The LOOPS Package

Computing with quasigroups and loops in GAP

Version 3.3.0

Gábor P. Nagy
Petr Vojtěchovský

Gábor P. Nagy Email: nagyg@math.u-szeged.hu
Address: Department of Mathematics, University of Szeged

Petr Vojtěchovský Email: petr@math.du.edu
Address: Department of Mathematics, University of Denver

Copyright

Contents

Chapter 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.

LOOPS has been accepted as an official package of GAP in May 2015.

1.1 License

LOOPS is free software. You can redistribute it and/or modify it under the terms of the GNU General Public License version 3 as published by the Free Software Foundation.

1.2 Installation

Have GAP 4.7 or newer installed on your computer.

If you do not see the subfolder `pkg/loops` in the main directory of GAP then download the LOOPS package from the distribution website `http://www.math.du.edu/loops` and unpack the downloaded file into the `pkg` subfolder.

The package LOOPS can then be loaded to GAP anytime by calling `LoadPackage("loops");`.

If you wish to load LOOPS automatically while starting GAP, start GAP, execute the following commands,

```
gap> pref := UserPreference( "PackagesToLoad " );;
gap> Add( pref, "loops" );;
gap> SetUserPreference( "PackagesToLoad", pref );;
gap> WriteGapIniFile();;
```

quit GAP and restart it.

1.3 Documentation

The documentation has been prepared with the GAPDoc package and is therefore available in several formats: LaTeX, pdf, ps, html, and as an inline help in GAP. All these formats have been obtained directly from the master XML documentation file. Consequently, the different formats of the documentation differ only in their appearance, not in content.

The preferred format of the documentation is html with MathJax turned on.

All formats of the documentation can be found in the `doc` folder of LOOPS. You can also download the documentation from the LOOPS distribution website.

The inline 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`.

1.4 Test Files

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

1.5 Memory Management

Some libraries of loops are loaded only upon explicit demand and can occupy a lot of memory. For instance, the library or RCC loops occupies close to 100 MB of memory when fully loaded. The command `LOOPS_FreeMemory();` will attempt to free memory by unbinding on-demand library data loaded by LOOPS.

1.6 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.

1.7 Acknowledgment

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, Aleš Drápal, Steve Flammia, Kenneth W. Johnson, Michael K. Kinyon, Alexander Konovalov, Frank Lübeck and Jonathan D.H. Smith.

The library of Moufang loops of order 243 was generated from data provided by Michael C. Slattery and Ashley L. Zenisek. The library of right conjugacy closed loops of order less than 28 was generated from data provided by Katharina Artic. The library of commutative automorphic loops of order 27, 81 and 243 was obtained jointly with Izabella Stuhl.

Gábor P. Nagy was supported by OTKA grants F042959 and T043758, and Petr Vojtěchovský was supported by the 2006 and 2016 University of Denver PROF grants and the Simons Foundation Collaboration Grant 210176.

Chapter 2

Mathematical Background

We assume that you are familiar with the theory of quasigroups and loops, for instance with the textbook of Bruck [Bru58] or Pflugfelder [Pfl90]. Nevertheless, we did include definitions and results in this manual in order to unify terminology and improve legibility 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.

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.

Let G be a groupoid with neutral element 1. Then an element x^{-1} is called a *two-sided inverse* of x in G if $x\cdot x^{-1} = x^{-1}\cdot x = 1$.

Recall that groups are associative groupoids with an identity element and two-sided inverses. Groups can be reached in another way from groupoids, namely via 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., 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. Conversely, it is not hard to show that associative quasigroups are groups.

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 $y \mapsto y\cdot x$.

The binary operation $x\backslash y = L_x^{-1}(y)$ is called the *left division*, and $x/y = R_y^{-1}(x)$ is called the *right division*.

Although it is possible to compose two left (right) translations of a quasigroup, the resulting permutation is not necessarily a left (right) 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 $\mathrm{Mlt}_\lambda(Q) = \langle L_x | x \in Q\rangle$ of S_Q generated by all left translations is the *left multiplication group* of Q. Similarly, $\mathrm{Mlt}_\rho(Q) = \langle R_x | x \in Q\rangle$ is the *right multiplication group* of Q. The smallest group containing both $\mathrm{Mlt}_\lambda(Q)$ and $\mathrm{Mlt}_\rho(Q)$ is called the *multiplication group* of Q and is denoted by $\mathrm{Mlt}(Q)$.

For a loop Q, the *left inner mapping group* $\mathrm{Inn}_\lambda(Q)$ is the stabilizer of 1 in $\mathrm{Mlt}_\lambda(Q)$. The *right inner mapping group* $\mathrm{Inn}_\rho(Q)$ is defined dually. The *inner mapping group* $\mathrm{Inn}(Q)$ is the stabilizer of 1 in Q.

2.3 Subquasigroups and Subloops

A nonempty subset S of a quasigroup Q is a *subquasigroup* if it is closed under multiplication and the left and right divisions. In the finite case, it suffices for S to be closed under multiplication. *Subloops* are defined analogously when Q is a loop.

The *left nucleus* $\mathrm{Nuc}_\lambda(Q)$ of Q consists of all elements x of Q such that $x(yz) = (xy)z$ for every y, z in Q. The *middle nucleus* $\mathrm{Nuc}_\mu(Q)$ and the *right nucleus* $\mathrm{Nuc}_\rho(Q)$ are defined analogously. The *nucleus* $\mathrm{Nuc}(Q)$ is the intersection of the left, middle and right nuclei.

The *commutant* $C(Q)$ of Q consists of all elements x of Q that commute with all elements of Q. The *center* $Z(Q)$ of Q is the intersection of $\mathrm{Nuc}(Q)$ with $C(Q)$.

A subloop S of Q is *normal* in Q if $f(S) = S$ for every inner mapping f of Q.

2.4 Nilpotence and Solvability

For a loop Q define $Z_0(Q) = 1$ and let $Z_{i+1}(Q)$ be the preimage of the center of $Q/Z_i(Q)$ in Q. A loop Q is *nilpotent of class n* if n is the least nonnegative integer such that $Z_n(Q) = Q$. In such case $Z_0(Q) \le Z_1(Q) \le \ldots \le Z_n(Q)$ is the *upper central series*.

The *derived subloop* Q' of Q is the least normal subloop of Q such that Q/Q' is a commutative group. Define $Q^{(0)} = Q$ and let $Q^{(i+1)}$ be the derived subloop of $Q^{(i)}$. Then Q is *solvable of class n* if n is the least nonnegative integer such that $Q^{(n)} = 1$. In such a case $Q^{(0)} \ge Q^{(1)} \ge \cdots \ge Q^{(n)}$ is the *derived series* of Q.

2.5 Associators and Commutators

Let Q be a quasigroup and let x, y, z be elements of Q. Then the *commutator* of x, y is the unique element $[x,y]$ of Q such that $xy = [x,y](yx)$, and the *associator* of x, y, z is the unique element $[x,y,z]$ of Q such that $(xy)z = [x,y,z](x(yz))$.

The *associator subloop* $A(Q)$ of Q is the least normal subloop of Q such that $Q/A(Q)$ is a group.

It is not hard to see that $A(Q)$ is the least normal subloop of Q containing all commutators, and Q' is the least normal subloop of Q containing all commutators and associators.

2.6 Homomorphism 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.

An 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, then (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.

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 $(K,\cdot)$.

Chapter 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),
- algorithms for specific varieties of loops, 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 general philosophy so that users can anticipate the capabilities and behavior of LOOPS.

3.1 Representing Quasigroups

Since permutation representation in the usual sense is impossible for nonassociative structures, and since the theory of nonassociative presentations is not well understood, we resorted 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. See Section 4.1 for more details.)

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.

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 $1, \ldots, n$ (see Section 6.1).

- 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 the conversion `IntoLoop(Q)` even if the quasigroup *Q* does not posses a neutral element.

Details of all conversions in LOOPS can be found in Section 4.10.

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 translations (see Section 2.2), and we illustrate it with a few examples:

EXAMPLE: 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 $L_{xy} = L_xL_y$ for every x, y in Q. Note that since L_{xy}, L_x and 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: 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 Q is left Bol if and only if $L_xL_yL_x$ is a left translation for every x, y in Q.

EXAMPLE: This example shows that many properties of loops become purely group-theoretical once they are expressed in terms of translations. A loop is *simple* if it has no nontrivial congruences. It is possible to show 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: the `View` mode (default, short), and the `Print` mode (longer). Moreover, when the name of an object is set, the name 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" and is shown as `<Moufang loop 32/4>`.

A generic quasigroup of order n is displayed as `<quasigroup of order n>`. Similarly, a loop of order n appears as `<loop of order n>`.

The displayed information of a generic loop is enhanced if more information about the loop becomes available. For instance, when it is established that a loop of order 12 has the left alternative property, the loop will be shown as `<left alternative loop of order 12>` until a stronger property is obtained. Which property is diplayed is governed by the filters built into LOOPS (see Appendix B).

3.4.1 SetQuasigroupElmName and SetLoopElmName

▷ `SetQuasigroupElmName(`*Q*`, `*name*`)` (function)
▷ `SetLoopElmName(`*Q*`, `*name*`)` (function)

The above functions change the names of elements of a quasigroup (resp. loop) *Q* to *name*.

By default, elements of a quasigroup appear as `qi` and elements of a loop appear as `li` in both display modes, where `i` is a positive integer. The neutral element of a loop is always indexed by 1.

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

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

Example
```
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 ]
```

Chapter 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 unfortunate feature of multiplication tables in practice is that they are often not bordered, that is, 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 positive 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

4.2.1 IsQuasigroupTable and IsQuasigroupCayleyTable

▷ `IsQuasigroupTable(`*T*`)` (operation)

▷ `IsQuasigroupCayleyTable(`*T*`)` (operation)

Returns: `true` if *T* is a quasigroup table as defined above, else `false`.

