Quasigroups and Related Systems **15** (2007), 77 – 94

Computing with small quasigroups and loops

Gábor P. Nagy and Petr Vojtěchovský

Abstract

This is a companion to our lectures *GAP and loops*, to be delivered at the *Workshops Loops 2007*, Prague, Czech Republic. In the lectures we introduce the GAP [6] package LOOPS [15], describe its capabilities, and explain in detail how to use it. In this paper we first outline the philosophy behind the package and its main features, and then we focus on three particular computational problems: construction of loop isomorphisms, classification of small Frattini Moufang loops of order 64, and the search for loops of nilpotency class higher than two with an abelian inner mapping group.

In particular, this is not a user's manual for LOOPS, which can be downloaded from the distribution website of LOOPS.

1. Main features

On the one hand, since there is no useful representation theory for quasigroups and loops, we have decided to represent quasigroups and loops in LOOPS by their Cayley tables, thus effectively limiting the scope of the package to quasigroups of order at most 500 or so. (A future project is to implement other loop representations, notably by connected transversals in groups.)

On the other hand, to take advantage of the powerful methods for groups already present in GAP, most calculations in LOOPS are delegated to the permutation groups associated with quasigroups, rather than performed on the level of Cayley tables. For instance, to decide if a loop is simple, we check whether its multiplication group is a primitive permutation group.

2000 Mathematics Subject Classification: Primary 20N05.

Keywords: loop, quasigroup, GAP, computation in nonassociative algebra, loop isomorphism, Latin square, Csörgő loop, small Frattini Moufang loop, LOOPS package, code loop.

This paper was written during the Marie Curie Fellowship of the first author at the University of Würzburg. The second author supported by the PROF 2006 grant of the University of Denver.

To avoid repeated calculations, we store most information obtained for a given quasigroup as its attribute. In GAP, there is no syntactical difference between calling a method or retrieving an attribute. For instance, when Q is a quasigroup then `Center(`Q`)` calculates and stores the center $Z(Q)$ of Q when called for the first time, while it retrieves the stored attribute $Z(Q)$ when called anytime later.

Moreover, GAP uses simple deduction process—filters—to obtain additional information about an object without an explicit user's request. For instance, if LOOPS knows that Q is a left Bol loop that is also commutative, the built-in filter `(IsMoufangLoop, IsLeftBolLoop and IsCommutative)` automatically deduces that Q is a Moufang loop and stores this information for Q. This is a powerful tool, since many filters built into LOOPS are deep theorems.

1.1. Creating quasigroups and loops

A (*quasigroup*) *Cayley table* is an $n \times n$ Latin square with integral entries $x_1 < \cdots < x_n$. A *canonical Cayley table* is a Cayley table with $x_1 = 1, \ldots, x_n = n$.

When T is a Cayley table, `QuasigroupByCayleyTable(`T`)` creates a quasigroup whose Cayley table is the canonical Cayley table obtained from T by replacing x_i with i. Should T be *normalized*—the first row and first column reads $x_1, \ldots, x_n$—then `LoopByCayleyTable(`T`)` returns the corresponding loop. The Cayley table of a quasigroup Q can be retrieved by `CayleyTable(`Q`)`.

Throughout this paper, we illustrate the methods of LOOPS by examples, often without any comments for self-explanatory commands. The syntax is that of GAP.

```
gap> Q := QuasigroupByCayleyTable([[2,1],[1,2]]); Elements(Q);
<quasigroup of order 2>
[ q1, q2 ]
gap> L := LoopByCayleyTable([[3,5],[5,3]]); Elements(L); L.2;
<loop of order 2>
[ l1, l2 ]
l2
gap> CayleyTable(Q);
[ [ 2, 1 ], [ 1, 2 ] ]
gap> Print(L);
<loop with multiplication table
[ [  1,  2 ],
  [  2,  1 ] ]
>
```

It is also possible to create quasigroups and loops by reading Cayley tables from files (with very relaxed conditions on the form of the Cayley table), by converting groups to quasigroups, by taking subquasigroups, subloops, factor loops, direct products, etc. See the manual for details.

1.2. Conversions

Even if a quasigroup happens to have a neutral element, it is not considered a loop in LOOPS unless it is declared as a loop. Similarly, a group of GAP is not considered a loop. We therefore provide conversions between these types of algebras:

```
gap> G := Group((1,2,3),(1,2)); AsLoop(G);
Group([ (1,2,3), (1,2) ])
<loop of order 6>
gap> Q := QuasigroupByCayleyTable([[2,1],[1,2]]);  AsLoop(Q);
<quasigroup of order 2>
<loop of order 2>
```

The neutral element of any loop L in LOOPS is always the first element of L, i.e., `One(`L`) = `L`.1`.

Given a quasigroup Q and elements $f, g \in Q$, the principal loop isotope (Q, f, g) of Q is obtained from Q via the isotopism $(R_g^{-1}, L_f^{-1}, \mathrm{id})$, cf. [17, p. 60]. Then (Q, f, g) is a loop with neutral element fg.

