Chapter 2: Constructors

2.1: Basic functionality

- Categories of objects for right quasigroups
- Dynamic display
- Accessing elements
- Arithmetic operations
- Built-in parser

Categories

- Declare a quasigroup

```
gap> Q := QuasigroupByCayleyTable( [[0,1],[1,0]] );
<quasigroup of size 2>
```

- It is in fact a group but GAP does not know this

```
gap> [ IsMagma( Q ), IsRightQuasigroup( Q ), IsQuasigroup( Q ) ]
[ true, true, true ]
gap> [ IsLoop( Q ), IsGroup( Q ) ];
[ false, false ]
```

- Detect the declared category

```
gap> CategoryOfRightQuasigroup( Q );
<Category "IsQuasigroup">
```

Dynamic display

- Displayed information changes on the fly depending on what is known

```
gap> Q := QuasigroupByCayleyTable( [[0,1],[1,0]] );
<quasigroup of size 2>
gap> IsAssociative( Q );
true
gap> Q;
<associative quasigroup of size 2>
```

- "Display" gives more information

```
gap> Display( Q );
<associative quasigroup of size 2 on 0, 1>
```

Displaying elements

- The underlying set

```
gap> Q := AsLoop( Group( (1,2) ) );
<associative loop of size 2>
gap> UnderlyingSet( Q );
[ (), (1,2) ]
gap> Elements( Q );
[ l(), l(1,2) ]
```

- Standard prefix for elements: "r", "q" or "l"
- Changing the prefix

```
gap> SetLoopElementsName( Q, "Loopy" );; Elements( Q );
[ Loopy(), Loopy(1,2) ]
```

Accessing elements

```
gap> Q := AsLoop( Group( (1,2) ) );;
```

- Using the index of the elements in the parent algebra (more later)

```
gap> Q.2;
l(1,2)
```

- Using the underlying set element

```
gap> Q[(1,2)];
l(1,2)
```

- Using "Elements"

```
gap> Elements(Q)[2];
l(1,2)
```

Binary arithmetic operations

- Multiplication

```
gap> Q := AsLoop( Group( (1,2) ) );;
gap> Q[()]*Q[(1,2)];
l(1,2)
gap> Q[(1,2)]*Q;
[ l(1,2), l() ]
```

- Right division

```
gap> Q.2/Q.2;;
gap> RightQuotient( Q.2, Q.2 );;
gap> RightDivision( Q.2, Q.2 );
l()
```

- Left division (operation symbol \ is not allowed)

```
gap> LeftQuotient( Q.2, Q.2 );;
gap> LeftDivision( Q.2, Q.2 );
l()
```

Unary and nullary operations (for loops only)

- Neutral element

```
gap> One( Q );;
```

- Left and right inverses

```
gap> RightInverse( Q[(1,2)] );;
gap> LeftInverse( Q[(1,2)] );;
```

- Two-sided inverse (if it exists)

```
gap> Inverse( Q[(1,2)] );
l(1,2)
```

A built-in parser

- There is a simple built-in parser in RightQuasigroups

```
gap> Q := AsLoop( CyclicGroup( 6 ) );;
gap> LoopSatisfiesIdentity( Q, "x*y=y*x" );
true
```

- Division symbols

```
gap> LoopSatisfiesIdentity( Q, "x|y=y/x" );; # | is for left division
```

- Can be used to find counterexamples

```
gap> LoopSatisfiesIdentity( Q, "x*x=1" );
[ [ 'x', lf1 ] ]
```

2.2 Index based vs. non-index based algebras

- Index based algebra
- Non-index based algebras
- Constructor style
- Conversions

Index based algebras

- Default
- Elements are indexed by a subset of $\{1, \ldots, n\}$
- The main object is the multiplication table
- All operations are based on the multiplication table
- Faster
- For algebras with up to several thousand elements
- The underlying set is cosmetic and can be changed

Index based example

```
gap> mult := function( x, y ) return (x+y) mod 6; end;
gap> Q := RightQuasigroupByFunction([0..5], mult );
<right quasigroup of size 6>
gap> Elements( Q );
[ r0, r1, r2, r3, r4, r5 ]
gap> A := Subrightquasigroup( Q, [2] );
<right quasigroup of size 3>
gap> Elements( A );
[ r0, r2, r4 ]
```

Parent indices

```
# recall A = [0,2,4] in [0..5]
gap> Parent( A ) = Q;
true
gap> ParentInd(A);
[ 1, 3, 5 ]
gap> [ Elements( A )[ 3 ], A.3, A[4] ];
[ r4, r2, r4 ]
```

Cayley table vs. multiplication table

```
gap> Display( CayleyTable( A ) ); # based on the underlying set
[ [  0,  2,  4 ],
  [  2,  4,  0 ],
  [  4,  0,  2 ] ]
gap> Display( MultiplicationTable( A ) ); # based on parent indices
[ [  1,  2,  3 ],
  [  2,  3,  1 ],
  [  3,  1,  2 ] ]
```

Non-index based algebras

- The main object is the multiplication function on the underlying set
- The underlying set cannot be (easily) changed
- Multiplication table not explicitly calculated
- Slower
- For algebras with up to million elements