4.2.2 IsLoopTable and IsLoopCayleyTable

▷ `IsLoopTable(`*T*`)` (operation)

▷ `IsLoopCayleyTable(`*T*`)` (operation)

Returns: `true` if *T* is a loop table as defined above, else `false`.

REMARK:The package GUAVA also contains operations dealing with latin squares. In particular, `IsLatinSquare` is declared in GUAVA.

4.3 Canonical and Normalized Cayley Tables

4.3.1 CanonicalCayleyTable

▷ `CanonicalCayleyTable(T)` (operation)

Returns: Canonical Cayley table constructed from Cayley table T by replacing entries x_i with i.

A Cayley table is said to be *canonical* if it is based on elements 1, ..., n. Although we do not assume that every quasigroup table is canonical, it is often desirable to present quasigroup tables in canonical way.

4.3.2 CanonicalCopy

▷ `CanonicalCopy(Q)` (operation)

Returns: A canonical copy of the quasigroup or loop Q.

This is a shorthand for `QuasigroupByCayleyTable(CanonicalCayleyTable(Q)` when Q is a declared quasigroup, and `LoopByCayleyTable(CanonicalCayleyTable(Q)` when Q is a loop.

4.3.3 NormalizedQuasigroupTable

▷ `NormalizedQuasigroupTable(T)` (operation)

Returns: A normalized version of the Cayley table T.

A given Cayley table T is normalized in three steps as follows: first, `CanonicalCayleyTable` is called to rename entries to 1, ..., n, then the columns of T are permuted so that the first row reads 1, ..., n, and finally the rows of T are permuted so that the first column reads 1, ..., n.

4.4 Creating Quasigroups and Loops From Cayley Tables

4.4.1 QuasigroupByCayleyTable and LoopByCayleyTable

▷ `QuasigroupByCayleyTable(T)` (operation)
▷ `LoopByCayleyTable(T)` (operation)

Returns: The quasigroup (resp. loop) with quasigroup table (resp. loop table) T.

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

Example
```
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 general method 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 *`filename`* and a string *`del`* as input (in essence, the characters of *`del`* will be ignored while reading the file):

- read the entire content of *`filename`* into a string *`s`*,
- replace all end-of-line characters in `s` by spaces,
- replace by spaces all characters of `s` that appear in *`del`*,
- split `s` into maximal substrings without spaces, called *chunks* here,
- 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 $1, \ldots, n$ 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 deletion string D produce multiplication table T.

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

$$\begin{array}{llll} 0 & 1 & 2 & 1 \\ 2 & 0 & 2 & \\ 0 & 1 & & \end{array} \quad + \quad \text{””} \quad \Longrightarrow \quad \begin{array}{ccc} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{array}$$

EXAMPLE: Chunks can be any strings.

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

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

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

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

$$\begin{array}{lll} x\& & y\& & z\backslash\backslash \\ y\& & z\& & x\backslash\backslash \\ z\& & x\& & y \end{array} \quad + \quad \text{”}\backslash\backslash\&\text{”} \quad \Longrightarrow \quad \begin{array}{ccc} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{array}$$

4.5.1 QuasigroupFromFile and LoopFromFile

▷ `QuasigroupFromFile(filename, del)` (operation)
▷ `LoopFromFile(filename, del)` (operation)

Returns: The quasigroup (resp. loop) whose multiplication table data is in file `filename`, ignoring the characters contained in the string `del`.

4.6 Creating Quasigroups and Loops From Sections

4.6.1 CayleyTableByPerms

▷ `CayleyTableByPerms(P)` (operation)

Returns: If `P` is a set of n permutations of an n-element set X, returns Cayley table C such that $C[i][j] = X[j]^{P[i]}$.

The cardinality of the underlying set is determined by the moved points of the first permutation in `P`, unless the first permutation is the identity permutation, in which case the second permutation is used.

In particular, if `P` is the left section of a quasigroup `Q`, `CayleyTableByPerms(Q)` returns the multiplication table of `Q`.

4.6.2 QuasigroupByLeftSection and LoopByLeftSection

▷ `QuasigroupByLeftSection(P)` (operation)
▷ `LoopByLeftSection(P)` (operation)

Returns: If `P` is a set of permutations corresponding to the left translations of a quasigroup (resp. loop), returns the corresponding quasigroup (resp. loop).

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

4.6.3 QuasigroupByRightSection and LoopByRightSection

▷ `QuasigroupByRightSection(P)` (operation)
▷ `LoopByRightSection(P)` (operation)

These are the dual operations to `QuasigroupByLeftSection` and `LoopByLeftSection`.

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 ] ]
```

4.7 Creating Quasigroups and Loops From Folders

Let G be a group, H a subgroup of G, and T a right transversal to H in G. Let $\tau : G \to T$ be defined by $x \in H\tau(x)$. Then the operation $\circ$ defined on the right cosets $Q = \{Ht | t \in T\}$ by $Hs \circ Ht = H\tau(st)$ 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 by letting $G = \mathrm{Mlt}_\rho(Q)$, $H = \mathrm{Inn}_\rho(Q)$ and $T = \{R_x | x \in Q\}$.)

We call the triple (G,H,T) a *right quasigroup (or loop) folder*.

4.7.1 QuasigroupByRightFolder and LoopByRightFolder

▷ `QuasigroupByRightFolder(G, H, T)` (operation)
▷ `LoopByRightFolder(G, H, T)` (operation)

Returns: The quasigroup (resp. loop) from the right folder (`G`, `H`, `T`).

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

Here is a simple example in which T is actually the right section of the resulting loop.

Example
```
gap> T := [ (), (1,2)(3,4,5), (1,3,5)(2,4), (1,4,3)(2,5), (1,5,4)(2,3) ];;
gap> G := Group( T );; H := Stabilizer( G, 1 );;
gap> LoopByRightFolder( G, H, T );
<loop of order 5>
```

4.8 Creating Quasigroups and Loops By Nuclear 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 $K \le N(Q)$.

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.

4.8.1 NuclearExtension

▷ `NuclearExtension(Q, K)` (operation)

Returns: The data necessary to construct `Q` as a nuclear extension of the subloop `K` by `Q/K`, namely $[K,F,\varphi,\theta]$ as above. Note that `K` must be a commutative subloop of the nucleus of `Q`.

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

4.8.2 LoopByExtension

▷ `LoopByExtension(K, F, f, t)` (operation)

Returns: The extension of an abelian group `K` by a loop `F`, using action `f` and cocycle `t`. The arguments must be formatted as the output of `NuclearExtension`.

Example
```
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
```

4.9 Random Quasigroups and Loops

An algorithm is said to select 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 [JM96], 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 $4(n-1)^3$ in the sense that no more than $4(n-1)^3$ properly chosen 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 smaller number of steps is probably sufficient.

4.9.1 RandomQuasigroup and RandomLoop

▷ `RandomQuasigroup(n[, iter])` (operation)
▷ `RandomLoop(n[, iter])` (operation)

Returns: A random quasigroup (resp. loop) of order `n` using the Jacobson-Matthews algorithm. If the optional argument `iter` is omitted, n^3 steps are used. Otherwise `iter` steps are used.

If `iter` is small, the Cayley table of the returned quasigroup (resp. loop) will be close to the canonical Cayley table of the cyclic group of order `n`.

4.9.2 RandomNilpotentLoop

▷ `RandomNilpotentLoop(lst)` (operation)

Returns: A random nilpotent loop as follows (see Section 6.9 for more information on nilpotency): `lst` must be a list of positive integers and/or finite abelian groups. If `lst=[a1]` and `a1` is an integer, a random abelian group of order `a1` is returned, else `a1` is an abelian group and `AsLoop(a1)` is returned. If `lst= [a1,...,am]`, a random central extension of `RandomNilpotentLoop([a1])` by `RandomNilpotentLoop([a2,...,am])` 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 is equal to

`Length(lst)`, else c is equal to `Length(lst)-1`.

4.10 Conversions

LOOPS contains methods that convert between magmas, quasigroups, loops and groups, provided such conversions are possible. Each of the conversion methods `IntoQuasigroup`, `IntoLoop` and `IntoGroup` returns `fail` if the requested conversion is not possible.

REMARK: Up to version 2.0.0 of LOOPS, we 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 a core method of GAP that returns an fp group if its argument is an associative loop.

4.10.1 IntoQuasigroup

▷ `IntoQuasigroup(M)` (operation)

Returns: If `M` is a declared magma that happens to be a quasigroup, the corresponding quasigroup is returned. If `M` is already declared as a quasigroup, `M` is returned.

4.10.2 PrincipalLoopIsotope

▷ `PrincipalLoopIsotope(M, f, g)` (operation)

Returns: An isomorphic copy of the principal isotope $(M,\circ)$ via the transposition $(1,f\cdot g)$. An isomorphic copy is returned rather than $(M,\circ)$ because in LOOPS all loops have to have neutral element labeled as 1.

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 2.6).

4.10.3 IntoLoop

▷ `IntoLoop(M)` (operation)

Returns: If `M` is a declared magma that happens to be a quasigroup (but not necessarily a loop!), a loop is returned as follows: If `M` is already declared as a loop, `M` is returned. Else, if `M` possesses a neutral element e and if f is the first element of `M`, then an isomorphic copy of `M` via the transposition (e,f) is returned. If `M` does not posses a neutral element, `PrincipalLoopIsotope(M, M.1, M.1)` is returned.

REMARK: One could obtain a loop from a declared magma `M` in yet another way, by normalizing the Cayley table of `M`. The three approaches can result in nonisomorphic loops in general.