The conversion `AsLoop(`Q`)` works as follows, starting with a quasigroup Q:

(i) When Q does not have a neutral element, it is first replaced by the principal loop isotope $(Q, Q.1, Q.1)$, thus turning Q into a loop with neutral element $(Q.1)(Q.1)$.

(ii) When Q has a neutral element k, it is replaced by its isomorphic copy via the transposition $(1, k)$.

1.3. Subquasigroups and subloops

A new quasigroup Q_2 is frequently obtained as a subquasigroup of an existing quasigroup Q_1. Since all information about Q_2 is already contained in the Cayley table of Q_1, and since it is often desirable to have access to the embedding of Q_2 into Q_1, we provide a mechanism in LOOPS for maintaining the inclusion of Q_2 and Q_1.

When Q_1 is a quasigroup and S is a subset of Q_1, `Subquasigroup(`Q_1`,` S`)` returns the subquasigroup Q_2 of Q_1 generated by S. At the same time, the attribute `Parent(`Q_2`)` is set to `Parent(`Q_1`)`, hence ultimately pointing to the largest quasigroup from which Q_2 has been created. The elements of Q_2 and the Cayley table of Q_2 are then calculated relative to the parent of Q_2.

```
gap> L := AsLoop(Group((1,2,3),(1,2))); S := Subloop(L,[3]);
<loop of order 6>
<loop of order 2>
gap> Parent( S ) = L; PosInParent( S ); Elements( S );
true
[ 1, 3 ]
[ l1, l3 ]
gap> HasCayleyTable( S ); CayleyTable( S );
false
[ [ 1, 3 ], [ 3, 1 ] ]
```

Note that the Cayley table of a subquasigroup is created only upon user's request.

1.4. Bijections as permutations on $\{1,\ldots,n\}$

When calculating isomorphisms, isotopisms, or other bijections of quasigroups of order n, the result is always returned as a permutation (triple of permutations) of $\{1,\ldots,n\}$. Equivalently, the quasigroups in question are first replaced by isomorphic copies with canonical Cayley tables, and only then the bijections are calculated. It is always possible to reconstruct the original bijection using the attribute `PosInParent`.

1.5. A few words about the implementation

One of the biggest strengths of the computer algebra system GAP is that most algebraic structures can be defined within it. In this subsection we briefly explain how the variety of quasigroups is implemented in LOOPS. In order to understand the implementation, we will need the following GAP terminology:

– A *filter*, such as `IsInteger` and `IsPermGroup`, is a special unary function on the set of GAP objects which returns either `true` or `false`. Roughly speaking, a filter is an *a priori* attribute of an object.

- A *category* is a class of objects defined by a collection of filters. An object can lie in several categories. For example, a row vector lies in the categories `IsList` and `IsVector`.
- All GAP objects are partitioned into *families.* The family of an object determines its relation to other objects. For instance, all permutations form a family, and groups presented by generators and relations form another family. However, a family is not a collection of objects, but abstract information about objects.
- Beside its name, a family can have further *labels.*
- Every GAP object has a *type.* The type of an object determines if a given operation can be performed with that object, and if so, how it is to be performed. The type of an object is derived from its family and its filters.
- A given data structure can be made into an *object* by specifying its type, that is, its family and its filters.

The following function constructs a quasigroup Q with Cayley table `ct`. First we define a family corresponding to the elements of Q and tell GAP that it will consist of quasigroup elements. Then we objectify the individual elements in this family, and label the family by the set of its elements, by the size of Q, and by the Cayley table. Then we objectify Q whose family will be the *collection* of its elements. Finally, we set some important attributes of Q.

```
function( ct )
    local F, Q, elms, n;
    # constructing the family of the elements of this quasigroup
    F := NewFamily( "QuasigroupByCayleyTableFam", IsQuasigroupElement );
    # installing data ("labels") for the family
    n := Length ( ct );
    F!.size := n;
    elms := Immutable( List( [1..n], i -> Objectify(
        NewType( F, IsQuasigroupElement and IsQuasigroupElmRep), [ i ] ) ) );
    F!.set := elms;
    F!.cayleyTable := ct;
    # creating the quasigroup by turning it into a GAP object
    # the family of Q is the collection of its elements
    Q := Objectify( NewType( FamilyObj( elms ),
        IsQuasigroup and IsAttributeStoringRep ), rec() );
    # setting some attributes for the quasigroup
    SetSize( Q, n );
    SetAsSSortedList( Q, elms );
```

```
    SetCayleyTable( Q, ct );
    return Q;
end;
```

Operations in GAP are overloaded, i.e., the same operation can be applied to different types of objects. In order to deal with this situation, GAP uses a method selection: When an operation is called, GAP first checks the types of the arguments, and then selects the appropriate method.

Here is how the multiplication of two quasigroup elements is implemented:

```
InstallMethod( \*, "for two quasigroup elements",
    IsIdenticalObj,
    [ IsQuasigroupElement, IsQuasigroupElement ],
function( x, y )
    local F;
    F := FamilyObj( x );
    return F!.set[ F!.cayleyTable[ x![ 1 ] ][ y![ 1 ] ] ];
end );
```

Note that the underlying quasigroup is easily accessed since the element x *knows* into which quasigroup it belongs.

