Otherwise `CanonicalForm` (see 1.3.7) of both arguments must be computed (if not already known), which may be expensive.

```
gap> y := UnipotChevElemByRootNumbers( U_G2, [ rec(
>        r := 1,
>        x := 2 ), rec(
>        r := 5,
>        x := 7 ) ] );
x_{1}( 2 ) * x_{5}( 7 )
gap>
gap> z := UnipotChevElemByRootNumbers( U_G2, [ rec(
>        r := 5,
>        x := 7 ), rec(
>        r := 1,
>        x := 2 ) ] );
x_{5}( 7 ) * x_{1}( 2 )
gap> y=z;
#I  CanonicalForm for the 1st argument is not known.
#I                   computing it may take a while.
#I  CanonicalForm for the 2nd argument is not known.
#I                   computing it may take a while.
true
gap>
```

11 ▶ `x * y` M

Special method for unipotent elements. The expressions in the form $x_r(t)x_r(u)$ will be reduced to $x_r(t+u)$ whenever possible.

```
gap> y;z;
x_{1}( 2 ) * x_{5}( 7 )
x_{5}( 7 ) * x_{1}( 2 )
gap> y*z;
x_{1}( 2 ) * x_{5}( 14 ) * x_{1}( 2 )
```

Note: If both arguments have the same representation, the product will have it too. But if the representations are different, the representation of the first argument will become the representation of the product.

```
gap> x; x1; x=x1;
x_{1}( 2 )
x_{[ 1, 0 ]}( 2 )
true
gap> x * x1;
x_{1}( 4 )
gap> x1 * x;
x_{[ 1, 0 ]}( 4 )
```

12 ▶ `OneOp( x )` M

Special method for unipotent elements. `OneOp` returns the multiplicative neutral element of x. This is equal to $x\hat{}0$.

13 ▶ `Inverse( x )` M
▶ `InverseOp( x )` M

Special methods for unipotent elements. We are using the fact

$$\Big(x_{r_1}(t_1)\cdots x_{r_m}(t_m)\Big)^{-1} = x_{r_m}(-t_m)\cdots x_{r_1}(-t_1).$$

14 ▶ `Comm( x, y )` M
▶ `Comm( x, y, "canonical" )` M

Special methods for unipotent elements.

`Comm` returns the commutator of x and y, i.e. $x^{-1}\cdot y^{-1}\cdot x\cdot y$. The second variant returns the canonical form of the commutator. In some cases it may be more efficient than `CanonicalForm( Comm( x, y ) )`

15 ▶ `IsRootElement( x )` P

`IsRootElement` returns `true` if and only if x is a *root element*, i.e $x = x_r(t)$ for some root r. We store this property just after creating objects.

Note: the canonical form of x may be a root element even if x isn't one.

A Installing and Loading unipot

This appendix describes the procedure of installing the share package

Installing `unipot` should be easy once you have installed GAP itself. We assume here that you want to install `unipot` in its standard location, which is in the "pkg" subdirectory of the main GAP4 installation.

A.1 Overview

You have to perform the following steps to install `unipot`:

- Get the sources.
- Unpack the sources with the *unzoo* utility.
- Optionally edit the ALLPKG file so that the `unipot` documentation will be available when GAP starts up.

A.2 What you need to install unipot

Being a share package for GAP4 and implemented in the GAP4 language, `unipot` of course needs at least GAP version 4.2. (See Preface.1 why you shouldn't use it with GAP4.1.) `Unipot` runs on any system supporting GAP4. It is tested with GAP4.2, but should work with any non-beta version of GAP4 (with exceptions stated in Preface.1 for GAP4.1).

A.3 Getting and unpacking the sources

You can download the sources from the same places as GAP. So the main FTP servers are:

```
ftp://ftp-gap.dcs.st-and.ac.uk/pub/gap/gap4/
ftp://ftp.math.rwth-aachen.de/pub/gap4/
ftp://ftp.ccs.neu.edu/pub/mirrors/ftp-gap.dcs.st-and.ac.uk/pub/gap/gap4/
ftp://pell.anu.edu.au/pub/algebra/gap4/
```

You need only one file with the name "unipot1r1.zoo" which is in the subdirectory for the share packages. When you installed GAP you used the utility *unzoo* to unpack the distribution. You will need this here again. See the GAP-manual, chapter "ref:installing gap" for instructions on how to get and compile this. You now change your current directory to the "pkg" subdirectory of the location where you installed GAP (you typed an *unzoo*-command, then a new directory called "gap4" or something like that was created, this directory contains the "pkg" subdirectory). The standard location would be: (do not type the prompt character `#`)

```
# cd /usr/local/lib/gap4/pkg
```

Now you extract the sources for the `unipot` share package:

```
# unzoo -x unipot1r1.zoo
unipot/README     -- extracted as text
...
/bin/mkdir: cannot make directory ‘unipot’: File exists
...
```

Note that the warning is **not** serious.

The *unzoo* utility unpacks the files and stores them into the apropriate subdirectories. `unipot` resides completely in the following subdirectory (assuming standard location):

```
/usr/local/lib/gap4/pkg/unipot
```

A.4 Installing in a different than the standard location

It could happen that you do not want to install `unipot` in its standard location, perhaps because you do not want to bother your system administrator and have no access to the GAP directory. In this case you can unpack `unipot` in any other location within a "pkg" directory with the *unzoo* command as described above. Let us call this directory "pkg" for the moment. You get an "unipot" subdirectory with all the files of `unipot` in it. Then you follow the standard procedure with following exceptions:

Say, the directory containing the "pkg" directory is "/home/user/mygap". Note that you have either to edit the startup script "gap.sh": Add "/home/user/mygap" separating it with semicolon `;` from previous directories for the variable "GAP_DIR". Or you have to start GAP with following command line option:

```
# gap4 -l "/usr/local/lib/gap4;/home/user/mygap"
```

A.5 Loading unipot in GAP

Add a line to the "ALLPKG" file in the "pkg" directory

```
# cd /name-of-gap-directory/pkg
# echo unipot >> ALLPKG
```

This makes the documentation of the package available in any GAP4 session, even if the package is not loaded. Like any other share package, `unipot` is loaded in GAP with

```
gap> RequirePackage("unipot");
```

within the GAP4 session.

If you have problems with this package, wish to make comments or suggestions, or if you find bugs, please send e-mail to me.

Sergei Haller, mailto:`Sergei.Haller@math.uni-giessen.de`

Also, I would like to hear about applications of this package. (See Preface.3)

Bibliography