4.10.4 IntoGroup

▷ `IntoGroup(M)` (operation)

Returns: If `M` is a declared magma that happens to be a group, the corresponding group is returned as follows: If `M` is already declared as a group, `M` is returned, else

`RightMultiplicationGroup(IntoLoop(M))` is returned, which is a permutation group isomorphic to M.

4.11 Products of Quasigroups and Loops

4.11.1 DirectProduct

▷ `DirectProduct(Q1, ..., Qn)` (operation)

Returns: If each `Qi` is either a declared quasigroup, declared loop or a declared group, the direct product of `Q1`, ..., `Qn` is returned. If every `Qi` is a declared group, a group is returned; if every `Qi` is a declared loop, a loop is returned; otherwise a quasigroup is returned.

4.12 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 $x * y = y \cdot x$.

4.12.1 Opposite, OppositeQuasigroup and OppositeLoop

▷ `Opposite(Q)` (attribute)

▷ `OppositeQuasigroup(Q)` (operation)

▷ `OppositeLoop(Q)` (operation)

Returns: The opposite of the quasigroup (resp. loop) `Q`. Note that if `OppositeQuasigroup(Q)` or `OppositeLoop(Q)` are called, then the returned quasigroup or loop is not stored as an attribute of `Q`.

Chapter 5

Basic Methods And Attributes

In this chapter we describe the basic core methods and attributes of the LOOPS package.

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.

5.1.1 Elements

▷ `Elements(Q)` (attribute)

Returns: The list of elements of a quasigroup `Q`.

See Section 3.4 for more information about element labels.

5.1.2 CayleyTable

▷ `CayleyTable(Q)` (attribute)

Returns: The Cayley table of a quasigroup `Q`.

See Section 4.1 for more information about quasigroup Cayley tables.

5.1.3 One

▷ `One(Q)` (attribute)

Returns: The identity element of a loop `Q`.

REMARK:If you want to know if a quasigroup `Q` has a neutral element, you can find out with the standard function for magmas `MultiplicativeNeutralElement(Q)`.

5.1.4 Size

▷ `Size(Q)` (attribute)

Returns: The size of a quasigroup `Q`.

5.1.5 Exponent

▷ `Exponent(Q)` (attribute)

Returns: The exponent of a power associative loop `Q`. (The method does not test if `Q` is power associative.)

When `Q` is a *power associative loop*, that is, the powers of elements are well-defined in `Q`, then the *exponent* of `Q` is the smallest positive integer divisible by the orders of all elements of `Q`.

5.2 Basic Arithmetic Operations

Each quasigroup element in GAP knows to which quasigroup it belongs. 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 parenthesize elements by default 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.

5.2.1 LeftDivision and RightDivision

▷ `LeftDivision(x, y)` (operation)
▷ `RightDivision(x, y)` (operation)

Returns: The left division `x\y` (resp. the right division `x/y`) of two elements `x`, `y` of the same quasigroup.

▷ `LeftDivision(S, x)` (operation)
▷ `LeftDivision(x, S)` (operation)
▷ `RightDivision(S, x)` (operation)
▷ `RightDivision(x, S)` (operation)

Returns: The list of elements obtained by performing the specified arithmetical operation elementwise using a list `S` of elements and an element `x`.

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

5.2.2 LeftDivisionCayleyTable and RightDivisionCayleyTable

▷ `LeftDivisionCayleyTable(Q)` (operation)
▷ `RightDivisionCayleyTable(Q)` (operation)

Returns: The Cayley table of the respective arithmetic operation of a quasigroup `Q`.

5.3 Powers and Inverses

Powers of elements are generally not well-defined in quasigroups. For magmas and a positive integral exponent, GAP calculates 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 about $\log_2(k)$ steps. For LOOPS, we have decided to keep this method, but the user should be aware that the method is sound only in power associative quasigroups.

Let x be an element of a loop Q with neutral element 1. Then the *left inverse* x^λ of x is the unique element of Q 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.

5.3.1 LeftInverse, RightInverse and Inverse

▷ `LeftInverse(x)` (operation)
▷ `RightInverse(x)` (operation)
▷ `Inverse(x)` (operation)

Returns: The left inverse, right inverse and inverse, respectively, of the quasigroup element `x`.

Example
```
gap> CayleyTable( Q );
[ [ 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> elms := Elements( Q );
gap> [ l1, l2, l3, l4, l5 ];
gap> [ LeftInverse( elms[3] ), RightInverse( elms[3] ), Inverse( elms[3] ) ];
[ l5, l4, fail ]
```

5.4 Associators and Commutators

See Section 2.5 for definitions of associators and commutators.

5.4.1 Associator

▷ `Associator(x, y, z)` (operation)

Returns: The associator of the elements `x`, `y`, `z` of the same quasigroup.

5.4.2 Commutator

▷ `Commutator(x, y)` (operation)

Returns: The commutator of the elements `x`, `y` of the same quasigroup.

5.5 Generators

5.5.1 GeneratorsOfQuasigroup and GeneratorsOfLoop

▷ `GeneratorsOfQuasigroup(Q)` (attribute)
▷ `GeneratorsOfLoop(Q)` (attribute)

Returns: A set of generators of a quasigroup (resp. loop) `Q`. (Both methods are synonyms of `GeneratorsOfMagma`.)

As usual in GAP, one can refer to the `i`th generator of a quasigroup `Q` by `Q.i`. Note that while it is often the case that `Q.i = Elements(Q)[i]`, it is not necessarily so.

5.5.2 GeneratorsSmallest

▷ `GeneratorsSmallest(Q)` (attribute)

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

5.5.3 SmallGeneratingSet

▷ `SmallGeneratingSet(Q)` (attribute)

Returns: A small generating set $\{q_0, \ldots, q_m\}$ of `Q` obtained as follows: q_0 is the least element for which $\langle q_0 \rangle$ is largest possible, $q_1$$ is the least element for which $\langle q_0, q_1$ is largest possible, and so on.

Chapter 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.

6.1.1 Parent

▷ `Parent(Q)` (attribute)

Returns: The parent quasigroup of the 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 set `Parent(Q)` equal to `Parent(H)`. Thus, in effect, `Parent(Q)` is the largest quasigroup from which `Q` has been created.

6.1.2 Position

▷ `Position(Q, x)` (operation)

Returns: The position of `x` among the elements of `Q`.

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)`.

While referring to elements of `Q` by their positions, the user should understand whether the positions are meant among the elements of `Q`, or among the elements of the parent `P` of `Q`. Since it requires no calculation to obtain `x![1]`, we always use the position of an element in its parent quasigroup in LOOPS. 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 $1, \ldots, n$ for some n.

6.1.3 PosInParent

▷ `PosInParent(S)` (operation)

Returns: When `S` is a list of quasigroup elements (not necessarily from the same quasigroup), returns the corresponding 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])`.

Quasigroups with the same parent can be compared as follows. 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` (see Section 5.5). Then we define $A < B$ if and only if $G_A < G_B$ lexicographically.

6.2 Subquasigroups and Subloops

6.2.1 Subquasigroup

▷ `Subquasigroup(Q, S)` (operation)

Returns: When `S` is a subset of elements or indices of a quasigroup (resp. loop) `Q`, returns the smallest subquasigroup (resp. subloop) of `Q` containing `S`.

We allow `S` to be a list of elements of `Q`, or a list of integers representing the positions of the respective elements in the parent quasigroup (resp. loop) of `Q`.

If `S` is empty, `Subquasigroup(Q,S)` returns the empty set if `Q` is a quasigroup, and it returns the one-element subloop of `Q` if `Q` is a loop.

REMARK: The empty set is sometimes considered to be a subquasigroup of `Q` (although not in LOOPS). The above convention is useful for handling certain situations, for instance when the user calls `Center(Q)` for a quasigroup `Q` with empty center.

6.2.2 Subloop

▷ `Subloop(Q, S)` (operation)

This is an analog of `Subquasigroup(Q,S)` that can be used only when `Q` is a loop. Since there is no difference in the outcome while calling `Subquasigroup(Q,S)` or `Subloop(Q,S)` when `Q` is a loop, it is safe to always call `Subquasigroup(Q,S)`, whether `Q` is a loop or not.

6.2.3 IsSubquasigroup and IsSubloop

▷ `IsSubquasigroup(Q, S)` (operation)
▷ `IsSubloop(Q, S)` (operation)

Returns: `true` if `S` is a subquasigroup (resp. subloop) of a quasigroup (resp. loop) `Q`, `false` otherwise. In other words, returns `true` if `S` and `Q` are quasigroups (resp. loops) with the same parent and `S` is a subset of `Q`.

6.2.4 AllSubquasigroups

▷ `AllSubquasigroups(Q)` (operation)

Returns: A list of all subquasigroups of a loop `Q`.

6.2.5 AllSubloops

▷ `AllSubloops(Q)` (operation)

Returns: A list of all subloops of a loop `Q`.

6.2.6 RightCosets

▷ `RightCosets(Q, S)` (function)

Returns: If `S` is a subloop of `Q`, returns a list of all right cosets of `S` in `Q`.

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

6.2.7 RightTransversal

▷ `RightTransversal(Q, S)` (operation)

Returns: A right transversal of a subloop `S` in a loop `Q`. The transversal consists of the list of first elements from the right cosets obtained by `RightCosets(Q,S)`.

When `S` is a subloop of `Q`, the right transversal of `S` with respect to `Q` is a subset of `Q` containing one element from each right coset of `S` in `Q`.

6.3 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., they are bijections of some interval 1, …, n. Moreover, following our convention, the numerical entries of the permutations point to the positions among elements of the parent of Q, not among elements of Q.