2. What is in the package

Here is a very brief overview of the methods implemented in LOOPS, version 1.4.0. See the manual for (much) more details. Argument Q stands for a quasigroup, and L for a loop. Thus the methods with argument Q apply to both quasigroups and loops, while those with argument L apply only to loops. Any additional restrictions on the arguments are listed in parentheses. The symbol $\triangleright$ is a shortcut for *returns*.

2.1. Basic methods and attributes

Cayley tables and elements:

`Elements(`Q`)` $\triangleright$ list of elements of Q,
`CayleyTable(`Q`)` $\triangleright$ Cayley table of Q,
`One(`L`)` $\triangleright$ the neutral element of L,
`MultiplicativeNeutralElement(`Q`)` $\triangleright$ the neutral element of Q, or fail
`Size(`Q`)` $\triangleright$ the size of Q,
`Exponent(`L`)` $\triangleright$ the exponent of L (L power-associative).

Arithmetic operations:
`LeftDivision`(x, y) ▷ $x\backslash y$,
`RightDivision`(x, y) ▷ x/y,
`LeftDivisionCayleyTable`(Q) ▷ Cayley table of left division in Q,
`RightDivisionCayleyTable`(Q) ▷ Cayley table of right division in Q.

Powers and inverses:
`LeftInverse`(x) ▷ x^λ, where $x^\lambda x = 1$,
`RightInverse`(x) ▷ x^ρ, where $xx^\rho = 1$,
`Inverse`(x) ▷ the two-sided inverse of x, if it exists.

Associators and commutators:
`Associator`(x, y, z) ▷ the unique element u with $(xy)z = (x(yz))u$,
`Commutator`(x, y) ▷ the unique element v with $xy = (yx)v$.

Generators:
`GeneratorsOfQuasigroup`(Q) ▷ a generating subset of Q,
`GeneratorsOfLoop`(L) ▷ a generating subset of L,
`GeneratorsSmallest`$(\mathtt{Q})$ ▷ a generating subset of Q of size $\leqslant \log_2 |Q|$.

Subquasigroups:
`IsSubquasigroup`(Q, S) ▷ true if S is a subquasigroup of Q,
`IsSubloop`(L, S) ▷ true if S is a subloop of L,
`AllSubloops`(L) ▷ list of all subloops of L,
`RightCosets`(L, S) ▷ right cosets modulo S ($S \leq L$),
`RightTransversal`(L, S) ▷ a right transversal modulo S ($S \leq L$).

Translations and sections:
`LeftTranslation`(Q, x) ▷ the left translation L_x by x in Q ($x \in Q$),
`RightTranslation`(Q, x) ▷ the right translation R_x by x in Q ($x \in Q$),
`LeftSection`(Q) ▷ the set of all left translations in Q,
`RightSection`(Q) ▷ the set of all right translations in Q.

Multiplication groups:
`LeftMultiplicationGroup`(Q) ▷ the left multiplication group of Q,
`RightMultiplicationGroup`(Q) ▷ the right multiplication group of Q,
`MultiplicationGroup`(Q) ▷ the multiplication group of Q,
`RelativeLeftMultiplicationGroup`(L, S) ▷ the group generated by all left translations of L restricted to S ($S \leq L$),
`RelativeRightMultiplicationGroup`(L, S) ▷ the group generated by all right translations of L restricted to S ($S \leq L$),
`RelativeMultiplicationGroup`(L, S) ▷ the group generated by all translations of L restricted to S ($S \leq L$).

Inner mapping groups:

`InnerMappingGroup(`L`)` ▷ the inner mapping group of L,
`LeftInnerMappingGroup(`L`)` ▷ the group generated by $L_{yx}^{-1}L_yL_x$,
`RightInnerMappingGroup(`L`)` ▷ the group generated by $R_{xy}^{-1}R_yR_x$.

Nuclei:

`LeftNucleus(`Q`)` ▷ the left nucleus of Q,
`RightNucleus(`Q`)` ▷ the right nucleus of Q,
`MiddleNucleus(`Q`)` ▷ the middle nucleus of Q,
`Nuc(`Q`)`, `NucleusOfQuasigroup(`Q`)` ▷ the nucleus of Q.

Commutant, center and associator subloop:

`Commutant(`Q`)` ▷ $\{x \in Q;\ xy = yx \text{ for every } y \in Q\}$,
`Center(`Q`)` ▷ the center of Q,
`AssociatorSubloop(`L`)` ▷ the smallest $S \trianglelefteq L$ such that L/S is a group.

Normal subloops:

`IsNormal(`L, S`)` ▷ true if S is a normal subloop of L,
`NormalClosure(`L, S`)` ▷ the smallest normal subloop of L containing S,
`IsSimple(`L`)` ▷ true if L is a simple loop.

