LOOPS

Version 2.2.0

Computing with quasigroups and loops in GAP

Gábor P. Nagy

Department of Mathematics

University of Szeged

email: nagyg@math.u-szeged.hu

Petr Vojtěchovský

Department of Mathematics

University of Denver

email: petr@math.du.edu

July 9, 2012

Contents

1 Introduction

LOOPS is a package for GAP4 whose purpose is to:

- provide researchers in nonassociative algebra with a powerful computational tool concerning finite loops and quasigroups,
- extend GAP toward the realm of nonassociative structures.

1.1 Installation

We assume that you have GAP 4.4 or newer installed on your computer. Download the LOOPS package from the distribution website

```
http://www.math.du.edu/loops
```

and unpack the downloaded file into the `pkg` subfolder of your GAP folder.

After this step, there should be a subfolder `loops` in your `pkg` folder. The package LOOPS can then be loaded to GAP anytime by calling

```
LoadPackage("loops");
```

If you wish to load LOOPS automatically while starting GAP, open the file `loops/PackageInfo.g`, and change `Autoload:=false` to `Autoload:=true` in the file.

1.2 Documentation

The documentation is available is several formats: TEX, pdf, dvi, ps, html, and as an online help in GAP. All these formats have been obtained directly from the master TEX documentation file. Consequently, the different formats of the documentation differ only in their appearance, not in content.

All formats of the documentation except html can be found in the `doc` folder of LOOPS, while the html version is in the `htm` folder. You can also download the documentation at the LOOPS distribution website.

The online GAP help is available upon installing LOOPS, and can be accessed in the usual way, i.e., upon typing `?`*command*, GAP displays the section of the LOOPS manual containing information about *command*.

1.3 Test files

Test files conforming to the GAP standards are provided for LOOPS. They can be found in the folder `tst`. The command `ReadPackage("loops", "tst/testall.g")` runs all tests for LOOPS.

1.4 Feedback

We welcome all comments and suggestions on LOOPS, especially those concerning the future development of the package. You can contact us by e-mail.

1.5 Acknowledgement

We thank the following people for sending us remarks and comments, and for suggesting new functionality of the package: Muniru Asiru, Bjoern Assmann, Andreas Distler, Steve Flammia, Kenneth W. Johnson, Michael K. Kinyon, Frank Lübeck, and Jonathan D. H. Smith.

Gábor P. Nagy was supported by OTKA grants F042959 and T043758, and Petr Vojtěchovský by the 2006 PROF grant of the University of Denver.

2 Mathematical background

We assume that you are familiar with the theory of quasigroups and loops, for instance with the textbook of Bruck [2] or Pflugfelder [18]. Nevertheless, we did include definitions and results in this manual in order to unify the terminology and improve the intelligibility of the text. Some general concepts of quasigroups and loops can be found in this chapter. More special concepts are defined throughout the text as needed.

2.1 Quasigroups and loops

A set with one binary operation (denoted $\cdot$ here) is called *groupoid* or *magma*, the latter name being used in GAP. Associative groupoid is a *semigroup*.

An element 1 of a groupoid G is a *neutral element* or an *identity element* if $1 \cdot x = x \cdot 1 = x$ for every x in G. Semigroup with a neutral element is a *monoid*.

Let G be a groupoid with neutral element 1. Then an element y is called a *two-sided inverse* of x in G if $x \cdot y = y \cdot x = 1$. A monoid in which every element has a two-sided inverse is called a *group*.

Groups can be reached in another way from groupoids, namely through quasigroups and loops.

A *quasigroup* Q is a groupoid such that the equation $x \cdot y = z$ has a unique solution in Q whenever two of the three elements x, y, z of Q are specified. Note that multiplication tables of finite quasigroups are precisely *Latin squares*, i.e., a square arrays with symbols arranged so that each symbol occurs in each row and in each column exactly once. A *loop* L is a quasigroup with a neutral element.

Groups are clearly loops, and one can show easily that an associative quasigroup is a group.

2.2 Translations

Given an element x of a quasigroup Q we can associative two permutations of Q with it: the *left translation* $L_x : Q \to Q$ defined by $y \mapsto x \cdot y$, and the *right translation* $R_x : Q \to Q$ defined by $y \mapsto y \cdot x$.

Although it is possible to compose two right (left) translations, the resulting permutation is not necessarily a right (left) translation. The set $\{L_x; x \in Q\}$ is called the *left section* of Q, and $\{R_x; x \in Q\}$ is the *right section* of Q.

Let S_Q be the symmetric group on Q. Then the subgroup $\mathrm{LMlt}(Q) = \langle L_x | x \in Q\rangle$ of S_Q generated by all left translations is the *left multiplication group* of Q. Similarly, $\mathrm{RMlt}(Q) = \langle R_x | x \in Q\rangle$ is the *right multiplication group* of Q. The smallest group containing both $\mathrm{LMlt}(Q)$ and $\mathrm{RMlt}(Q)$ is called the *multiplication group* of Q and is denoted by $\mathrm{Mlt}(Q)$.

2.3 Homomorphisms and homotopisms

Let K, H be two quasigroups. Then a map $f : K \to H$ is a *homomorphism* if $f(x) \cdot f(y) = f(x \cdot y)$ for every x, $y \in K$. If f is also a bijection, we speak of an *isomorphism*, and the two quasigroups are called *isomorphic*.

The ordered triple (α, β, γ) of maps α, β, $\gamma : K \to H$ is a *homotopism* if $\alpha(x) \cdot \beta(y) = \gamma(x \cdot y)$ for every x, $y \in K$. If the three maps are bijections, (α, β, γ) is an *isotopism*, and the two quasigroups are *isotopic*.

Isotopic groups are necessarily isomorphic, but this is certainly not true for nonassociative quasigroups or loops. In fact, every quasigroup is isotopic to a loop, as we shall see.

Let $(K, \cdot)$, $(K, \circ)$ be two quasigroups defined on the same set K. Then an isotopism $(\alpha, \beta, \mathrm{id}_K)$ is called a *principal isotopism*. An important class of principal isotopisms is obtained as follows:

Let $(K, \cdot)$ be a quasigroup, and let f, g be elements of K. Define a new operation $\circ$ on K by

$$x \circ y = R_g^{-1}(x) \cdot L_f^{-1}(y),$$

where R_g, L_f are translations. Then $(K, \circ)$ is a quasigroup isotopic to $(K, \cdot)$, in fact a loop with neutral element $f \cdot g$. We call $(K, \circ)$ a *principal loop isotope* of $(K, \cdot)$.

2.4 Extensions

Let K, F be loops. Then a loop Q is an *extension* of K by F if K is a normal subloop of Q such that Q/K is isomorphic to F. An extension Q of K by F is *nuclear* if K is an abelian group and $K \leq N(Q)$.

A map $\theta : F \times F \to K$ is a *cocycle* if $\theta(1, x) = \theta(x, 1) = 1$ for every $x \in F$.

The following theorem holds for loops Q, F and an abelian group K: Q is a nuclear extension of K by F if and only if there is a cocycle $\theta : F \times F \to K$ and a homomorphism $\varphi : F \to \mathrm{Aut}Q$ such that $K \times F$ with multiplication $(a, x)(b, y) = (a\varphi_x(b)\theta(x, y), xy)$ is isomorphic to Q.

3 How the package works

The package consists of three complementary components:

- the core algorithms for quasigroup theoretical notions (see Chapters 4, 5, 6 and 7),
- some specific algorithms, mostly for Moufang loops (see Chapter 8),
- the library of small loops (see Chapter 9).

Although we do not explain the algorithms in detail here, we describe the overarching ideas so that the user should be able to anticipate the capabilities and behavior of LOOPS during computation.

3.1 Representing quasigroups

Since the permutation representation in the usual sense is impossible for nonassociative structures, and since the theory of nonassociative presentations is not well understood, we had to resort to multiplication tables to represent quasigroups in GAP.

In order to save storage space, we sometimes use one multiplication table to represent several quasigroups (for instance when a quasigroup is a subquasigroup of another quasigroup).

Consequently, *the package is intended primarily for quasigroups and loops of small order*, say up to 1000.

The GAP categories `IsQuasigroupElement`, `IsLoopElement`, `IsQuasigroup`, and `IsLoop` are declared in LOOPS as follows:

```
DeclareCategory( "IsQuasigroupElement", IsMultiplicativeElement );
DeclareRepresentation( "IsQuasigroupElmRep",
    IsPositionalObjectRep and IsMultiplicativeElement, [1] );
DeclareCategory( "IsLoopElement",
    IsQuasigroupElement and IsMultiplicativeElementWithInverse );
DeclareRepresentation( "IsLoopElmRep",
    IsPositionalObjectRep and IsMultiplicativeElementWithInverse, [1] );
## latin (auxiliary category for GAP to tell apart IsMagma and IsQuasigroup)
DeclareCategory( "IsLatin", IsObject );
DeclareCategory( "IsQuasigroup", IsMagma and IsLatin );
DeclareCategory( "IsLoop", IsQuasigroup and
    IsMultiplicativeElementWithInverseCollection);
```

3.2 Conversions between magmas, quasigroups, loops and groups

Whether an object is considered a magma, quasigroup, loop or group is a matter of declaration in LOOPS. Loops are automatically quasigroups, and both groups and quasigroups are automatically magmas. All standard GAP commands for magmas are therefore available for quasigroups and loops, too.

In GAP, functions of the type `AsSomething(`X`)` convert the domain X into `Something`, if possible, without changing the underlying domain X. For example, if X is declared as magma but is associative and has neutral element and inverses, `AsGroup(` X `)` returns the corresponding group with the underlying domain X.

We have opted for a more general kind of conversions in LOOPS (starting with version 2.1.0), using functions of the type `IntoSomething(`X`)`. The two main features that distinguish `IntoSomething` from `AsSomething` are:

- The function `IntoSomething(` X `)` does not necessarily return the same domain as X. The reason is that X can be a group, for instance, defined on one of many possible domains, while `IntoLoop(` X `)` must result in a loop, and hence be defined on a subset of some interval `[1,..,n]` (see Section 4.1).
- In some special situations, the function `IntoSomething(` X `)` allows to convert X into `Something` even though X does not have all the properties of `Something`. For instance, every quasigroup is isotopic to a loop, so it makes sense to allow conversions of the type `IntoLoop(` Q `)` even if the quasigroup Q does not posses a neutral element.

Details of all conversions in LOOPS can be found in Section 4.9.

3.3 Calculating with quasigroups

Although the quasigroups are ultimately represented by multiplication tables, the algorithms are efficient because nearly all calculations are delegated to groups. The connection between quasigroups and groups is facilitated via the above-mentioned translations, and we illustrate it with a few examples:

Example 1: This example shows how properties of quasigroups can be translated into properties of translations in a straightforward way.

Let Q be a quasigroup. We ask if Q is associative. We can either test if $(xy)z = x(yz)$ for every x, y, $z \in Q$, or we can ask if $L_{xy} = L_x L_y$ for every x, $y \in Q$. Note that since L_{xy}, L_x, L_y are elements of a permutation group, we do not have to refer directly to the multiplication table once the left translations of Q are known.

Example 2: This example shows how properties of loops can be translated into properties of translations in a way that requires some theory.

A left Bol loop is a loop satisfying $x(y(xz)) = (x(yx))z$. We claim (without proof) that a loop L is left Bol if and only if $L_x L_y L_x$ is a left translation for every x, $y \in L$.

Example 3: This example shows that many properties of loops become purely group-theoretical once they are expresses in terms of translations.

A loop is simple if it has no nontrivial congruences. Then it is easy to see that a loop is simple if and only if its multiplication group is a primitive permutation group.

