Bachelor's Thesis Computing Science

Radboud University Nijmegen

QuandleRUN

Computational quandle theory in Magma

Author:
Niccolò Carrivale
s1035576

First supervisor/assessor:
Dr. Wieb Bosma

Second assessor:
Dr. Michael Müger

July 6, 2022

Abstract

Quandles are algebraic structures that arise in the fields of knot theory, statistical mechanics, and mathematical chemistry; other than being interesting algebraic objects. Quandles can therefore benefit from the help of computational tools, much like groups. This thesis introduces QuandleRUN, a modern tool dedicated to the study of quandles built on the solid mathematical foundations of the most recent developments in Quandle Theory, and Universal Algebra. QuandleRUN is built as a module for the computer algebra system Magma and it aims to improve Rig, the only package explicitly dedicated to quandles. By building on tools such as Magma and Rig that have withstood the test of time, and taking inspiration from modern tools such as CREAM, QuandleRUN offers algorithms to compute the key structures of any algebraic object such as the automorphism group, the set of subalgebras, the set of congruences and it also allows to find monomorphisms and isomorphisms between quandles, if they exist. This thesis also introduces a novel algorithm to compute the set of congruences of a quandle. The powerful underlying system, Magma, as well as a careful implementation of the algorithms allow QuandleRUN to outperform both Rig and CREAM, in most cases.

Contents

Chapter 1

Introduction

Quandles are relatively modern algebraic structures. They were independently described by David Joyce in [23] and Sergei V. Matveev in [29]. In the field of knot theory, the quandle axioms are equivalent to the three Reidemeister moves, in the context of Tietze transformations [13, 31]. In [23], David Joyce indicates quandles as a classifying invariant of knots which means that they play a role in what still is one of the fundamental problems of knot theory, the classification of knots – a computationally hard problem. Virtual knot theory, introduced by Louis H. Kauffman in [26] generalises classical knot theory to the point that many structures in classical knot theory generalize to the virtual domain; in fact, one can extend quandles to virtual quandles that yield generalizations of several invariants of traditional knots [24].

In the quest to solve the knot classification problem, knot invariants related to statistical mechanics arose [22, 25, 37]. A central theme of statistical mechanics is the Yang-Baxter equation [40], of which quandles arise as set-theoretical solutions [16]. Quandles also arise in connection to Hopf algebras and Nichols algebras [2] and as a useful tool in studying the relation between a Lie Algebra and its Lie Group [13]. Quandles can also be adapted to distinguish the topological types of proteins, thus obtaining *bondles* [1].

This motivates the efforts in the study of quandles and the creation of computational tools that can help in such an endeavour. At the time of writing, Rig [38], a package for the computer algebra system GAP [27], seems to be the only tool explicitly dedicated to the study quandles.

Computer algebra involves developing algorithms to carry out *exact* computations involving algebraic structures, such as groups, graphs and, algebraic curves which play an important role in modern mathematics. Contrary to results of numerical methods, which concern themselves with approximate solutions, computer algebraic procedures can help in solving decision problems such as *Is a defined quandle Q connected?* or, *Is quandle Q' a subalgebra of quandle Q?*.

This work is interested in developing solid foundations on which future computational work on quandles can be built and does so by combining results from the mathematical theory of quandles and adapting existing computer algebraic methods. QuandleRUN, a module for the computer algebra system Magma [11], aims to improve Rig and become a real research instrument. QuandleRUN draws inspiration from Rig itself and from CREAM [14], a very recent and versatile package for the computer algebra system GAP, that works with any unary and/or binary algebra.

QuandleRUN builds on some of their techniques and introduces new algorithms based on the most recent developments in the theory of quandles. In this way, it can achieve a competitive speed in the most important tasks. QuandleRUN outperforms Rig in every task and CREAM in most tasks, which means that it has the potential to be the go-to tool of future quandle theorists. All of QuandleRUN's algorithms directly work, or can be easily adapted to work, on left quasigroups.