Factor loops:

`FactorLoop(`L, N`)` ▷ L/N (N normal subloop of L),
`NaturalHomomorphismByNormalSubloop(`L, N`)` ▷ the projection $L \to L/N$ (N normal subloop of L).

Central nilpotency and central series:

`NilpotencyClassOfLoop(`L`)` ▷ the (central) nilpotency class of L,
`IsNilpotent(`L`)` ▷ true if L is nilpotent,
`IsStronglyNilpotent(`L`)` ▷ true if the mult. group of L is nilpotent,
`UpperCentralSeries(`L`)` ▷ the upper central series of L,
`LowerCentralSeries(`L`)` ▷ the lower central series of L,

Solvability:

`IsSolvable(`L`)` ▷ true if L is solvable,
`DerivedSubloop(`L`)` ▷ the derived subloop of L,
`DerivedLength(`L`)` ▷ the derived length of L,
`FrattiniSubloop(`L`)` ▷ the Frattini subloop of L (L strongly nilpotent).

Isomorphisms and automorphisms:

`IsomorphismLoops(`L, M`)` ▷ an isomorphism of loops $L \to M$, or fail,
`LoopsUpToIsomorphism(`ls`)` ▷ filtered list ls of loops up to isomorphism,
`AutomorphismGroup(`L`)` ▷ the automorphism group of L,
`IsomorphicCopyByPerm(`Q, p`)` ▷ an isomorphic copy of Q via the permutation p,
`IsomorphicCopyByNormalSubloop(`L, S`)` ▷ an isomorphic copy of L in

which $S \trianglelefteq L$ occupies the first $|S|$ elements of L and where the remaining elements correspond to the cosets of S in L.

Isotopisms:

`IsotopismLoops`(L, M) ▷ an isotopism $L \to M$, or fail,
`LoopsUpToIsotopism`(ls) ▷ filtered list ls of loops up to isotopism.

2.2. Testing properties of quasigroups and loops

Associativity, commutativity and generalizations:

`IsAssociative`(Q) ▷ true if Q is associative,
`IsCommutative`(Q) ▷ true if Q is commutative,
`IsPowerAssociative`(L) ▷ true if L is power associative,
`IsDiassociative`(L) ▷ true if L is diassociative.

Inverse properties:

`HasLeftInverseProperty`(L) ▷ true if $x^\lambda(xy) = y$,
`HasRightInverseProperty`(L) ▷ true of $(yx)x^\rho = y$,
`HasInverseProperty`(L) ▷ true if $x^\lambda(xy) = y = (yx)x^\rho$,
`HasTwosidedInverses`(L) ▷ true if $x^\lambda = x^\rho$,
`HasWeakInverseProperty`(L) ▷ true if $(xy)^\lambda x = y^\lambda$,
`HasAutomorphicInverseProperty`(L) ▷ true if $(xy)^\lambda = x^\lambda y^\lambda$,
`HasAntiautomorphicInverseProperty`(L) ▷ true if $(xy)^\lambda = y^\lambda x^\lambda$.

Some properties of quasigroups:

`IsSemisymmetric`(Q) ▷ true if $(xy)x = y$,
`IsTotallySymmetric`(Q) ▷ true if Q is semisymmetric and commutative,
`IsIdempotent`(Q) ▷ true it $x^2 = x$,
`IsSteinerQuasigroup`(Q) ▷ true if Q is totally symm. and commutative,
`IsUnipotent`(Q) ▷ true if $x^2 = y^2$,
`IsLeftDistributive`(Q) ▷ true if $x(yz) = (xy)(xz)$,
`IsRightDistributive`(Q) ▷ true if $(xy)z = (xz)(yz)$,
`IsDistributive`(Q) ▷ true if Q is left and right distributive,
`IsEntropic`(Q), `IsMedial`(Q) ▷ true if $(xy)(uv) = (xu)(yv)$.

Loops of Bol-Moufang type:

`IsExtraLoop`(L) ▷ true if $x(y(zx)) = ((xy)z)x$,
`IsCLoop`(L) ▷ true if $x(y(yz)) = ((xy)y)z$,
`IsMoufangLoop`(L) ▷ true if $(xy)(zx) = (x(yz))x$,
`IsRCLoop`(L) ▷ true if $x((yz)z) = (xy)(zz)$,
`IsLCLoop`(L) ▷ true if $(xx)(yz) = (x(xy))z$,
`IsRightBolLoop`(L) ▷ true if $x((yz)y) = ((xy)z)y$,
`IsLeftBolLoop`(L) ▷ true if $x(y(xz)) = (x(yx))z$,

`IsFlexible(`Q`)` ▷ true if $x(yx) = (xy)x$,
`IsRightNuclearSquareLoop(`L`)` ▷ true if $x(y(zz)) = (xy)(zz)$,
`IsMiddleNuclearSquareLoop(`L`)` ▷ true if $x((yy)z) = (x(yy))z$,
`IsLeftNuclearSquareLoop(`L`)` ▷ true if $(xx)(yz) = ((xx)y)z$,
`IsRightAlternative(`Q`)` ▷ true if $x(yy) = (xy)y$,
`IsLeftAlternative(`Q`)` ▷ true if $(xx)y = x(xy)$,
`IsAlternative(`Q`)` ▷ true if it is both left and right alternative.