6.3.1 LeftTranslation and RightTranslation

▷ `LeftTranslation(Q, x)` (operation)

▷ `RightTranslation(Q, x)` (operation)

Returns: If `x` is an element of a quasigroup `Q`, returns the left translation (resp. right translation) by `x` in `Q`.

6.3.2 LeftSection and RightSection

▷ `LeftSection(Q)` (operation)

▷ `RightSection(Q)` (operation)

Returns: The left section (resp. right section) of a quasigroup `Q`.

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 a 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.

Example
```
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) ]
```

6.4 Multiplication Groups

6.4.1 LeftMutliplicationGroup, RightMultiplicationGroup and MultiplicationGroup

▷ `LeftMultiplicationGroup(Q)` (attribute)
▷ `RightMultiplicationGroup(Q)` (attribute)
▷ `MultiplicationGroup(Q)` (attribute)

Returns: The left multiplication group, right multiplication group, resp. multiplication group of a quasigroup `Q`.

6.4.2 RelativeLeftMultiplicationGroup, RelativeRightMultiplicationGroup and RelativeMultiplicationGroup

▷ `RelativeLeftMultiplicationGroup(Q, S)` (operation)
▷ `RelativeRightMultiplicationGroup(Q, S)` (operation)
▷ `RelativeMultiplicationGroup(Q, S)` (operation)

Returns: The relative left multiplication group, the relative right multiplication group, resp. the relative multiplication group of a quasigroup `Q` with respect to a subquasigroup `S` of `Q`.

Let S be a subquasigroup of a quasigroup Q. Then the *relative left multiplication group* of Q 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 the *relative multiplication group* are defined analogously.

6.5 Inner Mapping Groups

By a result of Bruck, the left inner mapping group of a loop 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*

$R(x,y) = R_{xy}^{-1} R_y R_x$.

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

6.5.1 LeftInnerMapping, RightInnerMapping, MiddleInnerMapping

▷ `LeftInnerMapping(Q, x, y)` (operation)
▷ `RightInnerMapping(Q, x, y)` (operation)
▷ `MiddleInnerMapping(Q, x)` (operation)

Returns: The left inner mapping $L(x,y)$, the right inner mapping $R(x,y)$, resp. the middle inner mapping $T(x)$ of a loop `Q`.

6.5.2 LeftInnerMappingGroup, RightInnerMappingGroup, MiddleInnerMapping-Group

▷ `LeftInnerMappingGroup(Q)` (attribute)
▷ `RightInnerMappingGroup(Q)` (attribute)
▷ `MiddleInnerMappingGroup(Q)` (attribute)

Returns: The left inner mapping group, right inner mapping group, resp. middle inner mapping group of a loop `Q`.

6.5.3 InnerMappingGroup

▷ `InnerMappingGroup(Q)` (attribute)

Returns: The inner mapping group of a loop `Q`.

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

Example
```
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
```

6.6 Nuclei, Commutant, Center, and Associator Subloop

See Section 2.3 for the relevant definitions.

6.6.1 LeftNucles, MiddleNucleus, and RightNucleus

▷ `LeftNucleus(Q)` (attribute)
▷ `MiddleNucleus(Q)` (attribute)
▷ `RightNucleus(Q)` (attribute)

Returns: The left nucleus, middle nucleus, resp. right nucleus of a quasigroup `Q`.

6.6.2 Nuc, NucleusOfQuasigroup and NucleusOfLoop

▷ `Nuc(Q)` (attribute)
▷ `NucleusOfQuasigroup(Q)` (attribute)
▷ `NucleusOfLoop(Q)` (attribute)

Returns: These synonymous attributes return the nucleus of a quasigroup `Q`.

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.

REMARK: The name `Nucleus` is a global function of GAP with two variables. We have therefore used `Nuc` rather than `Nucleus` for the nucleus. This abbreviation is sometimes used in the literature, too.

6.6.3 Commutant

▷ `Commutant(Q)` (attribute)

Returns: The commutant of a quasigroup `Q`.

6.6.4 Center

▷ `Center(Q)` (attribute)

Returns: The center of a quasigroup `Q`.

If `Q` is a loop, the center of `Q` is a subgroup of `Q` and it is returned as an associative loop.

6.6.5 AssociatorSubloop

▷ `AssociatorSubloop(Q)` (attribute)

Returns: The associator subloop of a loop `Q`.

We calculate the associator subloop of `Q` as the smallest normal subloop of `Q` containing all elements $x\backslash\alpha(x)$, where x is an element of `Q` and α is a left inner mapping of `Q`.

6.7 Normal Subloops and Simple Loops

6.7.1 IsNormal

▷ `IsNormal(Q, S)` (operation)

Returns: `true` if `S` is a normal subloop of a loop `Q`.

A subloop S of a loop Q is *normal* if it is invariant under all inner mappings of Q.

6.7.2 NormalClosure

▷ `NormalClosure(Q, S)` (operation)

Returns: The normal closure of a subset `S` of a loop `Q`.

For a subset S of a loop Q, the *normal closure* of S in Q is the smallest normal subloop of Q containing S.

6.7.3 IsSimple

▷ `IsSimple(Q)` (operation)

Returns: `true` if `Q` is a simple loop.

A loop Q is *simple* if $\{1\}$ and Q are the only normal subloops of Q.

6.8 Factor Loops

6.8.1 FactorLoop

▷ `FactorLoop(Q, S)` (operation)

Returns: When `S` is a normal subloop of a loop `Q`, returns the factor loop `Q/S`.

6.8.2 NaturalHomomorphismByNormalSubloop

▷ `NaturalHomomorphismByNormalSubloop(Q, S)` (operation)

Returns: When `S` is a normal subloop of a loop `Q`, returns the natural projection from `Q` onto `Q/S`.