The main idea of the package is therefore to:

- calculate the translations and the associated permutation groups when they are needed,
- store them as attributes,
- use them in algorithms as often as possible.

3.4 Naming, viewing and printing quasigroups and their elements

GAP displays information about objects in two modes:

- `View` (default, short),
- `Print` (longer).

Moreover, when the name of an object is set, it is always shown, no matter which display mode is used.

Only loops contained in the libraries of LOOPS are named. For instance, the loop obtained via `MoufangLoop( 32, 4 )`, the 4th Moufang loop of order 32, is named `Moufang loop 32/4`.

When Q is a quasigroup of order n, it is displayed as `<quasigroup of order n>`. Similarly, a loop of order n appears as `<loop of order n>`.

The displayed information for a loop L is enhanced when it is known that L has certain additional properties. At this point, we support:

```
<associative loop ...>
<extra loop ...>
<Moufang loop ...>
<C loop ...>
<left Bol loop ...>
<right Bol loop ...>
<LC loop ...>
<RC loop ...>
<alternative loop ...>
<left alternative loop ...>
<right alternative loop ...>
<flexible loop ...>
```

The corresponding mathematical definitions and an example can be found in Section 7.4.

It is possible for a loop to have several of the above properties. In such a case, we display the first property on the list that is satisfied.

By default, elements of a quasigroup appear as `q`*n* and elements of a loop appear as `l`*n* in both display modes. The neutral element of a loop is always denoted by `l1`. However, one can change the names of elements of a quasigroup Q or loop L to *name* with

1 ▶ `SetQuasigroupElmName(` Q`,` *name* `)` O
▶ `SetLoopElmName(` L`,` *name* `)` O

For quasigroups and loops in the `Print` mode, we display the multiplication table (if it is known), or we display the elements.

In the following example, L is a loop with two elements.

```
gap> L;
<loop of order 2>
gap> Print( L );
<loop with multiplication table [ [ 1,  2 ], [  2,  1 ] ]>
gap> Elements( L );
[ l1, l2 ]
gap> SetLoopElmName( L, "loop_element" );; Elements( L );
[ loop_element1, loop_element2 ]
```

4 Creating quasigroups and loops

In this chapter we describe several ways in which quasigroups and loops can be created in LOOPS.

4.1 About Cayley tables

Let $X = \{x_1, \ldots, x_n\}$ be a set and $\cdot$ a binary operation on X. Then an n by n array with rows and columns bordered by x_1, …, x_n, in this order, is a *Cayley table*, or a *multiplication table* of $\cdot$, if the entry in the row x_i and column x_j is $x_i \cdot x_j$.

A Cayley table is a *quasigroup table* if it is a *Latin square*, i.e., if every entry x_i appears in every column and every row exactly once.

An annoying feature of quasigroup tables in practice is that they are often not bordered, and it is up to the reader to figure out what is meant. Throughout this manual and in LOOPS, we therefore make the following assumption: *All distinct entries in a quasigroup table must be integers, say $x_1 < x_2 < \cdots < x_n$, and if no border is specified, we assume that the table is bordered by x_1, …, x_n, in this order.* Note that we do not assume that the distinct entries x_1, …, x_n form the interval 1, …, n. The significance of this observation will become clear in Chapter 6.

Finally, we say that a quasigroup table is a *loop table* if the first row and the first column are the same, and if the entries in the first row are ordered in an ascending fashion.

4.2 Testing Cayley tables

A square array with integral entries is called a *matrix* in GAP. The following synonymous operations test if a matrix T is a quasigroup table, as defined above:

1 ► `IsQuasigroupTable(` T `)` O
► `IsQuasigroupCayleyTable(` T `)` O

The following synonymous operations test if a matrix T is a loop table:

2 ► `IsLoopTable(` T `)` O
► `IsLoopCayleyTable(` T `)` O

We would like to call attention to the fact that the package GUAVA also has some operations dealing with Latin squares. In particular, `IsLatinSquare` is declared in GUAVA.

4.3 Canonical and normalized Cayley tables

Although we do not assume that a quasigroup table with distinct entries $x_1 < \cdots < x_n$ satisfies $x_i = i$, it is often desirable to present quasigroup tables in the latter way. The rather general operation

1 ► `CanonicalCayleyTable(` T `)` O

takes any Cayley table T with distinct entries $x_1 < \cdots < x_n$, and returns a Cayley table in which x_i has been replaced by i.

The operation

2 ► `NormalizedQuasigroupTable(` T `)` O

makes a quasigroup table T into a loop table by:

- first calling `CanonicalCayleyTable` to rename the entries to 1, …, n,
- then permuting the columns of T so that the first row reads 1, …, n,
- and then permuting the rows of T so that the first column reads 1, …, n.

4.4 Creating quasigroups and loops manually

When T is a quasigroup table, the corresponding quasigroup is obtained by

1 ► `QuasigroupByCayleyTable(` T `)` O

Since `CanonicalCayleyTable` is called within the above operation, the resulting quasigroup will have a Cayley table with distinct entries 1, …, n.

Here is the analogous operation for a loop table T:

2 ► `LoopByCayleyTable(` T `)` O

And here is an example for methods concerning Cayley tables:

```
gap> ct := CanonicalCayleyTable( [[5,3],[3,5]] );
[ [ 2, 1 ], [ 1, 2 ] ]
gap> NormalizedQuasigroupTable( ct );
[ [ 1, 2 ], [ 2, 1 ] ]
gap> LoopByCayleyTable( last );
<loop of order 2>
gap> [ IsQuasigroupTable( ct ), IsLoopTable( ct ) ];
[ true, false ]
```

4.5 Creating quasigroups and loops from a file

Typing a large multiplication table manually is tedious and error-prone. We have therefore included a universal algorithm in LOOPS that reads multiplication tables of quasigroups from a file.

Instead of writing a separate algorithm for each common format, our algorithm relies on the user to provide a bit of information about the input file. Here is an outline of the algorithm, with file named F and a string D as arguments on the input:

- read the entire content of F into a string S,
- replace all end-of-line characters in S by spaces,
- replace by spaces all characters of S that appear in D,
- split S into maximal substrings without spaces, called *chunks*,
- recognize distinct chunks (let n be the number of distinct chunks),
- if the number of chunks is not n^2, report error,
- construct the multiplication table by assigning numerical values 1, …, n to chunks, depending on their position among distinct chunks.

The following examples clarify the algorithm and document its versatility. All examples are of the form $F + D \Longrightarrow T$, meaning that an input file containing F together with the string D produce multiplication table T.

Example 1: Data does not have to be arranged into an array of any kind.

$$\begin{array}{cccc} 0 & 1 & 2 & 1 \\ 2 & 0 & 2 & \\ 0 & 1 & & \end{array} \quad + \quad '''' \quad \Longrightarrow \quad \begin{array}{ccc} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{array}$$

Example 2: Chunks can be any strings.

$$\begin{array}{cc} \text{red} & \text{green} \\ \text{green} & \text{red} \end{array} \quad + \quad '''' \quad \Longrightarrow \quad \begin{array}{cc} 1 & 2 \\ 2 & 1 \end{array}$$

Example 3: A typical table produced by GAP is easily parsed by deleting brackets and commas.

$$[\ [0,1],\ [1,0]\ ] \quad + \quad ''[,]'' \quad \Longrightarrow \begin{array}{cc} 1 & 2 \\ 2 & 1 \end{array}$$

Example 4: A typical TEX table with rows separated by lines is also easily converted. Note that we have to use \\ to make sure that every occurrence of \ is deleted, since \\ represents the character \ in GAP.

$$\begin{array}{l} \text{x\& y\& z\textbackslash cr} \\ \text{y\& z\& x\textbackslash cr} + \quad \text{"\textbackslash\textbackslash cr\&"} \\ \text{z\& x\& y} \end{array} \Longrightarrow \begin{array}{ccc} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{array}$$

And here are the needed *LOOPS* commands:

1 ► `QuasigroupFromFile(` *F*`,` *D* `)` O
► `LoopFromFile(` *F*`,` *D* `)` O

4.6 Creating quasigroups and loops by sections

Let P be a set of n permutations of an n-element set X. If at most one permutation of P is the identity permutation, and if all other permutations of P move all points of X, the operation

1 ► `CayleyTableByPerms(` *P* `)` O

returns an $n \times n$ Cayley table C such that `C[i][j] = X[j]^P[i]`.

In particular, if P is the left section of a quasigroup Q, the above operation returns the multiplication table of Q.

If P is a set of permutations corresponding to the left translations of a quasigroup (or loop) Q, the operation

2 ► `QuasigroupByLeftSection(` *P* `)` O
► `LoopByLeftSection(` *P* `)` O

returns the corresponding quasigroup (or loop). The order of permutations in P is important in the quasigroup case, but it is disregarded in the loop case, since the order of rows in the corresponding multiplication table is determined by the presence of the neutral element.

Analogously, we define

3 ► `QuasigroupByRightSection(` *P* `)` O
► `LoopByRightSection(` *P* `)` O

Here is an example:

```
gap> S := Subloop( MoufangLoop( 12, 1 ), [ 3 ] );;
gap> ls := LeftSection( S );
[ (), (1,3,5), (1,5,3) ]
gap> CayleyTableByPerms( ls );
[ [ 1, 3, 5 ], [ 3, 5, 1 ], [ 5, 1, 3 ] ]
gap> CayleyTable( LoopByLeftSection( ls ) );
[ [ 1, 2, 3 ], [ 2, 3, 1 ], [ 3, 1, 2 ] ]
```

Let G be a group, H a subgroup, and T a right transversal to H in G. Then the operation $\circ$ defined on the right cosets $Q = \{Ht;\ t \in T\}$ by $Ht \circ Ht' = H(tt')$ turns Q into a quasigroup if and only if T is a right transversal to all conjugates $g^{-1}Hg$ of H in G. In fact, every quasigroup Q can be obtained in this way (let $G = \mathrm{Mlt}(Q)$, $H = \mathrm{RMlt}(Q)$ and $T = \{R_x;\ x \in Q\}$).

The resulting quasigroup (or loop) is returned via

4 ► `QuasigroupByRightSection(` *G*`,` *H*`,` *T* `)` O
► `LoopByRightSection(` *G*`,` *H*`,` *T* `)` O

We do not support the dual operations for left sections since, by default, actions in GAP act on the right.

To demonstrate `LoopByRightSection`, let us recall a construction due to Nagy [15]:

Let X be a simple group with subgroups Y_0, Y_1 such that $Y_0 \cap Y_1 = 1$ and $X = Y_0Y_1$. Let $G = X \times X$ and $H = Y_0 \times Y_1$. Then $T = \{(x, x^{-1});\ x \in X\}$ is a right transversal of H in G, and the corresponding loop is a simple (right) Bol loop.

The next example illustrates this construction with $X = A_5$, $Y_0 = A_4$, and $Y_1 = \langle(1, 2, 3, 4, 5)\rangle$.

```
gap> Shift := function( p ) # shifts permutation "up" by 5
>       local ls;
>       ls := ListPerm( p );
>       ls := Concatenation( [1,2,3,4,5], List( ls, x -> x + 5 ) );
>       return PermList( ls );
> end;
function( p ) ... end
gap> A := AlternatingGroup( 5 );;
gap> G := DirectProduct( A, A );;
gap> H := Subgroup( G, [ (1,2,3), (2,3,4), (6,7,8,9,10) ] );;
gap> T := List( A, x -> x * Shift(x)^(-1) );;
gap> L := LoopByRightSection( G, H, T );
<loop of order 60>
gap> [ IsRightBolLoop( L ), IsSimple( L ), IsMoufangLoop( L ) ];
[ true, true, false ]
```

4.7 Creating quasigroups and loops by extensions

If Q is a loop and K an abelian subgroup of $N(Q)$ then

1 ▶ `NuclearExtension(` Q, K `)` O

returns $[K_0, F, \varphi, \theta]$, where $K_0 \cong K$, $F \cong Q/K$, $\varphi : F \to \mathrm{Aut}K$ is a homomorphism, $\theta : F \times F \to K$ is a cocycle, and Q is isomorphic to $K_0 \times F$ with multiplication $(a, x)(b, y) = (a\varphi_x(b)\theta(x, y), xy)$.