Power alternative loops:

`IsLeftPowerAlternative(`L`)` ▷ true if $x^n(x^m y) = x^{n+m}y$,
`IsRightPowerAlternative(`L`)` ▷ true if $(xy^n)y^m = xy^{n+m}$,
`IsPowerAlternative(`L`)` ▷ true if L is left and right power alternative.

Conjugacy closed loops:

`IsLCCLoop(`L`)` ▷ true if left translations are closed under conjugation,
`IsRCCLoop(`L`)` ▷ true if right translations are closed under conjugations,
`IsCCLoop(`L`)` ▷ true if L is left and right conjugacy closed.

Additional varieties of loops:

`IsLeftBruckLoop(`L`)`, `IsLeftKLoop(`L`)` ▷ true if L is left Bol and has the automorphic inverse property,
`IsRightBruckLoop(`L`)`, `IsRightKLoop(`L`)` ▷ true if L is right Bol and has the automorphic inverse property.

Here is a nice, albeit trivial illustration of the filters built into the LOOPS package:

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

2.3. Libraries

Several libraries of small loops up to isomorphism are included in LOOPS. As of version 1.4.0, the libraries contain:

- all nonassociative left Bol loops of order $\leqslant 16$,
- all nonassociative Moufang loops of order $\leqslant 64$ and $= 81$,

- all nonassociative Steiner loops of order $\leqslant 16$,
- all (three) nonassociative conjugacy closed loops of order p^2, for every odd prime p,
- all (one) nonassociative conjugacy closed loops of order $2p$, for every odd prime p,
- the smallest nonassociative simple Moufang loop (of order 120),
- all nonassociative loops of order $\leqslant 6$.

There is also a library of all nonassociative loops of order $\leqslant 6$ up to isotopism.

The mth loop of order n in a given library can be retrieved via

$$\texttt{LeftBolLoop}(n,m), \quad \texttt{MoufangLoop}(n,m),$$

and so on.

We took great care to store the information in the libraries efficiently. For instance, the library of Moufang loops can be packed into less than 18 kilobytes, hence averaging about 4 bytes per loop.

Remark 2.1. All nonassociative Moufang loops of orderless than 64 can be found in [7]. Our numbering for these loops agrees with [7].

The 4262 nonassociative Moufang loops of order 64 were first constructed in [18], but it was proved (computationally) only in [16] that the list is complete.

The 2038 nonassociative left Bol loops of order 16 were enumerated for the first time by Moorhouse [12]. The first author obtained the same result by a different method, on which he will report in a separate paper [14].

The fact that for every odd prime p there are precisely three nonassociative conjugacy closed loops of order p^2 was established by Kunen [10]. Drápal and Csörgő derived simple formulas for multiplication in these three loops [4]. When p is an odd prime, Wilson [19] constructed a nonassociative conjugacy closed loop of order $2p$, and Kunen [10] showed there are no other such loops.

Our counts of small loops agree with the known results, e.g. [11].

The library of small Steiner loops is based on [2].

3. Constructing isomorphisms

There does not appear to be much research on the problem of finding an isomorphism between loops. In this section we explain the approach used in LOOPS. It works surprisingly well for many varieties of loops, including Moufang loops.

Let Q be a loop, and let $\mathcal{P}$ be a set of properties (of elements) invariant under isomorphisms. The nature of $\mathcal{P}$ depends on Q. For instance, when Q is power-associative, one of the invariant properties for an element x might be the order $|x|$.

Given $\mathcal{P}$ and a collection $\mathcal{C}$ of loops, define an equivalence on the (disjoint) union of $\mathcal{C}$ by $x \sim y$ if and only if $\varphi(x) = \varphi(y)$ for every $\varphi \in \mathcal{P}$. Then, if $f : Q \to L$ is an isomorphism and $\mathcal{C} = \{Q, L\}$, we must have $x \sim f(x)$ for every $x \in Q$. In other words, $\mathcal{P}$ partitions the elements into blocks invariant under isomorphism.

In order to find an isomorphism, we need a set of invariants $\mathcal{P}$ that is easy to calculate but results in a fine partition.

We have used the following invariants $\mathcal{P}$ for power-associative loops:

$$\begin{aligned}
\varphi_1(x) &= |x|,\\
\varphi_2(x) &= |\{y;\ y^2 = x\}|,\\
\varphi_3(x) &= |\{y;\ y^4 = x\}|,\\
\varphi_{4,k}(x) &= |\{y;\ xy = yx,\ |y| = k\}|,\ \text{for } k \geq 1.
\end{aligned}$$

The algorithm searching for an isomorphism $f : Q \to L$ first orders the equivalence classes of $\sim$ by increasing size on both Q and L. If the equivalence class sizes of Q and L do not match, it is clear that no isomorphism $f : Q \to L$ exists, and `fail` is returned. Otherwise, a backtrack search attempts to find an isomorphism respecting the partitions of $\sim$.