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 )
```

6.9 Nilpotency and Central Series

See Section 2.4 for the relevant definitions.

6.9.1 IsNilpotent

▷ `IsNilpotent(Q)` (property)

Returns: `true` if `Q` is a nilpotent loop.

6.9.2 NilpotencyClassOfLoop

▷ `NilpotencyClassOfLoop(Q)` (attribute)

Returns: The nilpotency class of a loop `Q` if `Q` is nilpotent, `fail` otherwise.

6.9.3 IsStronglyNilpotent

▷ `IsStronglyNilpotent(Q)` (property)

Returns: `true` if `Q` is a strongly nilpotent loop.

A loop Q is said to be *strongly nilpotent* if its multiplication group is nilpotent.

6.9.4 UpperCentralSeries

▷ `UpperCentralSeries(Q)` (attribute)

Returns: When `Q` is a nilpotent loop, returns the upper central series of `Q`, else returns `fail`.

6.9.5 LowerCentralSeries

▷ `LowerCentralSeries(Q)` (attribute)

Returns: When `Q` is a nilpotent loop, returns the lower central series of `Q`, else returns `fail`.

The *lower central series* for loops is defined analogously to groups.

6.10 Solvability, Derived Series and Frattini Subloop

See Section 2.4 for definitions of solvability an derived subloop.

6.10.1 IsSolvable

▷ `IsSolvable(Q)` (property)

Returns: `true` if `Q` is a solvable loop.

6.10.2 DerivedSubloop

▷ `DerivedSubloop(Q)` (attribute)

Returns: The derived subloop of a loop `Q`.

6.10.3 DerivedLength

▷ `DerivedLength(Q)` (attribute)

Returns: If `Q` is solvable, returns the derived length of `Q`, else returns `fail`.

6.10.4 FrattiniSubloop and FrattinifactorSize

▷ `FrattiniSubloop(Q)` (attribute)

Returns: The Frattini subloop of `Q`. The method is implemented only for strongly nilpotent loops.

Frattini subloop of a loop Q is the intersection of maximal subloops of Q.

6.10.5 FrattinifactorSize

▷ `FrattinifactorSize(Q)` (attribute)

6.11 Isomorphisms and Automorphisms

6.11.1 IsomorphismQuasigroups

▷ `IsomorphismQuasigroups(Q, L)` (operation)

Returns: An isomorphism from a quasigroup `Q` to a quasigroup `L` if the quasigroups are isomorphic, `fail` otherwise.

If an isomorphism exists, it is returned as a permutation f of $1,\dots,|Q|$, where $i^f = j$ means that the ith element of `Q` is mapped onto the jth element of `L`. Note that this convention is used even if the underlying sets of `Q`, `L` are not indexed by consecutive integers.

6.11.2 IsomorphismLoops

▷ `IsomorphismLoops(Q, L)` (operation)

Returns: An isomorphism from a loop `Q` to a loop `L` if the loops are isomorphic, `fail` otherwise, with the same convention as in `IsomorphismQuasigroups`.

6.11.3 QuasigroupsUpToIsomorphism

▷ `QuasigroupsUpToIsomorphism(ls)` (operation)

Returns: Given a list `ls` of quasigroups, returns a sublist of `ls` consisting of representatives of isomorphism classes of quasigroups from `ls`.

6.11.4 LoopsUpToIsomorphism

▷ `LoopsUpToIsomorphism(ls)` (operation)

Returns: Given a list `ls` of loops, returns a sublist of `ls` consisting of representatives of isomorphism classes of loops from `ls`.

6.11.5 AutomorphismGroup

▷ `AutomorphismGroup(Q)` (attribute)

Returns: The automorphism group of a loop or quasigroups `Q`, with the same convention on permutations as in `IsomorphismQuasigroups`.

REMARK: Since two isomorphisms differ by an automorphism, all isomorphisms from `Q` to `L` can be obtained by a combination of `IsomorphismLoops(Q,L)` (or `IsomorphismQuasigroups(Q,L)`) and `AutomorphismGroup(L)`.

While dealing with Cayley tables, it is often useful to rename or reorder the elements of the underlying quasigroup without changing the isomorphism type of the quasigroups. LOOPS contains several functions for this purpose.

6.11.6 IsomorphicCopyByPerm

▷ `IsomorphicCopyByPerm(Q, f)` (operation)

Returns: When `Q` is a quasigroup and `f` is a permutation of $1,\dots,|Q|$, returns a quasigroup defined on the same set as `Q` with multiplication $*$ defined by $x*y = f(f^{-1}(x)f^{-1}(y))$. When `Q` is a declared loop, a loop is returned. Consequently, when `Q` is a declared loop and $f(1) = k \neq 1$, then `f` is first replaced with $f\circ(1,k)$, to make sure that the resulting Cayley table is normalized.

6.11.7 IsomorphicCopyByNormalSubloop

▷ `IsomorphicCopyByNormalSubloop(Q, S)` (operation)

Returns: When `S` is a normal subloop of a loop `Q`, returns an isomorphic copy of `Q` 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.

In order to speed up the search for isomorphisms and automorphisms, we first calculate some loop invariants preserved under isomorphisms, and then we use these invariants to partition the loop into blocks of elements preserved under isomorphisms. The following two operations are used in the search.

6.11.8 Discriminator

▷ `Discriminator(Q)` (operation)

Returns: A data structure with isomorphism invariants of a loop `Q`.

See [Voj06] or the file `iso.gi` for more details. The format of the discriminator has been changed from version 3.2.0 up to accommodate isomorphism searches for quasigroups.

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

6.11.9 AreEqualDiscriminators

▷ `AreEqualDiscriminators(D1, D2)` (operation)

Returns: `true` if `D1`, `D2` are equal discriminators for the purposes of isomorphism searches.

6.12 Isotopisms

At the moment, LOOPS contains only slow methods for testing if two loops are isotopic. 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.

6.12.1 IsotopismLoops

▷ `IsotopismLoops(K, L)` (operation)

Returns: `fail` if `K`, `L` are not isotopic loops, else it returns an isotopism as a triple of bijections on $1, \dots, |K|$.

6.12.2 LoopsUpToIsotopism

▷ `LoopsUpToIsotopism(ls)` (operation)

Returns: Given a list `ls` of loops, returns a sublist of `ls` consisting of representatives of isotopism classes of loops from `ls`.

Chapter 7

Testing Properties of Quasigroups and Loops

Although loops are quasigroups, it is often the case in the literature that a property of the same name 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

7.1.1 IsAssociative

▷ `IsAssociative(Q)` (property)

Returns: `true` if *Q* is an associative quasigroup.

7.1.2 IsCommutative

▷ `IsCommutative(Q)` (property)

Returns: `true` if *Q* is a commutative quasigroup.

7.1.3 IsPowerAssociative

▷ `IsPowerAssociative(Q)` (property)

Returns: `true` if *Q* is a power associative quasigroup.

A quasigroup Q is said to be *power associative* if every element of Q generates an associative quasigroup, that is, a group.

7.1.4 IsDiassociative

▷ `IsDiassociative(Q)` (property)

Returns: `true` if *Q* is a diassociative quasigroup.

A quasigroup Q is said to be *diassociative* if any two elements of Q generate an associative quasigroup, that is, a group. Note that a diassociative quasigroup is necessarily a loop, but it need not be so declared in LOOPS.

7.2 Inverse Propeties

For an element x of a loop Q, the *left inverse* of x is the element x^λ of Q such that $x^\lambda \cdot x = 1$, while the *right inverse* of x is the element x^ρ of Q such that $x \cdot x^\rho = 1$.

7.2.1 HasLeftInverseProperty, HasRightInverseProperty and HasInverseProperty

▷ `HasLeftInverseProperty(Q)` (property)
▷ `HasRightInverseProperty(Q)` (property)
▷ `HasInverseProperty(Q)` (property)

Returns: `true` if a loop `Q` has the left inverse property, right inverse property, resp. inverse property.

A loop Q has the *left inverse property* if $x^\lambda(xy) = y$ for every x, y in Q. Dually, Q has the *right inverse property* if $(yx)x^\rho = y$ for every x, y in Q. If Q has both the left inverse property and the right inverse property, it has the *inverse property*.

7.2.2 HasTwosidedInverses

▷ `HasTwosidedInverses(Q)` (property)

Returns: `true` if a loop `Q` has two-sided inverses.

A loop Q is said to have *two-sided inverses* if $x^\lambda = x^\rho$ for every x in Q.

7.2.3 HasWeakInverseProperty

▷ `HasWeakInverseProperty(Q)` (property)

Returns: `true` if a loop `Q` has the weak inverse property.

A loop Q has the *weak inverse property* if $(xy)^\lambda x = y^\lambda$ (equivalently, $x(yx)^\rho = y^\rho$) holds for every x, y in Q.

7.2.4 HasAutomorphicInverseProperty

▷ `HasAutomorphicInverseProperty(Q)` (property)

Returns: `true` if a loop `Q` has the automorphic inverse property.

According to [Art59], a loop Q has the *automorphic inverse property* if $(xy)^\lambda = x^\lambda y^\lambda$, or, equivalently, $(xy)^\rho = x^\rho y^\rho$ holds for every x, y in Q.

7.2.5 HasAntiautomorphicInverseProperty

▷ `HasAntiautomorphicInverseProperty(Q)` (property)

Returns: `true` if a loop `Q` has the antiautomorphic inverse property.

A loop Q has the *antiautomorphic inverse property* if $(xy)^\lambda = y^\lambda x^\lambda$, or, equivalently, $(xy)^\rho = y^\rho x^\rho$ holds for every x, y in Q.

See Appendix B for implications implemented in LOOPS among various inverse properties.

7.3 Some Properties of Quasigroups

7.3.1 IsSemisymmetric

▷ `IsSemisymmetric(Q)` (property)

Returns: `true` if `Q` is a semisymmetric quasigroup.

A quasigroup Q is *semisymmetric* if $(xy)x = y$, or, equivalently $x(yx) = y$ holds for every x, y in Q.

7.3.2 IsTotallySymmetric

▷ `IsTotallySymmetric(Q)` (property)

Returns: `true` if `Q` is a totally symmetric quasigroup.

A commutative semisymmetric quasigroup is called *totally symmetric*. Totally symmetric quasigroups are precisely the quasigroups satisfying $xy = x\backslash y = x/y$.

7.3.3 IsIdempotent

▷ `IsIdempotent(Q)` (property)

Returns: `true` if `Q` is an idempotent quasigroup.

A quasigroup is *idempotent* if it satisfies $x^2 = x$.

7.3.4 IsSteinerQuasigroup

▷ `IsSteinerQuasigroup(Q)` (property)

Returns: `true` if `Q` is a Steiner quasigroup.

A totally symmetric idempotent quasigroup is called a *Steiner quasigroup*.

7.3.5 IsUnipotent

A quasigroup Q is *unipotent* if it satisfies $x^2 = y^2$ for every x, y in Q.▷ `IsUnipotent(Q)` (property)

Returns: `true` if `Q` is a unipotent quasigroup.

7.3.6 IsLeftDistributive, IsRightDistributive, IsDistributive

▷ `IsLeftDistributive(Q)` (property)
▷ `IsRightDistributive(Q)` (property)
▷ `IsDistributive(Q)` (property)

Returns: `true` if `Q` is a left distributive quasigroup, resp. a right distributive quasigroup, resp. a distributive quasigroup.

A quasigroup is *left distributive* if it satisfies $x(yz) = (xy)(xz)$, *right distributive* if it satisfies $(xy)z = (xz)(yz)$, and *distributive* if it is both left distributive and right distributive.

REMARK: In order to be compatible with GAPs terminology, we also support the synonyms `IsLDistributive` and `IsRDistributive` of `IsLeftDistributive` and `IsRightDistributive`, respectively.

7.3.7 IsEntropic and IsMedial

▷ `IsEntropic(Q)` (property)
▷ `IsMedial(Q)` (property)

Returns: `true` if `Q` is an entropic (aka medial) quasigroup.

A quasigroup is *entropic* or *medial* if it satisfies the identity $(xy)(uv) = (xu)(yv)$.

7.4 Loops of Bol Moufang Type

Following [Fen69] and [PV05], 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, and the variables occur in the same order on both sides.

It is proved in [PV05] 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 nuclear square and middle nuclear square,
- *RC loops* defined by $x((yz)z) = (xy)(zz)$, always right alternative, right nuclear square and middle nuclear square,
- *Moufang loops* defined by $(xy)(zx) = (x(yz))x$, always flexible, left Bol 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 of loops of Bol-Moufang type can be defined by one of several equivalent identities of Bol-Moufang type.

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

7.4.1 IsExtraLoop

▷ `IsExtraLoop(Q)` (property)

Returns: `true` if *Q* is an extra loop.

7.4.2 IsMoufangLoop

▷ `IsMoufangLoop(Q)` (property)

Returns: `true` if *Q* is a Moufang loop.

7.4.3 IsCLoop

▷ `IsCLoop(Q)` (property)

Returns: `true` if *Q* is a C loop.

7.4.4 IsLeftBolLoop

▷ `IsLeftBolLoop(Q)` (property)

Returns: `true` if *Q* is a left Bol loop.

7.4.5 IsRightBolLoop

▷ `IsRightBolLoop(Q)` (property)

Returns: `true` if *Q* is a right Bol loop.

7.4.6 IsLCLoop

▷ `IsLCLoop(Q)` (property)

Returns: `true` if *Q* is an LC loop.

7.4.7 IsRCLoop

▷ `IsRCLoop(Q)` (property)

Returns: `true` if *Q* is an RC loop.

7.4.8 IsLeftNuclearSquareLoop

▷ `IsLeftNuclearSquareLoop(Q)` (property)

Returns: `true` if *Q* is a left nuclear square loop.

7.4.9 IsMiddleNuclearSquareLoop

▷ `IsMiddleNuclearSquareLoop(Q)` (property)

Returns: `true` if *Q* is a middle nuclear square loop.

7.4.10 IsRightNuclearSquareLoop

▷ `IsRightNuclearSquareLoop(Q)` (property)

Returns: `true` if *Q* is a right nuclear square loop.

7.4.11 IsNuclearSquareLoop

▷ IsNuclearSquareLoop(*Q*) (property)

Returns: `true` if *Q* is a nuclear square loop.

7.4.12 IsFlexible

▷ IsFlexible(*Q*) (property)

Returns: `true` if *Q* is a flexible quasigroup.

7.4.13 IsLeftAlternative

▷ IsLeftAlternative(*Q*) (property)

Returns: `true` if *Q* is a left alternative quasigroup.

7.4.14 IsRightAlternative

▷ IsRightAlternative(*Q*) (property)

Returns: `true` if *Q* is a right alternative quasigroup.

7.4.15 IsAlternative

▷ IsAlternative(*Q*) (property)

Returns: `true` if *Q* is an alternative quasigroup.

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

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

Example
```
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 except for the situations explicitly noted above.