Constructor style

- constructor style specifies if algebras are supposed to be index based or not, and if arguments are supposed to be checked
- constructor style can be used in most constructors

```
gap> ConstructorStyle( true, false );
rec( checkArguments := false, indexBased := true )
```

- default constructor style is: indexBased = true, checkArguments = false
- default constructor style can be changed for a given GAP session

```
gap> SetDefaultConstructorStyle( false, false );
true
```

Non index based example

```
gap> ncs := ConstructorStyle( false, false );;
gap> P := ProjectionRightQuasigroup( [0..100000], ncs );  # x*y = x
<associative quandle of size 100001>
gap> P[5]*P[100000];
r5;
gap> HasMultiplicationTable( P );
false
gap> mult := MultiplicationFunction( P );
function( x, y ) ... end
gap> mult(5,100000);
5
```

Another non-index based example (shows inner workings)

```
gap> Q := RightQuasigroupByFunction( GF( 9 ), \+, ncs );
<right quasigroup of size 9>
gap> IsIndexBased( Q );
false
gap> F := FamilyObj( Q.1 ); # will provide access to inner workings
<Family: "RightQuasigroupFam">
gap> [ IsBound( F!.mult ), IsBound( F!.rdiv ), IsBound( F!.ldiv ) ];
[ true, true, false ]
gap> [ IsBound( F!.multTable ), IsBound( F!.rdivTable ), IsBound( F!.ldivTable ) ];
[ false, false, false ]
gap> MultiplicationFunction( Q );
<Operation "+">
```

- The constructor does not know that the right divison is <Operation "-">

```
gap> RightDivisionFunction( Q );
function( x, y ) ... end
```

Conversion from non-index based to index based

```
gap> Q := RightQuasigroupByFunction( GF( 9 ), \+, ncs );;
gap> R := IndexBasedCopy( Q );; IsIndexBased( R );
true
```

- No change to the underlying set

```
gap> UnderlyingSet( R );
[ 0*Z(3), Z(3)^0, Z(3), Z(3^2), Z(3^2)^2, Z(3^2)^3, Z(3^2)^5, Z(3^2)^6, Z(3^2)^7 ]
```

- What is bound and when

```
gap> F := FamilyObj( R.1 );;
gap> [ IsBound( F!.mult ), IsBound( F!.rdiv ), IsBound( F!.ldiv ) ];
[ true, true, false ]
gap> [ IsBound( F!.multTable ), IsBound( F!.rdivTable ), IsBound( F!.ldivTable ) ];
[ true, false, false ]
gap> R.1/R.1;;
gap> [ IsBound( F!.multTable ), IsBound( F!.rdivTable ), IsBound( F!.ldivTable ) ];
[ true, true, false ]
```

Canonical form

- Index based
- Underlying set is $\{1, \ldots, n\}$

```
gap> Q := RightQuasigroupByFunction( GF( 9 ), \+, ncs );;
gap> R := IndexBasedCopy( Q );;
gap> IsCanonical( R );
false
gap> C := CanonicalCopy( Q );;
gap> UnderlyingSet( C );
[ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

2.3: Some constructors

- Conversions between magmas
- By Cayley table
- By function

(more later)

Conversion between magmas

- upgrading

```
gap> M := MagmaByMultiplicationTable( [ [1,1], [2,2] ] );;
gap> IsRightQuasigroupMagma( M ); # test of math properties
true
gap> R := AsRightQuasigroup( M );
<right quasigroup of size 2>
```

- downgrading

```
gap> G := Group((1,2,3));;
gap> L := AsLoop( G ); # multiplicative group
<associative loop of size 3>
gap> Elements( L );
[ l(), l(1,2,3), l(1,3,2) ]
gap> AsRightQuasigroup(GF(7)^2); # additive group
<associative right quasigroup of size 49>
```

By Cayley table

```
gap> ct := [
  [ "red", "white", "white" ],
  [ "blue", "blue", "red" ],
  [ "white", "red", "blue" ] ];;
gap> Q := RightQuasigroupByCayleyTable( ct );
<right quasigroup of size 3>
```

- Note the ordering of elements induced by "blue" < "red" < "white"

```
gap> Elements( Q );
[ rblue, rred, rwhite ]
gap> PrintArray( CayleyTable( Q ) );
[ [    red,  white,  white ],
  [   blue,   blue,    red ],
  [  white,    red,   blue ] ]
```

By function

- at least the multiplication function must be provided
- other functions are optional (improve performance when given)

```
gap> S := GF(5);;
gap> mult := function( x, y ) return x+2*y; end;;
gap> IsQuasigroupFunction( S, mult );
true
gap> QuasigroupByFunction( S, mult );
<quasigroup of size 5>
gap> rdiv := function( x, y ) return x-2*y; end;;
gap> ldiv := function( x, y ) return (y-x)/2; end;;
gap> IsQuasigroupFunction( S, mult, rdiv, ldiv );
true
gap> QuasigroupByFunction( S, mult, rdiv, ldiv );
<quasigroup of size 5>
```