It would be an interesting project to analyze the speed of the algorithm depending on the choice of $\mathcal{P}$. We do not claim that the above $\mathcal{P}$ is optimized in any sense. Note, for instance, that the invariants φ_2, φ_3 are useless for many power associative loops of odd order, and $\varphi_{4,k}$ are useless for all commutative loops.

4. Classification of small Frattini Moufang loops of order 64

Let L be a loop and let the *Frattini subloop* $\Phi(L)$ be the normal subloop generated by all squares, commutators and associators of L. In other words, $\Phi(L)$ is the smallest normal subloop such that $L/\Phi(L)$ is an elementary abelian p-group. Following Hsu [9], we say that L is a *small Frattini p-loop* if $|\Phi(L)| \leqslant p$.

In this section, L will denote a small Frattini Moufang 2-loop of order 2^{n+1}. Moreover, in order to avoid trivialities, we assume that $|\Phi(L)| = 2$. Clearly, $\Phi(L) \leq Z(L)$, L is nilpotent of class 2, and it has a unique nontrivial square, commutator and associator element.

Remark 4.1. Small Frattini Moufang 2-loops are also called *code loops* due to their connection to doubly even linear binary codes. Some of these loops play an important role in the description of large sporadic simple groups.

We consider $V = L/\Phi(L)$ as a vector space over $\mathbb{F}_2$, and we identify $\Phi(L)$ and $\mathbb{F}_2$. In particular, we sometimes write the group operations additively.

Let us take arbitrary elements $u = x \mod \Phi(L)$, $v = y \mod \Phi(L)$, $w = z \mod \Phi(L)$ of V. Then, the following maps are well defined:

$$\begin{array}{ll} \sigma : V \to \mathbb{F}_2, & \sigma(u) = x^2, \\ \kappa : V \times V \to \mathbb{F}_2, & \kappa(u,v) = [x,y], \\ \alpha : V \times V \times V \to \mathbb{F}_2, & \alpha(u,v,w) = [x,y,z]. \end{array}$$

Moreover, α is an alternating trilinear form, κ is alternating, and we have

$$\begin{aligned} \sigma(u+v) &= \sigma(u) + \sigma(v) + \kappa(u,v), \\ \kappa(u+v,w) &= \kappa(u,w) + \kappa(v,w) + \alpha(u,v,w). \end{aligned}$$

Hence, by definition, V is a *symplectic cubic space*.

There are different ways in which a small Frattini Moufang 2-loop is obtained from a symplectic cubic space (cf. Griess [8], Chein and Goodaire [1], Hsu [9]). All of the above constructions induct on the dimension of V. In contrast, a new approach, [13], takes advantage of *groups with triality* and constructs the loop globally.

For this, let σ_i, κ_{ij} and α_{ijk} be the structure constants of σ, κ, α with respect to a fixed basis ov V. We define the group G with gerenators

g_i, f_i, h_i, $i \in \{1, \dots, n\}$, u and v by the following relations:

$$\begin{aligned}
&g_i^2 = u^{\sigma_i},\ f_i^2 = v^{\sigma_i},\ h_i^2 = u^2 = v^2 = 1,\\
&[g_i, g_j] = u^{\kappa_{ij}},\ [f_i, f_j] = v^{\kappa_{ij}},\\
&[g_i, f_j] = (uv)^{\kappa_{ij}} \prod_{k=1}^{n} h_k^{\alpha_{ijk}},\\
&[g_i, h_j] = u^{\delta_{ij}},\ [f_i, h_j] = v^{\delta_{ij}},\\
&[h_i, h_j] = [g_i, u] = [f_i, u] = [h_i, u] = [g_i, v] = [f_i, v] = [h_i, v] = 1.
\end{aligned}$$

Then, G is a group and the maps

$$\begin{aligned}
\tau &: g_i \leftrightarrow f_i,\ h_i \mapsto h_i,\ u \leftrightarrow v\\
\rho &: g_i \mapsto f_i,\ f_i \mapsto (g_i f_i)^{-1},\ h_i \mapsto h_i,\ u \mapsto v,\ v \mapsto uv
\end{aligned}$$

extend to *triality automorphisms* of G. The following function returns the Moufang loop associated to the group G with triality automorphisms τ, ρ:

```
TrialityGroupToLoop := function( G, tau, rho )
   local ccl, ct;
   ccl := Elements( ConjugacyClass( G, tau ) );
   ct := List( ccl, s1 ->
            List( ccl, s2 ->
               Position( ccl, s1^rho * s2^(rho^2) * s1^rho )
            )
         );
   return LoopByCayleyTable( NormalizedQuasigroupTable( ct ) );
end;
```

To complete the classification of small Frattini Moufang 2-loops of order 64, it now suffices to classify the symplectic cubic spaces of order 32. For a fixed basis, such a space is given by

$$\binom{5}{3} + \binom{5}{2} + 5 = 25$$

structure constants, which give rise to a 25-dimensional vector space W over $\mathbb{F}_2$.