7.5 Power Alternative Loops

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

7.5.1 IsLeftPowerAlternative, IsRightPowerAlternative and IsPowerAlternative

▷ `IsLeftPowerAlternative(Q)` (property)
▷ `IsRightPowerAlternative(Q)` (property)
▷ `IsPowerAlternative(Q)` (property)

Returns: `true` if `Q` is a left power alternative loop, resp. a right power alternative loop, resp. a power alternative loop.

7.6 Conjugacy Closed Loops and Related Properties

A loop Q is *left conjugacy closed* if the set of left translations of Q is closed under conjugation (by itself). Similarly, a loop Q is *right conjugacy closed* if the set of right translations of Q is closed under conjugation. A loop is *conjugacy closed* if it is both left conjugacy closed and right conjugacy closed. It is common to refer to these loops as LCC, RCC, and CC loops, respectively.

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

7.6.1 IsLCCLoop

▷ `IsLCCLoop(Q)` (property)
▷ `IsLeftConjugacyClosedLoop(Q)` (property)

Returns: `true` if `Q` is a left conjugacy closed loop.

7.6.2 IsRCCLoop

▷ `IsRCCLoop(Q)` (property)
▷ `IsRightConjugacyClosedLoop(Q)` (property)

Returns: `true` if `Q` is a right conjugacy closed loop.

7.6.3 IsCCLoop

▷ `IsCCLoop(Q)` (property)
▷ `IsConjugacyClosedLoop(Q)` (property)

Returns: `true` if `Q` is a conjugacy closed loop.

7.6.4 IsOsbornLoop

▷ `IsOsbornLoop(Q)` (property)

Returns: `true` if `Q` is an Osborn loop.

A loop is *Osborn* if it satisfies $x(yz\cdot x) = (x^{\lambda}\backslash y)(zx)$. Both Moufang loops and CC loops are Osborn.

7.7 Automorphic Loops

A loop Q whose all left (resp. middle, resp. right) inner mappings are automorphisms of Q is known as a *left automorphic loop* (resp. *middle automorphic loop*, resp. *right automorphic loop*).

A loop Q is an *automorphic loop* if all inner mappings of Q are automorphisms of Q.

Automorphic loops are also known as *A loops*, and similar terminology exists for left, right and middle automorphic loops.

The following results hold for automorphic loops:

- automorphic loops are power associative [BP56]
- in an automorphic loop Q we have $\mathrm{Nuc}(Q) = \mathrm{Nuc}_{\lambda}(Q) = \mathrm{Nuc}_{\rho}(Q) \leq \mathrm{Nuc}_{\mu}(Q)$ and all nuclei are normal [BP56]
- a loop that is left automorphic and right automorphic satisfies the anti-automorphic inverse property and is automorphic [JKNV11]
- diassociative automorphic loops are Moufang [KKP02]
- automorphic loops of odd order are solvable [KKPV16]
- finite commutative automorphic loops are solvable [GKN14]
- commutative automorphic loops of order p, $2p$, $4p$, p^2, $2p^2$, $4p^2$ (p an odd prime) are abelian groups [Voj15]
- commutative automorphic loops of odd prime power order are centrally nilpotent [JKV12]
- for any prime p, there are 7 commutative automorphic loops of order p^3 up to isomorphism [DBGV12]

See the built-in filters and the survey [Voj15] for additional properties of automorphic loops.

7.7.1 IsLeftAutomorphicLoop

▷ `IsLeftAutomorphicLoop(Q)` (property)
▷ `IsLeftALoop(Q)` (property)

Returns: `true` if `Q` is a left automorphic loop.

7.7.2 IsMiddleAutomorphicLoop

▷ `IsMiddleAutomorphicLoop(Q)` (property)
▷ `IsMiddleALoop(Q)` (property)

Returns: `true` if `Q` is a middle automorphic loop.

7.7.3 IsRightAutomorphicLoop

▷ `IsRightAutomorphicLoop(Q)` (property)
▷ `IsRightALoop(Q)` (property)

Returns: `true` if *Q* is a right automorphic loop.

7.7.4 IsAutomorphicLoop

▷ `IsAutomorphicLoop(Q)` (property)
▷ `IsALoop(Q)` (property)

Returns: `true` if *Q* is an automorphic loop.

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

7.8 Additonal Varieties of Loops

7.8.1 IsCodeLoop

▷ `IsCodeLoop(Q)` (property)

Returns: `true` if *Q* is a code loop.

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

7.8.2 IsSteinerLoop

▷ `IsSteinerLoop(Q)` (property)

Returns: `true` if *Q* is a Steiner loop.

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

7.8.3 IsLeftBruckLoop and IsLeftKLoop

▷ `IsLeftBruckLoop(Q)` (property)
▷ `IsLeftKLoop(Q)` (property)

Returns: `true` if *Q* is a left Bruck loop (aka left K loop).

A left Bol loop with the automorphic inverse property is known as a *left Bruck loop* or a *left K loop*.

7.8.4 IsRightBruckLoop and IsRightKLoop

▷ `IsRightBruckLoop(Q)` (property)
▷ `IsRightKLoop(Q)` (property)

Returns: `true` if *Q* is a right Bruck loop (aka right K loop).

A right Bol loop with the automorphic inverse property is known as a *right Bruck loop* or a *right K loop*.

Chapter 8

Specific Methods

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

8.1 Core Methods for Bol Loops

8.1.1 AssociatedLeftBruckLoop and AssociatedRightBruckLoop

▷ `AssociatedLeftBruckLoop(Q)` (attribute)
▷ `AssociatedRightBruckLoop(Q)` (attribute)

Returns: The left (resp. right) Bruck loop associated with a uniquely 2-divisible left (resp. right) Bol loop `Q`.

Let Q be a left Bol loop such that the mapping $x \mapsto x^2$ is a permutation of Q. Define a new operation $*$ on Q by $x * y = (x(y^2x))^{1/2}$. Then $(Q,*)$ is a left Bruck loop, called the *associated left Bruck loop*. (In fact, Bruck used the isomorphic operation $x*y = x^{1/2}(yx^{1/2})$ instead. Our approach is more natural in the sense that the left Bruck loop associated with a left Bruck loop is identical to the original loop.) Associated right Bruck loops are defined dually.

8.1.2 IsExactGroupFactorization

▷ `IsExactGroupFactorization(G, H1, H2)` (operation)

Returns: `true` if (`G`, `H1`, `H2`) is an exact group factorization.

Many right Bol loops can be constructed from exact group factorizations. The triple (G,H_1,H_2) is an *exact group factorization* if H_1, H_2 are subgroups of G such that $H_1H_2 = G$ and $H_1 \cap H_2 = 1$.

8.1.3 RightBolLoopByExactGroupFactorization

If (G,H_1,H_2) is an exact group factorization then $(G\times G, H_1\times H_2, T)$ with $T = \{(x,x^{-1})|x\in G\}$ is a loop folder that gives rise to a right Bol loop.▷ `RightBolLoopByExactGroupFactorization(arg)` (function)

Returns: The right Bol loop constructed from an exact group factorization. The argument `arg` can either be an exact group factorization `[G,H1,H2]`, or the tuple `[G,H]`, where `H` is a regular subgroup of `G`. We also allow `arg` to be separate entries rather than a list of entries.

8.2 Moufang Modifications

Drápal [Drá03] described two prominent families of extensions of Moufang loops. It turns out that these extensions suffice to obtain all nonassociative Moufang loops of order at most 64 if one starts with so-called Chein loops. We call the two constructions *Moufang modifications*. The library of Moufang loops included in LOOPS is based on Moufang modifications. See [DV06] for details.

8.2.1 LoopByCyclicModification

▷ `LoopByCyclicModification(Q, S, a, h)` (function)

Returns: The cyclic modification of a Moufang loop `Q` obtained from `S`, `a`$=\alpha$ and `h` described below.

Assume that Q is a Moufang loop with a normal subloop S such that Q/S is a cyclic group of order $2m$. Let $h \in S \cap Z(L)$. Let α be a generator of Q/S and write $Q = \bigcup_{i \in M} \alpha^i$, where $M = \{-m+1, \ldots, m\}$. 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 Q by $x * y = xyh^{\sigma(i+j)}$, where $x \in \alpha^i$, $y \in \alpha^j$, $i \in M$ and $j \in M$. Then $(Q, *)$ is a Moufang loop, a *cyclic modification* of Q.