If $n = |F|$ and $m = |K|$, the cocycle θ is returned as an $n \times n$ array with entries in $\{1, \dots, m\}$, and φ is returned as a list of length n of permutations of $\{1, \dots, m\}$.

Conversely, given K, F, $f = \varphi$, and $t = \theta$ as above,

2 ▶ `LoopByExtension(` K, F, f, t `)` O

returns the extension of K by F using the action f and cocycle t.

```
gap> F := IntoLoop( Group( (1,2) ) );
<loop of order 2>
gap> K := DirectProduct( F, F );;
gap> phi := [ (), (2,3) ];;
gap> theta := [ [ 1, 1 ], [ 1, 3 ] ];;
gap> LoopByExtension( K, F, phi, theta );
<loop of order 8>
gap> IsAssociative( last );
false
```

4.8 Random quasigroups and loops

We say that an algorithm selects a latin square of order n *at random* if every latin square of order n is returned by the algorithm with the same probability. Selecting a latin square at random is a nontrivial problem.

In [10], Jacobson and Matthews defined a random walk on the space of latin squares and so-called *improper latin squares* that visits every latin square with the same probability. The diameter of the space is no more than $4(n-1)^3$, that is, no more than $4(n-1)^3$ steps are needed to travel from one latin square of order n to another.

The Jacobson-Matthews algorithm can be used to generate random quasigroups as follows: (i) select any latin square of order n, for instance the canonical multiplication table of the cyclic group of order n, (ii) perform sufficiently many steps of the random walk, stopping at a proper or improper latin square, (iii) if necessary, perform a few more steps to end up with a proper latin square. Upon normalizing the resulting latin square, we obtain a random loop of order n.

By the above result, it suffices to use about n^3 steps to arrive at any latin square of order n from the initial latin square. In fact, a much smaller number of steps is probably sufficient.

1 ▶ `RandomLoop(` n `)` O

returns a random loop of order n, using the Jacobson-Matthews algorithm with n^3 steps. Every loop of order n is potentially returned, but the method is slow for $n > 50$, say. You can control the trade-off between randomness and speed with

2 ▶ `RandomLoop(` n, *iter* `)` O

returns a random loop of order n, using the Jacobson-Matthews algorithm with *iter* many steps. When *iter* is small, the returned loop will be close to the canonical Cayley table of the cyclic group of order n.

The corresponding commands for random quasigroups are

3 ▶ `RandomQuasigroup(` n `)` O
▶ `RandomQuasigroup(` n, *iter* `)` O

Finally,

4 ▶ `RandomNilpotentLoop(` *lst* `)` O

returns a random nilpotent loop as follows (see Section 6.10 for more on nilpotency):

lst must be a list of positive integers and/or finite abelian groups. If *lst* $= [a_1]$ and a_1 is an integer, a random abelian group of order a_1 is returned, else a_1 is an abelian group and `AsLoop(` a_1 `)` is returned. If *lst* $= [a_1, \ldots, a_m]$, a random central extension of `RandomNilpotentLoop(` $[a_1]$ `)` by `RandomNilpotentLoop(` $[a_2, \ldots, a_m]$ `)` is returned.

To determine the nilpotency class c of the resulting loop, assume that *lst* has length at least 2, contains only integers bigger than 1, and let m be the last entry of *lst*. If $m > 2$ then $c =$ `Length(`*lst*`)`, else $c =$ `Length(`*lst*`)`-1.

4.9 Conversions

As we have already mentioned, LOOPS contains methods that convert between magmas, quasigroups, loops and groups, provided such conversions are possible.

If M is a declared magma that happens to be a quasigroup, the corresponding quasigroup is returned via

1 ▶ `IntoQuasigroup(` M `)` O

Note that if M is already declared as a quasigroup, `IntoQuasigroup(` M `)` merely returns M.

Given a quasigroup M and two of its elements f, g, the principal loop isotope $x \circ y = R_g^{-1}(x) \cdot L_f^{-1}(y)$ turns $(M, \circ)$ into a loop with neutral element $f \cdot g$ (see Section 2.3). Since loops in LOOPS have to have neutral element labeled as 1, the function

2 ▶ `PrincipalLoopIsotope(` M, f, g `)`

returns an isomorphic copy of the principal isotope $(M, \circ)$ via the transposition $(1, f \cdot g)$.

If M is a declared magma that happens to be a quasigroup (not necessarily a loop!), the operation

3 ▶ `IntoLoop(` M `)` O

returns a loop L as follows:

- ∘ if M is already declared as a loop, $L = M$, else
- ∘ if M possesses a neutral element e and f is the first element of M, then L is an isomorphic copy of M via the transposition (e, f), else
- ∘ if M does not posses a neutral element, L is returned as `PrincipalLoopIsotope(` M, *M.1*, *M.1* `)`

One could obtain a loop from M in yet another way, by normalizing the Cayley table of M. These three approaches can result in nonisomorphic loops in general.

Finally, when M is a declared magma that happens to be a group, the corresponding group is returned by

4 ▶ `IntoGroup(` M `)` O

as follows:

- if M is already declared as a group, M is returned, else
- `RightMultiplciationGroup( IntoLoop(` M `) )` is returned, i.e., a permutation group isomorphic to M.

All three conversion methods return `fail` if the requested conversion is not possible.

Up to version 2.0.0, we have supported `AsQuasigroup`, `AsLoop` and `AsGroup` in place of `IntoQuasigroup`, `IntoLoop` and `IntoGroup`, respectively. We have changed the terminology starting with version 2.1.0 in order to comply with GAP naming rules for `AsSomething`, as explained in Chapter 3. Finally, the method `AsGroup` is built into the core of GAP and it returns an fp group if its argument is an associative loop.

4.10 Products of loops

Let $L1$, ..., Ln be a list consisting of loops and groups, where $n \geq 1$. Then

1 ► `DirectProduct(` *L1* `, ...,` *Ln*`)` O

returns the direct product of $L1$, ..., Ln.

If there are only groups among $L1$, ..., Ln, a group is returned. Otherwise a loop is returned. If $n = 1$, $L1$ is returned.

4.11 Opposite quasigroups and loops

When Q is a quasigroup with multiplication $\cdot$, the *opposite quasigroup* of Q is a quasigroup with the same underlying set as Q and with multiplication $*$ defined by $x * y = y \cdot x$.

Since the quasigroup-theoretical concepts are often chiral (cf. left Bol loops versus right Bol loops), it is useful to have access to the opposite quasigroup of Q:

1 ► `Opposite(` *Q* `)` O

5 Basic methods and attributes

We describe the basic core methods and attributes of the LOOPS package in this chapter.

5.1 Basic attributes

We associate many attributes with quasigroups in order to speed up computation. This section lists some basic attributes of quasigroups and loops.

The list of elements of a quasigroup Q is obtained by the usual command

1 ▶ `Elements(` *Q* `)` A

The Cayley table of a quasigroup Q is returned with

2 ▶ `CayleyTable(` *Q* `)` A

One can use `Display( CayleyTable(` *Q* `) )` for pretty matrix-style output of small Cayley tables.

The neutral element of a loop L is obtained via

3 ▶ `One(` *L* `)` A

If you want to know if a quasigroup Q has a neutral element, you can find out with the standard function for magmas

4 ▶ `MultiplicativeNeutralElement(` *Q* `)` A

The size of a quasigroup Q is calculated by

5 ▶ `Size(` *Q* `)` A

When L is a *power associative loop*, i.e., the powers of elements are well-defined in L, the *exponent* of L is the smallest positive integer divisible by orders of all elements of L. The following attribute calculates the exponent without testing for power-associativity:

6 ▶ `Exponent(` *L* `)` A

5.2 Basic arithmetic operations

Each quasigroup element in GAP knows which quasigroup it belongs to. It is therefore possible to perform arithmetic operations with quasigroup elements without referring to the quasigroup. All elements involved in the calculation must belong to the same quasigroup.

Two elements x, y of the same quasigroup are multiplied by $x * y$ in GAP. Since multiplication of at least three elements is ambiguous in the nonassociative case, we by default parenthesize elements from left to right, i.e., $x * y * z$ means $((x * y) * z)$. Of course, one can specify the order of multiplications by providing parentheses.

Universal algebraists introduce two additional operations for quasigroups. Namely the *left division* $x\backslash y$ defined by $x \cdot (x\backslash y) = y$, and the *right division* x/y defined by $(x/y) \cdot y = x$. These two operations can be found in LOOPS as:

1 ▶ `LeftDivision(` *x*, *y* `)` O
▶ `RightDivision(` *x*, *y* `)` O

When Q is a quasigroup, x is an element of Q, and S is a list of elements of Q, then

2 ▶ `LeftDivision(` S, x `)` O
▶ `LeftDivision(` x, S `)` O
▶ `RightDivision(` S, x `)` O
▶ `RightDivision(` x, S `)` O

returns the list of elements obtained by performing the respective division of S by x, or of x by S, using one element of S at a time.

We also support / in place of `RightDivision`. But we do not support \ in place of `LeftDivision`.

To obtain the Cayley tables for the operation left division or right division, use

3 ▶ `LeftDivisionCayleyTable(` Q `)` O
▶ `RightDivisionCayleyTable(` Q `)` O

5.3 Powers and inverses

Powers of elements are generally not well-defined in quasigroups. We say that the quasigroup Q is *power associative*, if for any $x \in Q$, the submagma generated by x is associative.

For magmas and positive integer exponents, GAP defines the powers in the following way: $x^1 = x$, $x^{2k} = (x^k) \cdot (x^k)$ and $x^{2k+1} = (x^{2k}) \cdot x$. One can easily see that this returns x^k in $\log_2(k)$ steps. For LOOPS, we have decided to keep this method, hoping that everybody will use it with care for quasigroups that are not power associative.

Let x be an element of a loop L with neutral element 1. Then the *left inverse* x^λ of x is the unique element of L satisfying $x^\lambda x = 1$. Similarly, the *right inverse* x^ρ satisfies $xx^\rho = 1$. If $x^\lambda = x^\rho$, we call $x^{-1} = x^\lambda = x^\rho$ the *inverse* of x.

1 ▶ `LeftInverse(` x `)` O
▶ `RightInverse(` x `)` O
▶ `Inverse(` x `)` O

The following examples illustrates the concept of inverses for a given loop M in which $M.i$ coincides with the ith element:

```
gap> CayleyTable( M );
[ [ 1, 2, 3, 4, 5 ],
  [ 2, 1, 4, 5, 3 ],
  [ 3, 4, 5, 1, 2 ],
  [ 4, 5, 2, 3, 1 ],
  [ 5, 3, 1, 2, 4 ] ]
gap> [ LeftInverse( M.3 ), RightInverse( M.3 ), Inverse( M.3 ) ];
[ l5, l4, fail ]
```

5.4 Associators and commutators

Let Q be a quasigroup and x, y, $z \in Q$. Then the *associator* of x, y, z is the unique element u such that $(xy)z = (x(yz))u$. The *commutator* of x, y is the unique element v such that $xy = (yx)v$.