Any linear map A of V defines a new symplectic cubic space with maps

$$\sigma^A(u) = \sigma(Au), \quad \kappa^A(u, v) = \kappa(Au, Av), \quad \alpha^A(u) = \alpha(Au, Av, Aw),$$

and hence A induces a linear map on W. This defines an action of $GL(5, 2)$ on W.

It is easy to show the 1-1 correspondence of loop isomorphisms and linear isomorphisms of symplectic cubic spaces. This implies that the orbits of $GL(5,2)$ on W will correspond precisely to the isomorphism classes of small Frattini Moufang 2-loops of order 64.

Since $|GL(5,2)|$ and 2^{25} are still too large for GAP to compute the needed orbits, one has to have a closer look at invariant subspaces of W. Once this is done, the classification is complete, with the result that there are precisely 80 nonisomorphic small Frattini Moufang loops of order 64.

5. An interesting Csörgő loop

One of the longer-standing problems in loop theory was the question if there is a loop with nilpotency class higher than two whose inner mapping group is abelian. In [3], Csörgő constructed such a loop (of order 128 and nilpotency class 3). The following GAP code returns this loop L. The code follows [3], where some insight is given.

```
# constructing a group of order 8192 by presenting relations
f := FreeGroup(13);
G := f/[ f.1^2, f.2^2, f.3^2, f.4^2, f.5^2, f.6^2, f.7^2, f.8^2, f.9^2, f.10^2,
f.11^2, f.12^2, f.13^2, (f.1*f.2)^2, (f.1*f.3)^2, (f.1*f.4)^2, (f.1*f.5)^2,
(f.1*f.6)^2, (f.1*f.7)^2, (f.1*f.8)^2, (f.1*f.9)^2, (f.1*f.10)^2, (f.1*f.11)^2,
(f.1*f.12)^2, (f.1*f.13)^2, (f.2*f.3)^2, (f.2*f.4)^2, (f.3*f.4)^2, (f.2*f.5)^2,
(f.2*f.6)^2, (f.2*f.7)^2, (f.3*f.5)^2, (f.3*f.6)^2, (f.3*f.7)^2, (f.4*f.5)^2,
(f.4*f.6)^2, (f.4*f.7)^2, (f.2*f.9)^2, (f.2*f.10)^2, (f.3*f.8)^2, (f.3*f.10)^2,
(f.4*f.8)^2, (f.4*f.9)^2, f.1*f.2*f.8*f.2*f.8, f.1*f.3*f.9*f.3*f.9,
f.1*f.4*f.10*f.4*f.10, (f.2*f.11)^2, (f.2*f.12)^2, (f.2*f.13)^2, (f.3*f.11)^2,
(f.3*f.12)^2, (f.3*f.13)^2, (f.4*f.11)^2, (f.4*f.12)^2, (f.4*f.13)^2, (f.5*f.6)^2,
(f.5*f.7)^2, (f.6*f.7)^2, (f.5*f.9)^2, (f.5*f.10)^2, (f.6*f.8)^2, (f.6*f.10)^2,
(f.7*f.8)^2, (f.7*f.9)^2, f.1*f.5*f.8*f.5*f.8, f.1*f.6*f.9*f.6*f.9,
f.1*f.7*f.10*f.7*f.10, (f.5*f.12)^2, (f.5*f.13)^2, (f.6*f.11)^2, (f.6*f.13)^2,
(f.7*f.11)^2, (f.7*f.12)^2, f.1*f.11*f.5*f.11*f.5, f.1*f.12*f.6*f.12*f.6,
f.1*f.13*f.7*f.13*f.7, f.2*f.5*f.9*f.10*f.9*f.10, f.3*f.6*f.8*f.10*f.8*f.10,
f.4*f.7*f.8*f.9*f.8*f.9, (f.8*f.11)^2, (f.9*f.12)^2, (f.10*f.13)^2,
f.8*f.12*f.8*f.4*f.12*f.7, f.8*f.13*f.8*f.3*f.13*f.6, f.10*f.11*f.10*f.3*f.11*f.6,
f.9*f.11*f.9*f.11*f.7, f.9*f.13*f.9*f.13*f.5, f.10*f.12*f.10*f.12*f.5,
(f.11*f.12)^2, (f.11*f.13)^2, (f.12*f.13)^2 ];
# auxiliary data
g := GeneratorsOfGroup(G);
N := Subgroup( G, [ g[5], g[6], g[7], g[1] ] );
W := Subgroup( G, [ g[5]*g[2], g[6]*g[3], g[7]*g[4], g[1] ] );
A_0 := [ One(G), g[8], g[9], g[10], g[8]*g[9], g[8]*g[10], g[9]*g[10]*g[2],
    g[8]*g[9]*g[10]*g[2] ];
B_0 := [ One(G), g[8]*g[11], g[9]*g[12], g[10]*g[13], g[8]*g[11]*g[9]*g[12],
    g[8]*g[11]*g[10]*g[13]*g[3], g[9]*g[12]*g[10]*g[13],
    g[8]*g[11]*g[9]*g[12]*g[10]*g[13]*g[3] ];
A := Union( List( Elements( N ), x -> A_0*x ) );
B := Union( List( Elements( W ), x -> B_0*x ) );
```

```
H := Subgroup( G, [ g[2], g[3], g[4], g[11], g[12], g[13] ] );
# constructing the loop
ListPosition := function( S, x )
    local i; i := 1; while not x in S[i] do i := i + 1; od; return i;
end;
m := MappingByFunction( Domain(Elements( G)), Domain([1..8192]),
    x -> Position( Elements(G), x ) );
CA := List( A, x -> x*Elements( H ) );
mCA := List( CA, c -> Set( c, x -> x^m ) );
T := List([1..128],i->[1..128]);
for ii in [1..128] do for jj in [1..128] do
    T[ii][jj] := ListPosition( mCA, (A[ii]*B[jj])^m );
od; od;
p := SortingPerm( T[1] );
T := List( T, r -> Permuted( r, p ) );
L := LoopByCayleyTable( T );
```