8.2.2 LoopByDihedralModification

▷ `LoopByDihedralModification(Q, S, e, f, h)` (function)

Returns: The dihedral modification of a Moufang loop `Q` obtained from `S`, `e`, `f` and `h` as described below.

Let Q be a Moufang loop with a normal subloop S such that Q/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, γ be two involutions of Q/S such that $\alpha = \beta\gamma$ generates a cyclic subgroup of Q/S of order $2m$. Let $e \in \beta$ and $f \in \gamma$ be arbitrary. Then Q can be written as a disjoint union $Q = \bigcup_{i \in M}(\alpha^i \cup e\alpha^i)$, and also $Q = \bigcup_{i \in M}(\alpha^i \cup \alpha^i f)$. 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 Q by $x * y = xyh^{(-1)^r \sigma(i+j)}$, where $x \in \alpha^i \cup e\alpha^i$, $y \in \alpha^j \cup \alpha^j f$, $i \in M$, $j \in M$, $y \in G_r$ and $r \in \{0, 1\}$. Then $(Q, *)$ is a Moufang loop, a *dihedral modification* of Q.

8.2.3 LoopMG2

▷ `LoopMG2(G)` (function)

Returns: The Chein loop constructed from a group `G`.

Let G be a group. Let $\overline{G} = \{\overline{g} | g \in G\}$ be a disjoint copy of elements of G. Define multiplication $*$ on $Q = G \cup \overline{G}$ by $g * h = gh$, $g * \overline{h} = \overline{hg}$, $\overline{g} * h = \overline{gh^{-1}}$ and $\overline{g} * \overline{h} = h^{-1}g$, where g, $h \in G$. Then $(Q, *) = M(G, 2)$ is a so-called *Chein loop*, which is always a Moufang loop, and it is associative if and only if G is commutative.

8.3 Triality for Moufang Loops

Let G be a group and σ, ρ be automorphisms of G satisfying $\sigma^2 = \rho^3 = (\sigma\rho)^2 = 1$. Below we write automorphisms 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 Q. See [NV03] for more details.

For any Moufang loop Q, we can calculate the triality group as a permutation group acting on $3|Q|$ points. If the multiplication group of Q 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.

8.3.1 TrialityPermGroup

▷ `TrialityPermGroup(Q)` (function)

Returns: A record with components `G`, `rho`, `sigma`, where `G` is the canonical group with triality associated with a Moufang loop `Q`, and `rho`, `sigma` are the corresponding triality automorphisms.

8.3.2 TrialityPcGroup

▷ `TrialityPcGroup(Q)` (function)

This is a variation of `TrialityPermGroup` in which `G` is returned as a pc group.

8.4 Realizing Groups as Multiplication Groups of Loops

It is difficult to determine which groups can occur as multiplication groups of loops.

The following operations search for loops whose multiplication groups are contained within a specified transitive permutation group `G`. In all these operations, one can speed up the search by increasing the optional argument `depth`, the price being a much higher memory consumption. The argument `depth` is optimally chosen if in the permutation group `G` there are not many permutations fixing `depth` elements. It is safe to omit the argument or set it equal to 2.

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

8.4.1 AllLoopTablesInGroup

▷ `AllLoopTablesInGroup(G[, depth[, infolevel]])` (operation)

Returns: All Cayley tables of loops whose multiplication group is contained in the transitive permutation group `G`.

8.4.2 AllProperLoopTablesInGroup

▷ `AllProperLoopTablesInGroup(G[, depth[, infolevel]])` (operation)

Returns: All Cayley tables of nonassociative loops whose multiplication group is contained in the transitive permutation group `G`.

8.4.3 OneLoopTableInGroup

▷ `OneLoopTableInGroup(G[, depth[, infolevel]])` (operation)

Returns: A Cayley table of a loop whose multiplication group is contained in the transitive permutation group `G`.

8.4.4 OneProperLoopTableInGroup

▷ `OneProperLoopTableInGroup(G[, depth[, infolevel]])` (operation)

Returns: A Cayley table of a nonassociative loop whose multiplication group is contained in the transitive permutation group `G`.

8.4.5 AllLoopsWithMltGroup

▷ `AllLoopsWithMltGroup(G[, depth[, infolevel]])` (operation)

Returns: A list of all loops (given as sections) whose multiplication group is equal to the transitive permutation group `G`.

8.4.6 OneLoopWithMltGroup

▷ `OneLoopWithMltGroup(G[, depth[, infolevel]])` (operation)

Returns: One loop (given as a section) whose multiplication group is equal to the transitive permutation group `G`.

Example
```
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
```

Chapter 9

Libraries of Loops

Libraries of small loops form an integral part of LOOPS. The loops are stored in libraries up to isomorphism and, sometimes, up to isotopism.

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`. For example, when the library is called *left Bol*, the corresponding data file is called `data/leftbol.tbl` and the corresponding data structure is named `LOOPS_left_bol_data`.

In most cases, the array `LOOPS_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][k][s]` contains data necessary to produce the `s`th loop of order `LOOPS_my_library_data[1][k]` in the library.

The format of `LOOPS_my_library_data[3]` depends heavily on the particular library and is not standardized in any way. The data is often coded to save space.

9.1.1 LibraryLoop

▷ `LibraryLoop(libname, n, m)` (function)

Returns: The `m`th loop of order `n` from the library named `libname`.

9.1.2 MyLibraryLoop

▷ `MyLibraryLoop(n, m)` (function)

This is a template function that retrieves the `m`th loop of order `n` from the library named *my library*.

For example, the `m`th left Bol loop of order `n` is obtained via `LeftBolLoop(n,m)` or via `LibraryLoop("left Bol",n,m)`.

9.1.3 DisplayLibraryInfo

▷ `DisplayLibraryInfo(libname)` (function)

Returns: Brief information about the loops contained in the library named `libname`.

We are now going to describe the individual libraries.

9.2 Left Bol Loops and Right Bol Loops

The library named *left Bol* contains all nonassociative left Bol loops of order less than 17, including Moufang loops, as well as all left Bol loops of order pq for primes $p > q > 2$. There are 6 such loops of order 8, 1 of order 12, 2 of order 15, 2038 of order 16, and $(p+q-4)/2$ of order pq.

The classification of left Bol loops of order 16 was first accomplished by Moorhouse [Moo]. Our library was generated independently and it agrees with Moorhouse's results. The left Bol loops of order pq were classified in [KNV15].

9.2.1 LeftBolLoop

▷ `LeftBolLoop(n, m)` (function)

Returns: The `m`th left Bol loop of order `n` in the library.

9.2.2 RightBolLoop

▷ `RightBolLoop(n, m)` (function)

Returns: The `m`th right Bol loop of order `n` in the library.

REMARK: Only left Bol loops are stored in the library. Right Bol loops are retrieved by calling `Opposite` on left Bol loops.

9.3 Moufang Loops

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

9.3.1 MoufangLoop

▷ `MoufangLoop(n, m)` (function)

Returns: The `m`th Moufang loop of order `n` in the library.

For $n \le 63$, our catalog numbers coincide with those of Goodaire et al. [GMR99]. The classification of Moufang loops of order 64 and 81 was carried out in [NV07]. The classification of Moufang loops of order 243 was carried out by Slattery and Zenisek [SZ12].

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	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) can be obtained as `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 [NV07] for a classification of small code loops.

9.4.1 CodeLoop

▷ `CodeLoop(n, m)` (function)

Returns: The `m`th code loop of order `n` in the library.

9.5 Steiner Loops

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

Example
```
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
```

Our labeling of Steiner loops of order 16 coincides with the labeling of Steiner triple systems of order 15 in [CR99].

9.5.1 SteinerLoop

▷ `SteinerLoop(n, m)` (function)

Returns: The `m`th Steiner loop of order `n` in the library.

9.6 Conjugacy Closed Loops

The library named *RCC* contains all nonassocitive right conjugacy closed loops of order $n \le 27$ up to isomorphism. The data for the library was generated by Katharina Artic [Art15] who can also provide additional data for all right conjugacy closed loops of order $n \le 31$.

Let Q be a right conjugacy closed loop, G its right multiplication group and T its right section. Then $\langle T \rangle = G$ is a transitive group, and T is a union of conjugacy classes of G. Every right conjugacy closed loop of order n can therefore be represented as a union of certain conjugacy classes of a transitive group of degree n. This is how right conjugacy closed loops of order less than 28 are represented in LOOPS. The following table summarizes the number of right conjugacy closed loops of a given order up to isomorphism:

order	6	8	9	10	12	14	15	16	18	20
loops	3	19	5	16	155	97	17	6317	1901	8248
order	21	22	24	25	26	27				
loops	119	10487	471995	119	151971	152701				

9.6.1 RCCLoop and RightConjugacyClosedLoop

▷ `RCCLoop(n, m)` (function)
▷ `RightConjugacyClosedLoop(n, m)` (function)

Returns: The `m`th right conjugacy closed loop of order `n` in the library.

9.6.2 LCCLoop and LeftConjugacyClosedLoop

▷ `LCCLoop(n, m)` (function)
▷ `LeftConjugacyClosedLoop(n, m)` (function)

Returns: The `m`th left conjugacy closed loop of order `n` in the library.

REMARK: Only the right conjugacy closed loops are stored in the library. Left conjugacy closed loops are obtained from right conjugacy closed loops via `Opposite`.

The library named *CC* contains all nonassociative conjugacy closed loops of order $n \leq 27$ and also of orders $2p$ and p^2 for all primes p.