1 ▶ `Associator(` x, y, z `)` O
▶ `Commutator(` x, y `)` O

5.5 Generators

The following two attributes are synonyms of `GeneratorsOfMagma`:

1 ▶ `GeneratorsOfQuasigroup(` Q `)` A
▶ `GeneratorsOfLoop(` L `)` A

As usual in GAP, one can refer to the ith generator of a quasigroup Q by `Q.i`. Note that it is not necessarily the case that `Q.i = Elements( Q )[ i ]`, since the set of generators can be a proper subset of the elements.

It is easy to prove that a quasigroup of order n can be generated by a subset containing at most $\log_2 n$ elements. When Q is a quasigroup

2 ▶ `GeneratorsSmallest(` Q `)` A

returns a generating set $\{q_0, \ldots, q_m\}$ of Q such that $Q_0 = \emptyset$, $Q_m = Q$, $Q_i = \langle q_1, \ldots, q_i \rangle$, q_{i+1} is the least element of $Q \setminus Q_i$.

6 Methods based on permutation groups

Most calculations in the LOOPS package are delegated to groups, taking advantage of the various permutations and permutation groups associated with quasigroups. This chapter explains in detail how the permutations associated with a quasigroup are calculated, and it also describes some of the core methods of LOOPS based on permutations. Additional core methods can be found in Chapter 7.

6.1 Parent of a quasigroup

Let Q be a quasigroup and S a subquasigroup of Q. Since the multiplication in S coincides with the multiplication in Q, it is reasonable not to store the multiplication table of S. However, the quasigroup S then must know that it is a subquasigroup of Q. In order to facilitate this relationship, we introduce the attribute

1 ▶ `Parent(` Q `)` A

for a quasigroup Q.

When Q is *not* created as a subquasigroup of another quasigroup, the attribute `Parent(` Q `)` is set to Q. When Q is created as a subquasigroup of a quasigroup H, we let `Parent(` Q `) := Parent(` H `)`. Thus, in effect, `Parent(` Q `)` is the largest quasigroup from which Q has been created.

Let Q be a quasigroup with parent P, where P is some n-element quasigroup. Let x be an element of Q. Then `x![1]` is the position of x among the elements of P, i.e., `x![1] = Position( Elements( P ), x )`. The position of x among the elements of Q is obtained via

2 ▶ `Position(` Q`, x )` O

While referring to elements of Q by their positions, we therefore must decide if the positions are meant among the elements of Q, or among the elements of P. Since it requires no calculation to obtain `x![1]`, *we always use the position of an element in its parent quasigroup*. In this way, many attributes of a quasigroup, including its Cayley table, are permanently tied to its parent. It is now clear why we have not insisted that Cayley tables of quasigroups must have entries covering the entire interval 1, …, m, for some m.

When S is a list of quasigroup elements, not necessarily from the same quasigroup, the operation

3 ▶ `PosInParent(` S `)`

returns the list of positions of elements of S in the corresponding parent, i.e., `PosInParent(` S `)[ i ] = S[ i ]![ 1 ] = Position( Parent( S[ i ] ), S[ i ] )`.

6.2 Comparing quasigroups with common parent

Assume that A, B are two quasigroups with common parent Q. Let G_A, G_B be the canonical generating sets of A and B, respectively, obtained by the method `GeneratorsSmallest`, described above. Then we define $A < B$ if and only if $G_A < G_B$ lexicographically.

Note that if A is a subquasigroup of B, we get $A < B$, but not necessarily vice versa.

6.3 Subquasigroups and subloops

When S is a subset of a quasigroup Q (loop L), the smallest subquasigroup of Q (subloop of L) generated by S is returned via:

1 ► `Subquasigroup(` Q, S `)` O
► `Subloop(` L, S `)` O

In fact, we allow S to be a list of integers, too, representing the positions of the respective elements in the parent quasigroup (loop).

If S is empty, `Subloop(` L, S `)` returns the one-element subloop of L, while `Subquasigroup(` Q, S `)` returns the empty set. The empty set is obviously not a subquasigroup of Q, but this convention is useful for handling certain situation, for instance when the user calls `Center(` Q `)` for a quasigroup Q with empty center.

The following two operations test if a quasigroup (loop) S is a subquasigroup (subloop) of a quasigroup Q. They return `true` if and only if Q and S have the same parent, and if S is a subset of Q.

2 ► `IsSubquasigroup(` Q, S `)` O
► `IsSubloop(` Q, S `)` O

The operation

3 ► `AllSubloops(` L `)` O

returns a list of all subloops of the loop L.

Let S be a subloop of L. The list of all right cosets of S in L is obtained via

4 ► `RightCosets(` L, S `)` F

The coset S is listed first, and the elements of each coset are ordered in the same way as elements of S, i.e., if $S = [s_1, \ldots, s_m]$ then $Sx = [s_1x, \ldots, s_mx]$.

When S is a subloop of L, the right transversal of S with respect to L is a subset of L containing one element from each right coset of S in L. It is obtained by

5 ► `RightTransversal(` L, S `)` O

and it returns the first element from each right coset obtained by `RightCosets(` L, S `)`.

6.4 Translations and sections

When x is an element of a quasigroup Q, the left translation L_x is a permutation of Q. In LOOPS, all permutations associated with quasigroups and their elements are permutations in the sense of GAP, i.e., bijections of some interval $1, \ldots, n$. Moreover, following our convention, the numerical entries of the permutation point to the positions among elements of `Parent(` Q `)`, not Q.

The left and right translations by x in Q are obtained by

1 ► `LeftTranslation(` Q, x `)` O
► `RightTranslation(` Q, x `)` O

The following two attributes calculate the left and right section of a quasigroup Q:

2 ► `LeftSection(` Q `)` A
► `RightSection(` Q `)` A

Here is an example illustrating the main features of the subquasigroup construction and the relationship between a quasigroup and its parent.

Note how the Cayley table of the subquasigroup is created only upon explicit demand. Also note that changing the names of elements of a subquasigroup (subloop) automatically changes the names of the elements of the parent subquasigroup (subloop). This is because the elements are shared.

```
gap> M := MoufangLoop( 12, 1 );; S := Subloop( M, [ M.5 ] );
<loop of order 3>
gap> [ Parent( S ) = M, Elements( S ), PosInParent( S ) ];
[ true, [ l1, l3, l5], [ 1, 3, 5 ] ]
gap> HasCayleyTable( S );
false
gap> SetLoopElmName( S, "s" );; Elements( S ); Elements( M );
[ s1, s3, s5 ]
[ s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12 ]
gap> CayleyTable( S );
[ [ 1, 3, 5 ], [ 3, 5, 1 ], [ 5, 1, 3 ] ]
gap> LeftSection( S );
[ (), (1,3,5), (1,5,3) ]
gap> [ HasCayleyTable( S ), Parent( S ) = M ];
[ true, true ]
gap> L := LoopByCayleyTable( CayleyTable( S ) );; Elements( L );
[ l1, l2, l3 ]
gap> [ Parent( L ) = L, IsSubloop( M, S ), IsSubloop( M, L ) ];
[ true, true, false ]
gap> LeftSection( L );
[ (), (1,2,3), (1,3,2) ]
```

6.5 Multiplication groups

The left multiplication group, right multiplication group and the multiplication group of a quasigroup Q are calculated as follows:

1 ► `LeftMultiplicationGroup(` *Q* `)` A
► `RightMultiplicationGroup(` *Q* `)` A
► `MultiplicationGroup(` *Q* `)` A

Let S be a subloop of a loop L. Then the *relative left multiplication group* of L with respect to S is the group $\langle L(x)|x \in S\rangle$, where $L(x)$ is the left translation by x in Q restricted to S. The *relative right multiplication group* and *relative multiplication group* are defined analogously.

2 ► `RelativeLeftMultiplicationGroup(` *L*, *S* `)` O
► `RelativeRightMultiplicationGroup(` *L*, *S* `)` O
► `RelativeMultiplicationGroup(` *L*, *S* `)` O

See Section 8.4 for more on that topic.

6.6 Inner mapping groups

The *inner mapping group* of a loop L is the stabilizer of the unit element in $\mathrm{Mlt}(L)$. The elements of this stabilizer are called *inner maps* of L.

The *left inner mapping group* of a loop L is the stabilizer of the unit element in $\mathrm{LMlt}(L)$. The *right inner mapping group* is defined dually.

Equivalently, the left inner mapping group is generated by all *left inner mappings* $L(x,y) = L_{yx}^{-1}L_yL_x$, and the right inner mapping group is generated by all *right inner mappings* $R(x,y) = R_{xy}^{-1}R_yR_x$.

In analogy with group theory, we define the *conjugation*, or the *middle inner mapping* by x as $T(x) = L_x^{-1}R_x$. The *middle inner mapping grroup* is then the subgroup of $\mathrm{Mlt}(L)$ generated by all conjugations.

The corresponding commands in LOOPS are

1 ► `LeftInnerMapping(` *L*, *x*, *y* `)` O
► `MiddleInnerMapping(` *L*, *x* `)` O
► `RightInnerMapping(` *L*, *x*, *y* `)` O
► `LeftInnerMappingGroup(` *L* `)` A
► `MiddleInnerMappingGroup(` *L* `)` A
► `RightInnerMappingGroup(` *L* `)` A
► `InnerMappingGroup(` *L* `)` A

Here is an example for multiplication groups and inner mapping groups:

```
gap> M := MoufangLoop( 12, 1 );
<Moufang loop 12/1>
gap> LeftSection( M )[ 2 ];
(1,2)(3,4)(5,6)(7,8)(9,12)(10,11)
gap> Mlt := MultiplicationGroup( M ); Inn := InnerMappingGroup( M );
<permutation group of size 2592 with 23 generators>
Group([ (4,6)(7,11), (7,11)(8,10), (2,6,4)(7,9,11), (3,5)(9,11), (8,12,10) ])
gap> Size( Inn );
216
```

6.7 Nuclei, commutant, center, and associator subloop

Let Q be a quasigroup. The *left nucleus* $N_\lambda(Q)$ of Q is the set $\{x \in Q | x(yz) = (xy)z$ for every $y, z \in Q\}$. One defines similarly the *middle nucleus* $N_\mu(Q)$ and the *right nucleus* $N_\rho(Q)$. Then the *nucleus* $N(Q)$ of Q is the intersection of the three nuclei.

The nuclei are calculated in LOOPS as follows:

1 ► `LeftNucleus(` *Q* `)` A
► `MiddleNucleus(` *Q* `)` A
► `RightNucleus(` *Q* `)` A
► `Nuc(` *Q* `)` A

Unfortunately, the word `Nucleus` is reserved in the core of GAP for a global function with two variables. That is the reason why we have used the abbreviation `Nuc`, which is also common in the literature. However, we support these synonyms of `Nuc`:

2 ► `NucleusOfLoop(` *Q* `)` A
► `NucleusOfQuasigroup(` *Q* `)` A

Since all nuclei are subquasigroups of Q, they are returned as subquasigroups (resp. subloops). When Q is a loop then all nuclei are in fact groups, and they are returned as associative loops.

The *commutant* $C(Q)$ of Q is the set $\{x \in Q \,|\, xy = yx$ for every $y \in Q\}$. It is obtained via

3 ► `Commutant(` *Q* `)` A

The center $Z(Q)$ is defined as the intersection of $C(Q)$ and $N(Q)$, and it is obtained via

4 ► `Center(` *Q* `)` A

It is a subgroup of Q and is therefore returned as an associative loop.

Finally, the *associator subloop* of a loop L is the smallest normal subloop $A(L)$ of L containing all associators of L. Equivalently, $A(L)$ is the smallest normal subloop K such that L/K is associative. We use another equivalent reformulation for the purposes of computation: $A(L)$ is the smallest normal subloop of L containing $\{x \setminus \alpha(x) \,|\, x \in L, \alpha$ is a left inner mapping$\}$.

