# Chapter 4: Universal algebra

- Direct products
- Opposite (inheritance of properties)
- Subrightquasigroup, cosets and transversals
- Intersections and joins
- Congruences
- Factor algebras
- Demonstration: *Paige loops constructed in several ways*

# Direct products

```
gap> G := Group((1,2));;
gap> L := LoopByCayleyTable( [[1,2,3],[2,3,1],[3,1,2]] );;
gap> D := DirectProduct( G, L );
<loop of size 6>
```

- note the form of elements in the direct product

```
gap> D.1;
l[ (), l1 ]
```

- any list of right quasigroups, quasigroups, loops and groups works

```
gap> R := ProjectionRightQuasigroup( [1..3] );;
gap> DirectProduct( G, L, R );
<right quasigroup of size 18>
```

## Opposite

- $x *^{op} y = y * x$
- dual properties are inherited

```
gap> B := LeftBolLoop( 8, 1 );
LeftBolLoop( 8, 1 )
gap> OppositeLoop( B );
<right Bol loop of size 8>
```

# Subralgebras

- refer recursively to the parent algebra
- do not duplicate underlying set elements, only point to them
- indexing: parent and canonical

```
gap> mult := function( x, y ) return (x+y) mod 8; end;
gap> Q := RightQuasigroupByFunction([0..7], mult );;
gap> A := Subrightquasigroup( Q, [2] );
<right quasigroup of size 4>
gap> B := Subrightquasigroup( A, [4] );
<right quasigroup of size 2>
gap> Parent( B ) = Q;
true
```

# Some methods for subalgebras

- all subalgebras

```
gap> AllSubrightquasigroups( Q );;
gap> AllMinimalSubquasigroups( Q );;
gap> AllMaximalSubloops( Q );;
```

- transversals and cosets

```
gap> RightCosets( Q, S );;
gap> LeftTransversal( Q, S );;
```

# Intersections and joins

```
gap> P := ProjectionRightQuasigroup( 10 );;
gap> A := Subrightquasigroup( P, [1..4] );;
gap> B := Subrightquasigroup( P, [3..7] );;
gap> Intersection( A, B );
<associative quandle of size 2>
gap> Elements( last );
[ r3, r4 ]
gap> Join( A, B );
<associative quandle of size 7>
gap> Elements( last );
[ r1, r2, r3, r4, r5, r6, r7 ]
```

## Congruences

- equivalance relations are implemented in GAP

```
gap> G := SymmetricGroup( 3 );;
gap> C := EquivalenceRelationByPartition( G, [[(),(1,2,3),(1,3,2)],[(1,2),(1,3),(2,3)]] );
<equivalence relation on SymmetricGroup( [ 1 .. 3 ] ) >
gap> Source( C );
Sym( [ 1 .. 3 ] )
gap> EquivalenceClasses( C );
[ {()}, {(1,2)} ]
gap> Elements( last[1] );
[ (), (1,2,3), (1,3,2) ]
```

## Congruence examples

- RQ can deal with congruences of right quasigroups

```
gap> Q := QuasigroupByFunction( [0..3], function(x,y) return (x-y) mod 4; end );;
gap> C := EquivalenceRelationByPartition( Q, [ [Q[0],Q[2]], [Q[1],Q[3]] ] );
<equivalence relation on <quasigroup of size 4 on 0, 1, 2, 3> >
gap> IsQuasigroupCongruence( C );
true
```

```
gap> Q := QuasigroupByFunction( GF(27), \- );;
gap> C := QuasigroupCongruenceByPartition( Q, [ [ Q.1, Q.2, Q.3 ], [ Q.4, Q.5 ] ] );;
gap> List( EquivalenceClasses( C ), Size );
[ 9, 9, 9 ]
```

# Factor algebras

- congruences are used to construct factor algebras

```
gap> Q := ProjectionRightQuasigroup( 6 );;
gap> C := EquivalenceRelationByPartition( Q, [[Q.1,Q.2],[Q.3,Q.4,Q.5],[Q.6]] );;
gap> [ IsRightQuasigroupCongruence( C ), IsQuasigroupCongruence( C ) ];
[ true, false ]
gap> F := Q/C;
<associative quandle of size 3>
gap> Elements( F );
[ r<object>, r<object>, r<object> ]
```

# Demonstration: Paige loops

- Paige loops are nonassociative finite simple Moufang loops
- Zorn matrix algebra over $F$ consists of matrices

$$x = \begin{pmatrix} a & \alpha \\ \beta & b \end{pmatrix}$$

with $a, b \in F$, $\alpha, \beta \in F^3$

- Norm is the "determinant" $N(x) = ab - \alpha \cdot \beta$
- Addition componentwise, multiplication by

$$\begin{pmatrix} a & \alpha \\ \beta & b \end{pmatrix} \begin{pmatrix} c & \gamma \\ \delta & d \end{pmatrix} = \begin{pmatrix} ac + \alpha \cdot \delta & a\gamma + d\alpha - \beta \times \delta \\ c\beta + b\delta + \alpha \times \gamma & \beta \cdot \gamma + bd \end{pmatrix}$$