By results of Kunen [Kun00], for every odd prime p there are precisely 3 nonassociative conjugacy closed loops of order p^2. Csörgő and Drápal [CD05] described these 3 loops by multiplicative formulas on $\mathbb{Z}_{p^2}$ and $\mathbb{Z}_p \times \mathbb{Z}_p$ as follows:

- 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 [WJ75] constructed a nonassociative conjugacy closed loop of order $2p$ for every odd prime p, and Kunen [Kun00] showed that there are no other nonassociative conjugacy closed oops of this order. Here is the relevant multiplication formula on $\mathbb{Z}_2 \times \mathbb{Z}_p$: $(0,m)(0,n) = (0,m+n)$, $(0,m)(1,n) = (1,-m+n)$, $(1,m)(0,n) = (1,m+n)$, $(1,m)(1,n) = (0,1-m+n)$.

9.6.3 CCLoop and ConjugacyClosedLoop

▷ `CCLoop(n, m)` (function)
▷ `ConjugacyClosedLoop(n, m)` (function)

Returns: The `m`th conjugacy closed loop of order `n` in the library.

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.

9.7.1 SmallLoop

▷ `SmallLoop(n, m)` (function)

Returns: The mth loop of order n in the library.

9.8 Paige Loops

Paige loops are nonassociative finite simple Moufang loops. By [Lie87], there is precisely one Paige loop for every finite field.

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

9.8.1 PaigeLoop

▷ `PaigeLoop(q)` (function)

Returns: The Paige loop constructed over the finite field of order q. Only the case q=2 is implemented.

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.

See [DV09] for more on enumeration of nilpotent loops. For instance, there are 2623755 nilpotent loops of order 12, and 123794003928541545927226368 nilpotent loops of order 22.

9.9.1 NilpotentLoop

▷ `NilpotentLoop(n, m)` (function)

Returns: The mth nilpotent loop of order n in the library.

9.10 Automorphic Loops

The library named *automorphic* contains all nonassociative automorphic loops of order less than 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), all commutative automorphic loops of order 3, 9, 27 and 81 (there are 1, 2, 7 and 72 such loops, respectively, including abelian groups), and commutative automorphic loops Q of order 243 possessing a central subloop S of order 3 such that Q/S is not the elementary abelian group of order 81 (there are 118451 such loops).

9.10.1 AutomorphicLoop

▷ `AutomorphicLoop(n, m)` (function)

Returns: The mth automorphic loop of order n in the library.

9.11 Interesting Loops

The library named *interesting* contains some loops that are illustrative in the theory of loops. At this point, the library contains a nonassociative loop of order 5, a nonassociative nilpotent loop of order 6, a non-Moufang left Bol loop of order 16, the loop of sedenions of order 32 (sedenions generalize octonions), and the unique nonassociative simple right Bol loop of order 96 and exponent 2.

9.11.1 InterestingLoop

▷ `InterestingLoop(n, m)` (function)

Returns: The `m`th interesting loop of order `n` in the library.

9.12 Libraries of Loops Up To Isotopism

For the library named *small* we also provide the corresponding library of loops up to isotopism. In general, given a library named *libname*, the corresponding library of loops up to isotopism is named *itp lib*, and the loops can be retrieved by the template `ItpLibLoop(n,m)`.

9.12.1 ItpSmallLoop

▷ `ItpSmallLoop(n, m)` (function)

Returns: The `m`th small loop of order `n` up to isotopism in the library.

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.

Appendix A

Files

Below is a list of all relevant files forming the LOOPS package. Some technical files are not included. A typical user will not need to know any of this information. All paths are relative to the `pkg/loops` folder.

`../README.loops` (installation and usage instructions)
`init.g` (declarations)
`PackageInfo.g` (loading info for GAP 4.4)
`read.g` (implementations)
`data/automorphic.tbl` (library of automorphic loops)
`data/automorphic/*.*` (addition files for the library of automorphic loops)
`data/cc.tbl` (library of conjugacy closed 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/nilpotent.tbl` (library of small nilpotent loops)
`data/rcc.tbl` (library of right conjugacy closed loops)
`data/rcc/*.*` (additional files for the library of right conjugacy closed loops)
`data/paige.tbl` (library of Paige loops)
`data/small.tbl` (library of small loops)
`data/steiner.tbl` (library of Steiner loops)
`doc/*.*` (all documentation files)
`doc/loops.xml` (the main documentation file for GAPDoc)
`doc/loops.bib` (the main bibliography file for documentation)
`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/convert.gd .gi` (methods for data conversion and compression)
`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/memory.gd .gi` (memory management)
`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)
`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)

Appendix B

Filters

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 and IsLoop )
( 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 )
```

References

[Art59] R. Artzy. On automorphic-inverse properties in loops. *Proc. Amer. Math. Soc.*, 10:588–591, 1959. 37

[Art15] K. Artic. *On conjugacy closed loops and conjugacy closed loop folders*. PhD thesis, RWTH Aachen University, 2015. 51

[BP56] R. H. Bruck and L. J. Paige. Loops whose inner mappings are automorphisms. *Ann. of Math. (2)*, 63:308–323, 1956. 43

[Bru58] R. H. Bruck. *A survey of binary systems*. Ergebnisse der Mathematik und ihrer Grenzgebiete. Neue Folge, Heft 20. Reihe: Gruppentheorie. Springer Verlag, Berlin, 1958. 8

[CD05] P. Csörgő and A. Drápal. Left conjugacy closed loops of nilpotency class two. *Results Math.*, 47(3-4):242–265, 2005. 52

[CR99] C. J. Colbourn and A. Rosa. *Triple systems*. Oxford Mathematical Monographs. The Clarendon Press Oxford University Press, New York, 1999. 51

[DBGV12] D. A. S. De Barros, A. Grishkov, and P. Vojtěchovský. Commutative automorphic loops of order p^3. *J. Algebra Appl.*, 11(5):1250100, 15, 2012. 43

[Drá03] A. Drápal. Cyclic and dihedral constructions of even order. *Comment. Math. Univ. Carolin.*, 44(4):593–614, 2003. 46

[DV06] A. Drápal and P. Vojtěchovský. Moufang loops that share associator and three quarters of their multiplication tables. *Rocky Mountain J. Math.*, 36(2):425–455, 2006. 46

[DV09] D. Daly and P. Vojtěchovský. Enumeration of nilpotent loops via cohomology. *J. Algebra*, 322(11):4080–4098, 2009. 53

[Fen69] F. Fenyves. Extra loops. II. On loops with identities of Bol-Moufang type. *Publ. Math. Debrecen*, 16:187–192, 1969. 39

[GKN14] A. Grishkov, M. Kinyon, and G. P. Nagy. Solvability of commutative automorphic loops. *Proc. Amer. Math. Soc.*, 142(9):3029–3037, 2014. 43

[GMR99] E. G. Goodaire, S. May, and M. Raman. *The Moufang loops of order less than 64*. Nova Science Publishers Inc., Commack, NY, 1999. 50

[JKNV11] K. W. Johnson, M. K. Kinyon, G. P. Nagy, and P. Vojtěchovský. Searching for small simple automorphic loops. *LMS J. Comput. Math.*, 14:200–213, 2011. 43

[JKV12] P. Jedlička, M. Kinyon, and P. Vojtěchovský. Nilpotency in automorphic loops of prime power order. *J. Algebra*, 350:64–76, 2012. 43

[JM96] M. T. Jacobson and P. Matthews. Generating uniformly distributed random Latin squares. *J. Combin. Des.*, 4(6):405–437, 1996. 19

[KKP02] M. K. Kinyon, K. Kunen, and J. D. Phillips. Every diassociative A-loop is Moufang. *Proc. Amer. Math. Soc.*, 130(3):619–624, 2002. 43

[KKPV16] M. K. Kinyon, K. Kunen, J. D. Phillips, and P. Vojtěchovský. The structure of automorphic loops. *Trans. Amer. Math. Soc.*, 368(12):8901–8927, 2016. 43

[KNV15] M. K. Kinyon, G. P. Nagy, and P. Vojtěchovský. Bol loops and bruck loops of order pq. 2015. preprint. 50

[Kun00] K. Kunen. The structure of conjugacy closed loops. *Trans. Amer. Math. Soc.*, 352(6):2889–2911, 2000. 52

[Lie87] M. W. Liebeck. The classification of finite simple Moufang loops. *Math. Proc. Cambridge Philos. Soc.*, 102(1):33–47, 1987. 53

[Moo] G. E. Moorhouse. Bol loops of small order. http://www.uwyo.edu/moorhouse/pub/bol/. 50

[NV03] G. P. Nagy and P. Vojtěchovský. Octonions, simple Moufang loops and triality. *Quasigroups Related Systems*, 10:65–94, 2003. 46

[NV07] G. P. Nagy and P. Vojtěchovský. The Moufang loops of order 64 and 81. *J. Symbolic Comput.*, 42(9):871–883, 2007. 50, 51

[Pfl90] H. O. Pflugfelder. *Quasigroups and loops: introduction*, volume 7 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, 1990. 8

[PV05] J. D. Phillips and P. Vojtěchovský. The varieties of loops of Bol-Moufang type. *Algebra Universalis*, 54(3):259–271, 2005. 39

[SZ12] M. Slattery and A. Zenisek. Moufang loops of order 243. *Commentationes Mathematicae Universitatis Carolinae*, 53(3):423–428, 2012. 50

[Voj06] P. Vojtěchovský. Toward the classification of Moufang loops of order 64. *European J. Combin.*, 27(3):444–460, 2006. 35

[Voj15] P. Vojtěchovský. Three lectures on automorphic loops. *Quasigroups Related Systems*, 23(1):129–163, 2015. 43

[WJ75] R. L. Wilson Jr. Quasidirect products of quasigroups. *Comm. Algebra*, 3(9):835–850, 1975. 52

Index