5 ► `AssociatorSubloop(` *L* `)` A

6.8 Normal subloops

A subloop S of a loop L is *normal* if it is invariant under all inner mappings of L. Normality is tested via:

1 ► `IsNormal(` *L*, *S* `)` O

When S is a subset of a loop L or a subloop of L, the *normal closure* of S in L is the smallest normal subloop of L containing S. It is obtained by

2 ► `NormalClosure(` *L*, *S* `)` O

A loop L is *simple* if all normal subloops of L are trivial. The corresponding test in LOOPS is:

3 ► `IsSimple(` *L* `)` O

6.9 Factor loops

When N is a normal subloop of a loop L, the factor loop L/N can be obtained directly via the command `L/N`, or by

1 ► `FactorLoop(` *L*, *N* `)` O

The natural projection from L to L/N is returned by

2 ► `NaturalHomomorphismByNormalSubloop(` *L*, *N* `)` O

Here is an illustrating example:

```
gap> M := MoufangLoop( 12, 1 );; S := Subloop( M, [ M.3 ] );
<loop of order 3>
gap> IsNormal( M, S );
true
gap> F := FactorLoop( M, S );
<loop of order 4>
gap> NaturalHomomorphismByNormalSubloop( M, S );
MappingByFunction( <loop of order 12>, <loop of order 4>,
    function( x ) ... end )
```

6.10 Nilpotency and central series

The definition of nilpotency and nilpotence class is the same as in group theory. The corresponding commands are:

1 ► `NilpotencyClassOfLoop(` *L* `)` A
► `IsNilpotent(` *L* `)` P

When L is not nilpotent, `NilpotencyClassOfLoop(` *L* `)` returns `fail`.

A loop L is said to be *strongly nilpotent* if its multiplication group is nilpotent. This property is obtained by

2 ► `IsStronglyNilpotent(` *L* `)` P

Let L be a loop. Define *iterated centers* $Z_i(L)$ as follows: $Z_0(L) = Z(L)$, $Z_{i+1}(L) = \pi_i^{-1}(Z_i(L))$, where π_i is the canonical projection $L \to L/Z_i(L)$. The longest series $Z_i(L)$, $Z_{i-1}(L)$, …, $Z_0(L)$ with $Z_i(L) > Z_{i-1}(L) > \cdots > Z_0(L)$ is called the *upper central series* of L, and is returned via

3 ► `UpperCentralSeries(` *L* `)` A

The *lower central series*, defined in the usual way, is obtained by

4 ► `LowerCentralSeries(` *L* `)` A

6.11 Solvability

The definition of solvability, derived subloop, derived length, Frattini subloop and Frattini factor size is the same as for groups. Frattini subloop is calculated only for strongly nilpotent loops.

1 ► `IsSolvable(` *L* `)` P
► `DerivedSubloop(` *L* `)` A
► `DerivedLength(` *L* `)` A
► `FrattiniSubloop(` *L* `)` A
► `FrattinifactorSize(` *L* `)` A

6.12 Isomorphisms and automorphisms

All isomorphisms between two loops can be found with LOOPS. The operation

1 ► `IsomorphismLoops(` *L*, *M* `)` O

returns a single isomorphism between loops L, M if the loops are isomorphic, and `fail` otherwise.

If an isomorphism exists, it is returned as a permutation p of 1, …, $|L|$, where $i^p = j$ means that the ith element of L is mapped onto the jth element of M. This is true even if L or M are not their own parents.

Since one frequently needs to filter a list of loops up to isomorphism, we support

2 ► `LoopsUpToIsomorphism(` *ls* `)` O

Given a list *ls* of loops, the operation returns a sublist of *ls* containing one loop from each isomorphism class of loops present in *ls*.

The attribute

3 ► `AutomorphismGroup(` *L* `)` A

returns the automorphism group of the loop L, with the same convention on permutations as in the case of `IsomorphismLoops`.

Since two isomorphisms "differ" by an automorphism, all isomorphisms can be obtained by the above two functions.

While dealing with Cayley tables, it is often useful to rename or reorder the elements without changing the isomorphism type. When Q is a quasigroup of size n and p is a permutation of $\{1, \dots, n\}$,

4 ► `IsomorphicCopyByPerm( Q, p )`

returns quasigroup $(Q, \circ)$ such that $p(xy) = p(x) \circ p(y)$, i.e., $x \circ y = p(p^{-1}(x)p^{-1}(y))$. When Q is a declared loop, a loop is returned. Consequently, when Q is a declared loop and $p(1) = k \neq 1$, then p is first replaced by $p \circ (1, k)$, to make sure that the Cayley table is normalized.

When S is a normal subloop of L,

5 ► `IsomorphicCopyByNormalSubloop( L, S )`

returns an isomorphic copy of L in which the elements are ordered according to the right cosets of S. In particular, the Cayley table of S will appear in the top left corner of the Cayley table of the resulting loop.

6.13 How are isomorphisms computed

In order to speed up the search for isomorphisms and automorphisms, we first calculate some loop invariants preserved under isomorphisms, and use these invariants to partition the loop into blocks of elements preserved under isomorphisms. These invariants for a loop L can be obtained via

1 ► `Discriminator(` *L* `)` O

Since the details are technical, we will not present them here. See [21] or the file `iso.gi` for more.

If two loops have different discriminators, they are not isomorphic. If they have identical discriminators, they may or may not be isomorphic. The operation

2 ► `AreEqualDiscriminators(` *D1*, *D2* `)` O

returns `true` if the discriminators $D1$, $D2$ are equal.

6.14 Isotopisms

The package contains a slow method for testing if two loops are isotopic:

1 ► `IsotopismLoops(` *K*`,` *L* `)` O

It returns `fail` if K, L are not isotopic, else it returns an isotopism as a triple of bijections on $\{1, \ldots, |K|\}$.

The method works as follows: It is well known that if a loop K is isotopic to a loop L then there exist a principal loop isotope P of K such that P is isomorphic to L. The algorithm first finds all principal isotopes of K, then filters them up to isomorphism, and then checks if any of them is isomorphic to L. This is rather slow already for small orders, say 30.

The function

2 ► `LoopsUpToIsotopism(` *ls* `)` O

filters the list *ls* in a way similar to `LoopsUpToIsomorphism`, but using isotopism as the underlying equivalence relation.

7 Testing properties of quasigroups and loops

The reader should be aware that although loops are quasigroups, it is often the case in the literature that a property named P can differ for quasigroups and loops. For instance, a Steiner loop is not necessarily a Steiner quasigroup.

To avoid such ambivalences, we often include the noun `Loop` or `Quasigroup` as part of the name of the property, e.g. `IsSteinerQuasigroup` versus `IsSteinerLoop`.

On the other hand, some properties coincide for quasigroups and loops and we therefore do not include `Loop`, `Quasigroup` as part of the name of the property, e.g. `IsCommutative`.

7.1 Associativity, commutativity and generalizations

The following properties test if a quasigroup Q is associative and commutative:

1 ► `IsAssociative(` Q `)` P
► `IsCommutative(` Q `)` P

A loop L is said to be *power associative* (resp. *diassociative*) if every monogenic subloop of L (resp. every 2-generated subloop of L) is a group.

2 ► `IsPowerAssociative(` L `)` P
► `IsDiassociative(` L `)` P

7.2 Inverse properties

A loop L has the *left inverse property* if $x^\lambda(xy) = y$ for every x, $y \in L$, where x^λ is the left inverse of x. Dually, L has the *right inverse property* if $(yx)x^\rho = y$ for every x, $y \in L$, where x^ρ is the right inverse of x. If L has both the left and right inverse properties, it has the *inverse property*. We say that L has *two-sided inverses* if $x^\lambda = x^\rho$ for every $x \in L$.

1 ► `HasLeftInverseProperty(` L `)` P
► `HasRightInverseProperty(` L `)` P
► `HasInverseProperty(` L `)` P
► `HasTwosidedInverses(` L `)` P

A loop has the *weak inverse property* if $(xy)^\lambda x = y^\lambda$. Equivalently, a loop has the weak inverse property if $x(yx)^\rho = y^\rho$.

2 ► `HasWeakInverseProperty(` L `)` P

According to [1], a loop L has the *automorphic inverse property* if $(xy)^\lambda = x^\lambda y^\lambda$, or, equivalently, $(xy)^\rho = x^\rho y^\rho$. (In particular, when L has two-sided inverses and the automorphic inverse property, it satisfies $(xy)^{-1} = x^{-1}y^{-1}$.) Similarly, L has the *antiautomorphic inverse property* if $(xy)^\lambda = y^\lambda x^\lambda$, or, equivalently, $(xy)^\rho = y^\rho x^\rho$.

3 ► `HasAutomorphicInverseProperty(` L `)` P
► `HasAntiautomorphicInverseProperty(` L `)` P

The following implications among inverse properties hold and are implemented in LOOPS:

- Inverse property implies left and right inverse properties, two-sided inverses, weak inverse property, and antiautomorphic inverse property.
- Antiautomorphic inverse property loops have two-sided inverses.
- If a loop has any two of the left inverse property, right inverse property, weak inverse property or antiautomorphic inverse property, it also has the inverse property.

7.3 Some properties of quasigroups

A quasigroup Q is *semisymmetric* if $(xy)x = y$ for every x, $y \in Q$. Equivalently, Q is semisymmetric if $x(yx) = y$ for every x, $y \in Q$. A semisymmetric commutative quasigroup is known as *totally symmetric*. Totally symmetric quasigroups are precisely quasigroups satisfying $xy = x\backslash y = x/y$.

1 ► `IsSemisymmetric(` *Q* `)` P
► `IsTotallySymmetric(` *Q* `)` P

A quasigroup Q is *idempotent* if $x^2 = x$ for every $x \in Q$. Idempotent totally symmetric quasigroups are known as *Steiner quasigroups*. A quasigroup Q is *unipotent* if $x^2 = y^2$ for every x, $y \in Q$.

2 ► `IsIdempotent(` *Q* `)` P
► `IsSteinerQuasigroup(` *Q* `)` P
► `IsUnipotent(` *Q* `)` P

A quasigroup is *left distributive* if it satisfies $x(yz) = (xy)(xz)$. Similarly, it is *right distributive* if it satisfies $(xy)z = (xz)(yz)$. A *distributive quasigroup* is a quasigroup that is both left and right distributive. A quasigroup is called *entropic* or *medial* if it satisfies $(xy)(zw) = (xz)(yw)$.

3 ► `IsLeftDistributive(` *Q* `)` P
► `IsRightDistributive(` *Q* `)` P
► `IsDistributive(` *Q* `)` P
► `IsEntropic(` *Q* `)` P
► `IsMedial(` *Q* `)` P

In order to be compatible with GAP's terminology, we also support the synonyms

4 ► `IsLDistributive(` *Q* `)` P
► `IsRDistributive(` *Q* `)` P

for `IsLeftDistributive` and `IsRightDistributive` respectively.

7.4 Loops of Bol-Moufang type

Following [8] and [19], a variety of loops is said to be of *Bol-Moufang type* if it is defined by a single *identity of Bol-Moufang type*, i.e., by an identity that:

- contains the same 3 variables on both sides,
- exactly one of the variables occurs twice on both sides,
- the variables occur in the same order on both sides.