- $S(F) = \{x : N(x) = 1\}$
- $\mathrm{Paige}(F) = S(F)/Z(S(F))$
- Paige proved that every $\mathrm{Paige}(GF(q))$ is a nonassociative finite simple Moufang loop
- Liebeck proved that there are no other

# Paige loops: Auxiliary functions

```
gap> DotProduct := function( x, y )
>  return Sum( [1..Length(x)], i -> x[i]*y[i] );
end;;
gap> CrossProduct := function( x, y )
>  return [ x[2]*y[3]-x[3]*y[2], x[3]*y[1]-x[1]*y[3], x[1]*y[2]-x[2]*y[1]];
end;;
gap> PaigeNorm := function( x )
>  return x[1]*x[8] - DotProduct( x{[2,3,4]},x{[5,6,7]} );
end;;
gap> PaigeMult := function( x, y )
>   local a, b, c, d;
>   a := x[1]*y[1] + DotProduct(x{[2,3,4]},y{[5,6,7]});
>   b := x[1]*y{[2,3,4]} + x{[2,3,4]}*y[8] - CrossProduct(x{[5,6,7]},y{[5,6,7]});
>   c := x{[5,6,7]}*y[1] + x[8]*y{[5,6,7]} + CrossProduct(x{[2,3,4]},y{[2,3,4]});
>   d := DotProduct(x{[5,6,7]},y{[2,3,4]})+x[8]*y[8];
>   return Concatenation( [a], b, c, [d] );
end;;
```

# Paige loops: Over GF(2), index based approach

```
gap> F := GF(2);;
gap> S := Filtered( F^8, x -> PaigeNorm( x ) = One( F ) );;
gap> P := LoopByFunction( S, PaigeMult, ConstructorStyle( true, true ) );
<loop of size 120>
```

- checking properties

```
gap> IsMoufangLoop( P );
true
gap> P;
<Moufang loop of size 120>
gap> IsAssociative( P );
false
gap> IsSimpleLoop( P );
true
```

# Paige loops: general approach using congruences

- Main idea: create a congruence "modulo $\pm 1$"

```
gap> n := 3;;
gap> F := GF(n);;
gap> S := Filtered( F^8, x -> PaigeNorm( x ) = One( F ) );;
gap> M := LoopByFunction( S, PaigeMult, ConstructorStyle( false, false ) );;
gap> C := EquivalenceRelationByPartition( M, Set( S, x -> Set( [ M[x], M[-x] ] ) ) );;
gap> P := FactorLoop( C, ConstructorStyle( false, false ) );
<loop of size 1080>
```

## Paige loops: general approach using normal subloops

- Main idea: Factor out the center consisting of $\pm 1$

```
gap> n := 3;; F := GF(n);;
gap> S := Filtered( F^8, x -> PaigeNorm( x ) = One( F ) );;
gap> M := LoopByFunction( S, PaigeMult, ConstructorStyle( false, false ) );;
gap> one := [ Z(n)^0, 0*Z(n), 0*Z(n), 0*Z(n), 0*Z(n), 0*Z(n), 0*Z(n), Z(n)^0 ];;
gap> N := Subloop( M, [-one] );;
gap> P := FactorLoop( M, N, ConstructorStyle( false, false ) );
<loop of size 1080>
```

# Paige loops: general approach using tricky multiplication

- Main idea: Split $F \setminus \{0\}$ into two parts $A$, $B$ that are bijective under $x \mapsto -x$. Take a subset of $S(F)$ with first nonzero entry in $A$. Multiply on this subset. If the product has first entry in $B$, remultiply by $-1$.

```
FirstNonzeroEntry := function( ls, F ) # works for F = {0,1,-1}
  local pos;
  pos := First( [1..Length(ls)], i -> ls[i]<>Zero(F) );
  if pos<>fail then return ls[pos]; else return fail; fi;
end;
```

```
NewPaigeMult := function( x, y ) # works for F = {0,1,-1}
  local z;
  z := PaigeMult( x, y );
  if FirstNonzeroEntry(z,F) <> One(F) then z := -z; fi;
  return z;
end;
```

## … tricky multiplication example continued

```
gap> n := 3;;
gap> F := GF( n );;
gap> S := Filtered( F^8, x ->
  PaigeNorm(x) = One( F ) and FirstNonzeroEntry(x,F) = One(F)
);;
gap> P := LoopByFunction( S, NewPaigeMult, ConstructorStyle( false, false ) );
<loop of size 1080>
```