In addition, the following properties hold for L: (a) the nucleus of L is elementary abelian of order 16, (b) the left and middle nuclei have order 32, (c) the right nucleus has order 16, (d) the two-element center coincides with the associator subloop.

An interesting, more symmetric loop K is obtained from L by this greedy algorithm:

Given a groupoid Q, let $\mu(Q) = |\{(a,b,c) \in Q \times Q \times Q;\ a(bc) \neq (ab)c\}|$. Hence $\mu(Q)$ is a crude measure of (non)associativity of Q.

Let T be a multiplication table of L split into blocks of size 16×16 according to the cosets of the nucleus of L. Let h be the nontrivial central element of L.

(*) For $1 \leqslant i \neq j \leqslant 16$, let T_{ij} be obtained from T by multiplying the (i,j)th block and the (j,i)th block of T by h. Let (s,t) be such that $\mu(T_{st})$ is minimal among all $\mu(T_{ij})$. If $\mu(T_{st}) \geq \mu(T)$, stop, and return T. Else replace T by T_{st}, and repeat (*).

It turns out that the multiplication table T found by the above greedy algorithm yields another loop K of nilpotency class 3 whose inner mapping group is abelian. In addition, the following properties hold for K: (a) the nucleus is elementary abelian of order 16, (b) the left, middle, and right nuclei have order 64, (c) the two-element center coincides with the associator subloop. In particular, K is not isomorphic to L. Among other peculiar features, it contains a nonassociative power associative loop of order 64 that is the union of its nuclei.

The construction of L takes a minute or so in GAP, since calculations in free groups are slow. A more direct, systematic, and much faster construction of L and K will be presented elsewhere [5].

References

[1] **O. Chein and E. G. Goodaire**: *Moufang loops with a unique nonidentity commutator (associator, square)*, J. Algebra **130** (1990), 369 – 384.

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

[3] **P. Csörgő**: *Abelian inner mappings and nilpotency class greater than two*, European J. Combin., to appear.

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

[5] **A. Drápal and P. Vojtěchovský**: *Explicit constructions of loops with commuting inner mappings*, submitted.

[6] **The GAP Group**: *GAP – Groups, Algorithms, and Programming, Version* 4.4.9; 2006. `http://www.gap-system.org`

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

[8] **R. L. Griess, Jr.**: *Code loops*, J. Algebra **100** (1986), 224 – 234.

[9] **T. Hsu**:*Explicit constructions of code loops as centrally twisted products*, Math. Proc. Cambridge Philos. Soc. **128** (2000), 223 – 232.

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

[11] **B. D. McKay, A. Meynert and W. Myrvold**: *Small Latin squares, quasigroups and loops*, J. Combinatorial Designs, to appear.

[12] **G. E. Moorhouse**: *Bol loops of small order*, available at
http://www.uwyo.edu/moorhouse/pub/bol/

[13] **G. P. Nagy**: *Direct construction of code loops*, Discr. Math., to appear.

[14] **G. P. Nagy**: *Doubling of finite Bol loops*, in preparation, 2007.

[15] **G. P. Nagy and P. Vojtěchovský**: LOOPS – a GAP package, version 1.4.0, Feb. 2007, `http://www.math.du.edu/loops`

[16] **G. P. Nagy and P. Vojtěchovský**: *The Moufang loops of order* 64 *and* 81, submitted.

[17] **H. O. Pflugfelder**: *Quasigroups and Loops: Introduction*, Sigma Series in Pure Math. **8**, Heldermann Verlag, Berlin, 1990.

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

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

Received February 25, 2007

Gábor P. Nagy
Bolyai Institute
University of Szeged
Aradi vértanúk tere 1
H-6720 Szeged
Hungary
E-mail: nagyg@math.u-szeged.hu

Petr Vojtěchovský
Department of Mathematics
University of Denver
2360 S Gaylord St
Denver, Colorado 80208
U.S.A.
E-mail: petr@math.du.edu