It is proved in [19] that there are 13 varieties of nonassociative loops of Bol-Moufang type. These are:

- *left alternative loops*, defined by $x(xy) = (xx)y$,
- *right alternative loops*, defined by $x(yy) = (xy)y$,
- *left nuclear square loops*, defined by $(xx)(yz) = ((xx)y)z$,
- *middle nuclear square loops*, defined by $x((yy)z) = (x(yy))z$,
- *right nuclear square loops*, defined by $x(y(zz)) = (xy)(zz)$,
- *flexible loops*, defined by $x(yx) = (xy)x$,
- *left Bol loops*, defined by $x(y(xz)) = (x(yx))z$, always left alternative,
- *right Bol loops*, defined by $x((yz)y) = ((xy)z)y$, always right alternative,
- *LC-loops*, defined by $(xx)(yz) = (x(xy))z$, always left alternative, left and middle nuclear square,
- *RC-loops*, defined by $x((yz)z) = (xy)(zz)$, always right alternative, right and middle nuclear square,
- *Moufang loops*, defined by $(xy)(zx) = (x(yz))x$, always flexible, left and right Bol,
- *C-loops*, defined by $x(y(yz)) = ((xy)y)z$, always LC and RC,
- *extra loops*, defined by $x(y(zx)) = ((xy)z)x$, always Moufang and C.

Note that although some of the defining identities are not of Bol-Moufang type, they are equivalent to a Bol-Moufang identity. Moreover, many varieties are defined in several ways, by equivalent identities of Bol-Moufang type.

There are several varieties related to loops of Bol-Moufang type. A loop is said to be *alternative* if it is both left and right alternative, and *nuclear square* if it is left, middle and right nuclear square.

Here are the corresponding LOOPS commands (argument L indicates that the property applies only to loops, argument Q indicates that the property applies also to quasigroups):

1 ▶ `IsExtraLoop( L )` P
▶ `IsMoufangLoop( L )` P
▶ `IsCLoop( L )` P
▶ `IsLeftBolLoop( L )` P
▶ `IsRightBolLoop( L )` P
▶ `IsLCLoop( L )` P
▶ `IsRCLoop( L )` P
▶ `IsLeftNuclearSquareLoop( L )` P
▶ `IsMiddleNuclearSquareLoop( L )` P
▶ `IsRightNuclearSquareLoop( L )` P
▶ `IsNuclearSquareLoop( L )` P
▶ `IsFlexible( Q )` P
▶ `IsLeftAlternative( Q )` P
▶ `IsRightAlternative( Q )` P
▶ `IsAlternative( Q )` P

While listing the varieties of loops of Bol-Moufang type, we have also listed all inclusions among them. These inclusions are built into LOOPS.

The following trivial example shows some of the implications and the naming conventions of LOOPS at work:

```
gap> L := LoopByCayleyTable( [ [ 1, 2 ], [ 2, 1 ] ] );
<loop of order 2>
gap> [ IsLeftBolLoop( L ), L ]
[ true, <left Bol loop of order 2> ]
gap> [ HasIsLeftAlternativeLoop( L ), IsLeftAlternativeLoop( L ) ];
[ true, true ]
gap> [ HasIsRightBolLoop( L ), IsRightBolLoop( L ) ];
[ false, true ]
gap> L;
<Moufang loop of order 2>
gap> [ IsAssociative( L ), L ];
[ true, <associative loop of order 2> ]
```

The analogous terminology for quasigroups of Bol-Moufang type is not standard yet, and hence is not supported in LOOPS.

7.5 Power alternative loops

A loop is *left power alternative* if it is power associative and $x^n(x^m y) = x^{n+m}y$ for every x, y and all integers n, m. Similarly, a loop is *right power alternative* if it is power associative and $(xy^n)y^m = xy^{n+m}$ for all x, y and all integers n, m. A loop that is both left and right power alternative is said to be *power alternative*.

Left power alternative loops are left alternative and have the left inverse property. Left Bol loops and LC-loops are left power alternative.

1 ▶ `IsLeftPowerAlternative( L )` P
▶ `IsRightPowerAlternative( L )` P
▶ `IsPowerAlternative( L )` P

7.6 Conjugacy closed loops and related properties

A loop is *left* (resp. *right*) *conjugacy closed* if its left (resp. right) translations are closed under conjugation. A loop that is both left and right conjugacy closed is called *conjugacy closed*. It is common to refer to these loops as LCC-, RCC-, CC-loops, respectively.

1 ► `IsLCCLoop(` *L* `)` P
► `IsRCCLoop(` *L* `)` P
► `IsCCLoop(` *L* `)` P

The equivalence LCC + RCC = CC is built into LOOPS.

A loop is *Osborn* if it satisfies $x(yz \cdot x) = (x^\lambda \backslash y)(zx)$, where x^λ is the left inverse of x. Both Moufang loops and CC-loops are Osborn.

2 ► `IsOsbornLoop(` *L* `)` P

7.7 Additional varieties of loops

An *(even) code loop* is a Moufang 2-loop with Frattini subloop of order 1 or 2. Code loops are extra and conjugacy closed.

1 ► `IsCodeLoop(` *L* `)` P

Steiner loop is an inverse property loop of exponent 2. Steiner loops are commutative.

2 ► `IsSteinerLoop(` *L* `)` P

A left (resp. right) Bol loop with the automorphic inverse property is known as *left* (resp. *right*) *Bruck loop*. Bruck loops are also known as *K-loops*.

3 ► `IsLeftBruckLoop(` *L* `)` P
► `IsLeftKLoop(` *L* `)` P
► `IsRightBruckLoop(` *L* `)` P
► `IsRightKLoop(` *L* `)` P

A loop whose all left (resp. middle, right) inner mappings are automorphisms is called a *left* (resp. *middle*, *right*) *automorphic loop*. A loop whose every inner mapping is an automorphism is known as an *automorphic loop*. Diassociative automorphic loops are Moufang by [11]. See the built-in filters for additional properties of automorphic loops.

4 ► `IsLeftAutomorphicLoop(` *L* `)` P
► `IsMiddleAutomorphicLoop(` *L* `)` P
► `IsRightAutomorphicLoop(` *L* `)` P
► `IsAutomorphicLoop(` *L* `)` P

Automorphic loops have historically been called *A-loops*. We therefore support the synonyms

5 ► `IsLeftALoop(` *L* `)` P
► `IsMiddleALoop(` *L* `)` P
► `IsRightALoop(` *L* `)` P
► `IsALoop(` *L* `)` P

Be careful not to confuse `IsALoop` and `IsLoop`.

8 Specific methods

This chapter describes methods of LOOPS that apply to some special loops, mostly Bol and Moufang loops.

8.1 Core methods for Bol loops

Let L be a left Bol loop such that the mapping $x \mapsto x^2$ is a permutation of L. Define a new operation $*$ on L by $x * y = (x(y^2x))^{1/2}$. Then Bruck showed that $(L, *)$ is a left Bruck loop, called the *assoicated left Bruck loop.*. (In fact, Bruck used the isomorphic operation $x*y = x^{1/2}(yx^{1/2})$ instead. Our approach is more natural, since the associated left Bruck loop to a left Bruck loop L is identical to L then.) The associated left Bruck loop of L is returned via

1 ▶ `AssociatedLeftBruckLoop(` L `)` A

8.2 Moufang modifications

Aleš Drápal discovered two prominent families of extensions of Moufang loops. It turns out that these extensions can be used to obtain all nonassociative Moufang loops of order at most 64. We call these two constructions *Moufang modifications*. The library of Moufang loops included with LOOPS is based on Moufang modifications. We describe the two modifications briefly here. See [7] for details.

Assume that L is a Moufang loop with normal subloop S such that L/S is a cyclic group of order $2m$. Let $h \in S \cap Z(L)$. Let α be a generator of L/S and write $L = \bigcup_{i\in M} \alpha^i$, where $M = \{-m+1, \dots, m\}$. Let $\sigma : \mathbb{Z} \to M$ be defined by $\sigma(i) = 0$ if $i \in M$, $\sigma(i) = 1$ if $i > m$, and $\sigma(i) = -1$ if $i < -m+1$. Introduce a new multiplication $*$ on L defined by

$$x * y = xyh^{\sigma(i+j)},$$

where $x \in \alpha^i$, $y \in \alpha^j$, $i \in M$, $j \in M$. Then $(L, *)$ is a Moufang loop, a *cyclic modification* of L.

When L, S, α, h are as above and when a is any element of α, the corresponding cyclic modification is obtained via

1 ▶ `LoopByCyclicModification(` L, S, a, h `)` F

Now assume that L is a Moufang loop with normal subloop S such that L/S is a dihedral group of order $4m$, with $m \geq 1$. Let M and σ be defined as in the cyclic case. Let β, $\gamma \in L/S$ be two involutions of L/S such that $\alpha = \beta\gamma$ generates a cyclic subgroup of L/S of order $2m$. Let $e \in \beta$ and $f \in \gamma$ be arbitrary. Then L can be written as a disjoint union $L = \bigcup_{i\in M}(\alpha^i \cup e\alpha^i)$, and also $L = \bigcup_{i\in M}(\alpha^i \cup \alpha^i f$. Let $G_0 = \bigcup_{i\in M} \alpha^i$, and $G_1 = L \setminus G_0$. Let $h \in S \cap N(L) \cap Z(G_0)$. Introduce a new multiplication $*$ on L defined by

$$x * y = xyh^{(-1)^r\sigma(i+j)},$$

where $x \in \alpha^i \cup e\alpha^i$, $y \in \alpha^j \cup \alpha^j f$, $i \in M$, $j \in M$, $y \in G_r$, $r \in \{0,1\}$. Then $(L, *)$ is a Moufang loop, a *dihedral modification* of L.

When L, S, e, f and h are as above, the corresponding dihedral modification is obtained via

2 ▶ `LoopByDihedralModification(` L, S, e, f, h `)` F

In order to apply the cyclic and dihedral modifications, it is beneficial to have access to a class of nonassociative Moufang loops. The following construction is due to Chein:

Let G be a group. Let $\overline{G} = \{\overline{g};\ g \in G\}$ be a set of new elements. Define multiplication $*$ on $L = G \cup \overline{G}$ by

$$g * h = gh,\ g * \overline{h} = \overline{hg},\ \overline{g} * h = \overline{gh^{-1}},\ \overline{g} * \overline{h} = h^{-1}g,$$

where g, $h \in G$. Then $L = M(G, 2)$ is a Moufang loop that is nonassociative if and only if G is nonabelian.

The loop $M(G, 2)$ can be obtained from a finite group G with

3 ▶ `LoopMG2(` G `)` F

8.3 Triality for Moufang loops

Let G be a group and σ, ρ be automorphisms of G, satisfying $\sigma^2 = \rho^3 = (\sigma\rho)^2 = 1$. We write the automorphisms of a group as exponents and $[g,\sigma]$ for $g^{-1}g^{\sigma}$. We say that the triple (G,ρ,σ) is a *group with triality* if $[g,\sigma][g,\sigma]^{\rho}[g,\sigma]^{\rho^2} = 1$ holds for all $g \in G$. It is known that one can associate a group with triality (G,ρ,σ) in a canonical way with a Moufang loop L. See [16] for more details.

For any Moufang loop L, we can calculate the triality group as a permutation group acting on $3|L|$ points. If the multiplication group of L is polycyclic, then we can also represent the triality group as a pc group. In both cases, the automorphisms σ and ρ are in the same family as the elements of G.

Given a Moufang loop L, the function

1 ▶ `TrialityPermGroup(` *L* `)` F

returns a record $[G, \rho, \sigma]$, where G is the group with triality associated with L, and ρ, σ are the corresponding triality automorphisms.

The function

2 ▶ `TrialityPcGroup(` *L* `)` F

differs from `TrialityPermGroup` only in that G is returned as a pc group.

8.4 Realizing groups as multiplication groups of loops

The following commands look for loops such that the multiplication group is contained in a given transitive permutation group G.

The resulting loops are given by their right sections.

1 ▶ `AllLoopTablesInGroup(` *G*`[,` *depth*`[,` *infolevel*`]] )` O
▶ `AllProperLoopTablesInGroup(` *G*`[,` *depth*`[,` *infolevel*`]] )` O
▶ `OneLoopTableInGroup(` *G*`[,` *depth*`[,` *infolevel*`]] )` O
▶ `OneProperLoopTableInGroup(` *G*`[,` *depth*`[,` *infolevel*`]] )` O
▶ `AllLoopsWithMltGroup(` *G*`[,` *depth*`[,` *infolevel*`]] )` O
▶ `OneLoopWithMltGroup(` *G*`[,` *depth*`[,` *infolevel*`]] )` O

One can speed up the search by setting the argument *depth* higher, the price is much higher memory consumption. *depth* is optimally chosen if in the permutation group G, there are not many permutations fixing *depth* elements. You can omit the argument *depth* or set it equal to 2 with little harm.

The parameter *infolevel* determines the amount of information you get during the search. With *infolevel*=0, no information is provided. With *infolevel*=1, you get the information on timing and hits. With *infolevel*=2, the results are printed, as well.

```
gap> g:=PGL(3,3);
Group([ (6,7)(8,11)(9,13)(10,12), (1,2,5,7,13,3,8,6,10,9,12,4,11) ])
gap> a:=AllLoopTablesInGroup(g,3,0);; Size(a);
56
gap> a:=AllLoopsWithMltGroup(g,3,0);; Size(a);
52
```

9 Libraries of small loops

Libraries of small loops form an integral part of LOOPS. Loops in libraries are stored up to isomorphism or up to isotopism. The name of a library up to isotopism starts with *itp*.

9.1 A typical library

A library named *my Library* is stored in file `data/mylibrary.tbl`, and the corresponding data structure is named `LOOPS_my_library_data`.

In most cases, the array `my_library_data` consists of three lists

- `LOOPS_my_library_data[ 1 ]` is a list of orders for which there is at least one loop in the library,
- `LOOPS_my_library_data[ 2 ][ k ]` is the number of loops of order `LOOPS_my_library_data[ 1 ][ k ]` in the library,
- `LOOPS_my_library_data[ 3 ][ s ]` contains data necessary to produce the *s*th loop in the library.

The format of `LOOPS_my_library_data[ 3 ]` depends on the particular library and is not standardized in any way.

The user can retrieve the *m*th loop of order *n* from library named *my Library* according to the template

1 ▶ `MyLibraryLoop(` *n*`,` *m* `)` F

It is also possible to obtain the same loop with

2 ▶ `LibraryLoop(` *name*`,` *n*`,` *m* `)` F

where *name* is the name of the library.

For example, when the library is called *left Bol*, the corresponding data file is called `data/leftbol.tbl`, the corresponding data structure is named `LOOPS_left_bol_data`, and the *m*th left Bol loop of order *n* is obtained via

`LeftBolLoop(` *n*`,` *m* `)`

or via

`LibraryLoop("left Bol",` *n*`,` *m* `)`

We are now going to describe the individual libraries in detail. A brief information about the library named *name* can also be obtained in LOOPS with

3 ▶ `DisplayLibraryInfo(` *name* `)` F

9.2 Left Bol loops

The library named *left Bol* contains all nonassociative left Bol loops of order less than 17, including Moufang loops. There are 6 such loops of order 8, 1 of order 12, 2 of order 15, and 2038 of order 16. (The classification of left Bol loops of order 16 was first accomplished by Moorhouse [14]. Our library was generated independently, and agrees with Moorhouse's results.)

Following the general pattern, the *m*th nonassociative left Bol loop of order *n* is obtained by

1 ▶ `LeftBolLoop(` *n*`,` *m* `)` F

9.3 Moufang loops

The library named *Moufang* contains all nonassociative Moufang loops of order $n \leq 64$ and $n \in \{81, 243\}$.

The mth nonassociative Moufang loop of order n is obtained by

1 ► `MoufangLoop(` n, m `)` F

For $n \leq 63$, our catalog numbers coincide with those of Goodaire et al. [9]. The classification of Moufang loops of order 64 and 81 was carried out in [17]. The classification of Moufang loops of order 243 was carried out by Slattery and Zenisek [20].

The extent of the library is summarized below:

order	12	16	20	24	28	32	36	40	42	44	48	52	54	56	60	64	81	243
loops in the libary	1	5	1	5	1	71	4	5	1	1	51	1	2	4	5	4262	5	72

The *octonion loop* of order 16 (i.e., the multiplication loop of the ± basis elements in the 8-dimensional standard real octonion algebra) is `MoufangLoop( 16, 3 )`.

9.4 Code loops

The library named *code* contains all nonassociative code loops of order less than 65. There are 5 such loops of order 16, 16 of order 32, and 80 of order 64, all Moufang. The library merely points to the corresponding Moufang loops. See [17] for a classification of small code loops.

The mth nonassociative code loop of order n is obtained by

1 ► `CodeLoop(` n, m `)` F

9.5 Steiner loops

Here is how the libary *Steiner* is described within LOOPS:

```
gap> DisplayLibraryInfo( "Steiner" );
The library contains all nonassociative Steiner loops of order less or equal to 16.
It also contains the associative Steiner loops of order 4 and 8.
------
Extent of the library:
   1 loop of order 4
   1 loop of order 8
   1 loop of order 10
   2 loops of order 14
   80 loops of order 16
true
```

The mth Steiner loop of order n is obtained by

1 ► `SteinerLoop(` n, m `)` F

Our catalog numbers coincide with those of Colbourn and Rosa [4].

9.6 CC-loops

By results of Kunen [12], for every odd prime p there are precisely 3 nonassociative conjugacy closed loops of order p^2. Csörgő and Drápal [5] described these 3 loops by multiplicative formulas on $\mathbb{Z}_{p^2}$ and $\mathbb{Z}_p \times \mathbb{Z}_p$.

Case $m = 1$**:** Let k be the smallest positive integer relatively prime to p and such that k is a square modulo p (i.e., $k = 1$). Define multiplication on $\mathbb{Z}_{p^2}$ by $x \cdot y = x + y + kpx^2y$.

Case $m = 2$**:** Let k be the smallest positive integer relatively prime to p and such that k is not a square modulo p. Define multiplication on $\mathbb{Z}_{p^2}$ by $x \cdot y = x + y + kpx^2y$.

Case $m = 3$**:** Define multiplication on $\mathbb{Z}_p \times \mathbb{Z}_p$ by $(x, a)(y, b) = (x + y, a + b + x^2y)$.

Moreover, Wilson [22] constructed a nonassociative CC-loop of order $2p$ for every odd prime p, and Kunen [12] showed that there are no other nonassociative CC-loops of this order. Here is the construction:

Let N be an additive cyclic group of order $n > 2$, $N = \langle 1 \rangle$. Let G be the additive cyclic group of order 2. Define multiplication on $L = G \times N$ as follows:

$$\begin{array}{ll} (0,m)(0,n) = (0,m+n), & (0,m)(1,n) = (1,-m+n), \\ (1,m)(0,n) = (1,m+n), & (1,m)(1,n) = (0,1-m+n)\cdot \end{array}$$

The CC-loops described above can be obtained by

1► `CCLoop(` *n*`,` *m* `)` F

9.7 Small loops

The library named *small* contains all nonassociative loops of order 5 and 6. There are 5 and 107 such loops, respectively. The loops are obtained by

1► `SmallLoop(` *n*`,` *m* `)` F

9.8 Paige loops

Paige loops are nonassociative finite simple Moufang loops. By [13], there is precisely one Paige loop for every finite field GF(q).

The library named *Paige* contains the smallest nonassociative simple Moufang loop

1► `PaigeLoop(` *2* `)` F

9.9 Nilpotent loops

The library named *nilpotent* contains all nonassociative nilpotent loops of order less than 12, up to isomorphism. There are 2 nonassociative nilpotent loops of order 6, 134 of order 8, 8 of order 9 and 1043 of order 10. They are obtained as usual with

1► `NilpotentLoop(` *n*`,` *m* `)` F

See [6] for more on enumeration of nilpotent loops. For instance, there are 2623755 nilpotent loops of order 12, and 123794003928541545927226368 nilpotent loops of order 22.

9.10 Automorphic loops

The library named *automorphic* contains all nonassociative automorphic loops of order less that 16, up to isomorphism. There is 1 such loop of order 6, 7 of order 8, 3 of order 10, 2 of order 12, 5 of order 14, and 2 of order 15. They are obtained as usual with

1► `AutomorphicLoop(` *n*`,` *m* `)` F

9.11 Interesting loops

The library named *interesting* contains some loops that are illustrative for the theory of loops. At this point, the library contains a nonassociative loop of order 5, a nonassociative nilpotent loop of order 6, a nonMoufang left Bol loop of order 16, and the loop of sedenions of order 32 (sedenions generalize octonions).

The loops are obtained with

1 ► `InterestingLoop(` *n*, *m* `)` F

9.12 Libraries of loops up to isotopism

For the library *small* we also provide the corresponding library of loops up to isotopism.

In general, given a library named *lib*, the corresponding library up to isotopism is named *itp lib*, and the loops can be retrieved by the template function `ItpLibLoop( n, m )`. Thus we have

1 ► `ItpSmallLoop( n, m )` O

Here is an example:

```
gap> SmallLoop( 6, 14 );
<small loop 6/14>
gap> ItpSmallLoop( 6, 14 );
<small loop 6/42>
gap> LibraryLoop( "itp small", 6, 14 );
<small loop 6/42>
```

Note that loops up to isotopism form a subset of the corresponding library of loops up to isomorphism. For instance, the above example shows that the 14th small loop of order 6 up to isotopism is in fact the 42nd small loop of order 6 up to isomorphism.

Here is the list of all supported libraries up to isotopism and their extent, as displayed by LOOPS:

```
gap> DisplayLibraryInfo("itp small");
The library contains all nonassociative loops of order less than 7 up to
isotopism.
------
Extent of the library:
   1 loop of order 5
   20 loops of order 6
```

A Files

Below is a list of all relevant files forming the LOOPS package. Some technical files are not mentioned. You do not need any of this information unless you want to modify the package. All paths are relative to the `loops` folder.

```
../README.loops (installation and usage instructions)
init.g (declarations)
PackageInfo.g (loading info for GAP 4.4)
read.g (implementations)
data/cc.tbl (library of CC-loops)
data/code.tbl (library of code loops)
data/interesting.tbl (library of interesting loops)
data/itp_small.tbl (library of small loops up to isotopism)
data/leftbol.tbl (library of left Bol loops)
data/moufang.tbl (library of Moufang loops
data/paige.tbl (library of Paige loops)
data/small.tbl (library of small loops)
data/steiner.tbl (library of Steiner loops)
doc/manual.* (documentation files in all but html format)
gap/banner.g (banner of LOOPS)
gap/bol_core_methods.gd .gi (core methods for Bol loops)
gap/classes.gd .gi (properties of quasigroups and loops)
gap/core_methods.gd .gi (core methods for quasigroups and loops)
gap/elements.gd .gi (elements and basic arithmetic operations)
gap/examples.gd .gi (methods for libraries of loops)
gap/extensions.gd .gi (methods for extensions of loops)
gap/iso.gd .gi (methods for isomorphisms and isotopisms of loops)
gap/mlt_search.gd .gi (realizing groups as multiplication groups of loops)
gap/moufang_modifications.gd .gi (methods for Moufang modifications)
gap/moufang_triality.gd .gi (methods for triality of Moufang loops)
gap/quasigroups.gd .gi (representing, creating and displaying quasigroups)
gap/random.gd .gi (random quasigroups and loops)
htm/*.htm (documentation files in html format)
tst/bol.tst (test file for Bol loops)
tst/core_methods.tst (test file for core methods)
tst/iso.tst (test file for isomorphisms and automorphisms)
tst/lib.tst (test file for libraries of loops, except Moufang loops)
tst/nilpot.tst (test file for nilpotency and triality)
tst/testall.g (batch for all tets files)
```

B Filters built into the package

Many implications among properties of loops are built directly into LOOPS. A sizeable portion of these properties are of trivial character or are based on definitions (e.g., alternative loops = left alternative loops + right alternative loops). The remaining implications are theorems.

All filters of LOOPS are summarized below, using the GAP convention that the property on the left is implied by the property (properties) on the right.

```
( IsExtraLoop, IsAssociative and IsLoop )
( IsExtraLoop, IsCodeLoop )
( IsCCLoop, IsCodeLoop )
( HasTwosidedInverses, IsPowerAssociative )
( IsPowerAlternative, IsDiassociative )
( IsFlexible, IsDiassociative )
( HasAntiautomorphicInverseProperty, HasAutomorphicInverseProperty and IsCommutative )
( HasAutomorphicInverseProperty, HasAntiautomorphicInverseProperty and IsCommutative )
( HasLeftInverseProperty, HasInverseProperty )
( HasRightInverseProperty, HasInverseProperty )
( HasWeakInverseProperty, HasInverseProperty )
( HasAntiautomorphicInverseProperty, HasInverseProperty )
( HasTwosidedInverses, HasAntiautomorphicInverseProperty )
( HasInverseProperty, HasLeftInverseProperty and IsCommutative )
( HasInverseProperty, HasRightInverseProperty and IsCommutative )
( HasInverseProperty, HasLeftInverseProperty and HasRightInverseProperty )
( HasInverseProperty, HasLeftInverseProperty and HasWeakInverseProperty )
( HasInverseProperty, HasRightInverseProperty and HasWeakInverseProperty )
( HasInverseProperty, HasLeftInverseProperty and HasAntiautomorphicInverseProperty )
( HasInverseProperty, HasRightInverseProperty and HasAntiautomorphicInverseProperty )
( HasInverseProperty, HasWeakInverseProperty and HasAntiautomorphicInverseProperty )
( HasTwosidedInverses, HasLeftInverseProperty )
( HasTwosidedInverses, HasRightInverseProperty )
( HasTwosidedInverses, IsFlexible and IsLoop )
( IsMoufangLoop, IsExtraLoop )
( IsCLoop, IsExtraLoop )
( IsExtraLoop, IsMoufangLoop and IsLeftNuclearSquareLoop )
( IsExtraLoop, IsMoufangLoop and IsMiddleNuclearSquareLoop )
( IsExtraLoop, IsMoufangLoop and IsRightNuclearSquareLoop )
( IsLeftBolLoop, IsMoufangLoop )
( IsRightBolLoop, IsMoufangLoop )
( IsDiassociative, IsMoufangLoop )
( IsMoufangLoop, IsLeftBolLoop and IsRightBolLoop )
( IsLCLoop, IsCLoop )
( IsRCLoop, IsCLoop )
( IsDiassociative, IsCLoop and IsFlexible)
( IsCLoop, IsLCLoop and IsRCLoop )
( IsRightBolLoop, IsLeftBolLoop and IsCommutative )
( IsLeftPowerAlternative, IsLeftBolLoop )
( IsLeftBolLoop, IsRightBolLoop and IsCommutative )
( IsRightPowerAlternative, IsRightBolLoop )
( IsLeftPowerAlternative, IsLCLoop )
( IsLeftNuclearSquareLoop, IsLCLoop )
( IsMiddleNuclearSquareLoop, IsLCLoop )
( IsRCLoop, IsLCLoop and IsCommutative )
( IsRightPowerAlternative, IsRCLoop )
( IsRightNuclearSquareLoop, IsRCLoop )
( IsMiddleNuclearSquareLoop, IsRCLoop )
( IsLCLoop, IsRCLoop and IsCommutative )
( IsRightNuclearSquareLoop, IsLeftNuclearSquareLoop and IsCommutative )
( IsLeftNuclearSquareLoop, IsRightNuclearSquareLoop and IsCommutative )
( IsLeftNuclearSquareLoop, IsNuclearSquareLoop )
( IsRightNuclearSquareLoop, IsNuclearSquareLoop )
( IsMiddleNuclearSquareLoop, IsNuclearSquareLoop )
( IsNuclearSquareLoop, IsLeftNuclearSquareLoop
    and IsRightNuclearSquareLoop and IsMiddleNuclearSquareLoop )
( IsFlexible, IsCommutative )
( IsRightAlternative, IsLeftAlternative and IsCommutative )
( IsLeftAlternative, IsRightAlternative and IsCommutative )
( IsLeftAlternative, IsAlternative )
( IsRightAlternative, IsAlternative )
( IsAlternative, IsLeftAlternative and IsRightAlternative )
( IsLeftAlternative, IsLeftPowerAlternative )
( HasLeftInverseProperty, IsLeftPowerAlternative )
( IsPowerAssociative, IsLeftPowerAlternative )
( IsRightAlternative, IsRightPowerAlternative )
( HasRightInverseProperty, IsRightPowerAlternative )
( IsPowerAssociative, IsRightPowerAlternative )
( IsLeftPowerAlternative, IsPowerAlternative )
( IsRightPowerAlternative, IsPowerAlternative )
( IsAssociative, IsLCCLoop and IsCommutative )
( IsExtraLoop, IsLCCLoop and IsMoufangLoop )
( IsAssociative, IsRCCLoop and IsCommutative )
( IsExtraLoop, IsRCCLoop and IsMoufangLoop )
( IsLCCLoop, IsCCLoop )
( IsRCCLoop, IsCCLoop )
( IsCCLoop, IsLCCLoop and IsRCCLoop )
( IsOsbornLoop, IsMoufangLoop )
( IsOsbornLoop, IsCCLoop )
( HasAutomorphicInverseProperty, IsLeftBruckLoop )
( IsLeftBolLoop, IsLeftBruckLoop )
( IsRightBruckLoop, IsLeftBruckLoop and IsCommutative )
( IsLeftBruckLoop, IsLeftBolLoop and HasAutomorphicInverseProperty )
( HasAutomorphicInverseProperty, IsRightBruckLoop )
( IsRightBolLoop, IsRightBruckLoop )
( IsLeftBruckLoop, IsRightBruckLoop and IsCommutative )
( IsRightBruckLoop, IsRightBolLoop and HasAutomorphicInverseProperty )
( IsCommutative, IsSteinerLoop )
( IsCLoop, IsSteinerLoop )
( IsLeftAutomorphicLoop, IsAutomorphicLoop )
( IsRightAutomorphicLoop, IsAutomorphicLoop )
( IsMiddleAutomorphicLoop, IsAutomorphicLoop )
( IsMiddleAutomorphicLoop, IsCommutative )
( IsAutomorphicLoop, IsLeftAutomorphicLoop and IsCommutative )
( IsAutomorphicLoop, IsRightAutomorphicLoop and IsCommutative )
( IsLeftAutomorphicLoop, IsRightAutomorphicLoop and HasAntiautomorphicInverseProperty )
( IsRightAutomorphicLoop, IsLeftAutomorphicLoop and HasAntiautomorphicInverseProperty )
( IsFlexible, IsMiddleAutomorphicLoop )
( HasAntiautomorphicInverseProperty, IsFlexible and IsLeftAutomorphicLoop )
( HasAntiautomorphicInverseProperty, IsFlexible and IsRightAutomorphicLoop )
( IsMoufangLoop, IsAutomorphicLoop and IsLeftAlternative )
( IsMoufangLoop, IsAutomorphicLoop and IsRightAlternative )
( IsMoufangLoop, IsAutomorphicLoop and HasLeftInverseProperty )
( IsMoufangLoop, IsAutomorphicLoop and HasRightInverseProperty )
( IsMoufangLoop, IsAutomorphicLoop and HasWeakInverseProperty )
( IsLeftAutomorphicLoop, IsLeftBruckLoop )
( IsLeftAutomorphicLoop, IsLCCLoop )
( IsRightAutomorphicLoop, IsRightBruckLoop )
( IsRightAutomorphicLoop, IsRCCLoop )
( IsAutomorphicLoop, IsCommutative and IsMoufangLoop )
```

Bibliography

[1] R. Artzy. On automorphic-inverse properties in loops. *Proc. Amer. Math. Soc.*, 10:588–591, 1959.

[2] Richard Hubert Bruck. *A survey of binary systems*. Ergebnisse der Mathematik und ihrer Grenzgebiete. Neue Folge, Heft 20. Reihe: Gruppentheorie. Springer Verlag, Berlin, 1958.

[3] O. Chein, H. O. Pflugfelder, and J. D. H. Smith, editors. *Quasigroups and loops: theory and applications*, volume 8 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, 1990.

[4] Charles J. Colbourn and Alexander Rosa. *Triple systems*. Oxford Mathematical Monographs. The Clarendon Press Oxford University Press, New York, 1999.

[5] Piroska Csörgő and Aleš Drápal. Left conjugacy closed loops of nilpotency class two. *Results Math.*, 47(3-4):242–265, 2005.

[6] Daniel Daly and Petr Vojtěchovský. Enumeration of nilpotent loops via cohomology. *J. Algebra*, 322(11):4080–4098, 2009.

[7] Aleš Drápal and Petr Vojtěchovský. Moufang loops that share associator and three quarters of their multiplication tables. *Rocky Mountain J. Math.*, 36(2):425–455, 2006.

[8] Ferenc Fenyves. Extra loops. II. On loops with identities of Bol-Moufang type. *Publ. Math. Debrecen*, 16:187–192, 1969.

[9] Edgar G. Goodaire, Sean May, and Maitreyi Raman. *The Moufang loops of order less than 64*. Nova Science Publishers Inc., Commack, NY, 1999.

[10] Mark T. Jacobson and Peter Matthews. Generating uniformly distributed random Latin squares. *J. Combin. Des.*, 4(6):405–437, 1996.

[11] Michael K. Kinyon, Kenneth Kunen, and J. D. Phillips. Every diassociative A-loop is Moufang. *Proc. Amer. Math. Soc.*, 130(3):619–624, 2002.

[12] Kenneth Kunen. The structure of conjugacy closed loops. *Trans. Amer. Math. Soc.*, 352(6):2889–2911, 2000.

[13] Martin W. Liebeck. The classification of finite simple Moufang loops. *Math. Proc. Cambridge Philos. Soc.*, 102(1):33–47, 1987.

[14] G. Eric Moorhouse. Bol loops of small order. http://www.uwyo.edu/moorhouse/pub/bol/.

[15] Gábor P. Nagy. A class of simple proper Bol loops. *Manuscripta Math.*, 127(1):81–88, 2008.

[16] Gábor P. Nagy and Petr Vojtěchovský. Octonions, simple Moufang loops and triality. *Quasigroups Related Systems*, 10:65–94, 2003.

[17] Gábor P. Nagy and Petr Vojtěchovský. The Moufang loops of order 64 and 81. *J. Symbolic Comput.*, 42(9):871–883, 2007.

[18] Hala O. Pflugfelder. *Quasigroups and loops: introduction*, volume 7 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, 1990.

[19] J. D. Phillips and Petr Vojtěchovský. The varieties of loops of Bol-Moufang type. *Algebra Universalis*, 54(3):259–271, 2005.

[20] M. Slattery and A. Zenisek. Moufang loops of order 243. preprint.

[21] Petr Vojtěchovský. Toward the classification of Moufang loops of order 64. *European J. Combin.*, 27(3):444–460, 2006.

[22] Robert L. Wilson, Jr. Quasidirect products of quasigroups. *Comm. Algebra*, 3(9):835–850, 1975.