This is a self-contained introduction to QuandleRUN. In Section 2, there is a short presentation of the underlying mathematics. The concepts of left quasigroup, subalgebras, quotient structures, and homomorphisms will accompany the reader throughout the entire document, the interested reader can find out more in [12, 15, 36]. Section 3 describes the computer representation of quandles and the algorithms to compute informative groups, subalgebras, congruences, and quotient algebras. Section 4 shows some practical capabilities of QuandleRUN. Section 5 presents a brief comparison in terms of running times of QuandleRUN, Rig, and CREAM.

Chapter 2

Preliminaries

For any non-empty set S, one can define particular functions known as *operations*. A function $f : S \to S$ is known as a *unary* while a *binary* operation on S is a function $f : S \times S \to S$.
This work will only discuss *finite* binary algebras, that is finite sets endowed with binary operations. This is convenient because binary operations on finite sets can be represented by their *Cayley table*.
Define $S = \{x_1, x_2, \ldots, x_n\}$ and $\cdot$ a binary operation on S.
Let $(S, \cdot)$ denote the set S endowed with the binary operation $\cdot$.
The following is the Cayley table of $(S, \cdot)$:

$\cdot$	x_1	x_2	$\ldots$	x_n
x_1	$x_1 \cdot x_1$	$x_1 \cdot x_2$	$\cdots$	$x_1 \cdot x_n$
x_2	$x_2 \cdot x_1$	$x_2 \cdot x_2$	$\cdots$	$x_2 \cdot x_n$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
x_n	$x_n \cdot x_1$	$x_n \cdot x_2$	$\cdots$	$x_n \cdot x_n$

Definition 2.1.

a. Let S be a non-empty set endowed with two binary operations $\cdot$ and $\backslash$ such that

$$\forall x, y \in S \quad x \cdot (x \backslash y) = y = x \backslash (x \cdot y)$$

then $(S, \cdot, \backslash)$ is called a *left quasigroup*.

b. Let $(S, \cdot, \backslash)$ be a left quasigroup. If

$$\forall x, y, z \in S \quad x \cdot (y \cdot z) = (x \cdot y) \cdot (x \cdot z) \qquad \text{(left distributivity of } \cdot\text{)},$$

then $(S, \cdot, \backslash)$ is called a *rack*.

c. Let $(S, \cdot, \backslash)$ be a rack. If

$$\forall x \in S \quad x \cdot x = x \qquad \text{(idempotency of } \cdot\text{)},$$

then $(S, \cdot, \backslash)$ is called a *quandle*.

It is appropriate to describe the idea of *(left-)*translation map L_x, defined as $L_x(y) = x \cdot y$ for any elements x, y of a left quasigroup $\mathbf{A} = (S, \cdot, \backslash)$. Left translation maps are of primary importance in the theory to be discussed. They are bijections. Definition 2.1a suggests $y = x\backslash z$ as *a* solution, whenever $x, z \in S$ are given. Definition 2.1a also excludes the existence of any other solution t: $t = x\backslash(x \cdot t) = x\backslash z = y$. This gives insight into how division ($\backslash$) is defined for a left quasigroup $(S, \cdot, \backslash)$:

$$x\backslash y = L_x^{-1}(y) \quad \text{for all } x, y \in S$$

Left translation maps are the generators of the *Left multiplication group*
$\mathrm{LMlt}(\mathbf{A}) = \langle L_x : x \in \mathbf{A} \rangle$. It will play a fundamental role throughout this work because it is involved in several computations *and* allows to distinguish *connected* left quasigroups: a left quasigroup $\mathbf{A}$ is called *connected* if the action of its Left multiplication group is transitive. The class of quandles involved with classifying invariants of knots mentioned in Section 1 is that of connected quandles; connectedness is also an algebraically relevant property according to the characterization of Mal'cev varieties of left quasigroups [6].
The definition of left division given above makes it possible to cut down the symbols used, thus any left quasigroup $(S, \cdot, \backslash)$ can be denoted by the pair $(S, \cdot)$ [6, 8].
This does not affect the structure of *finite* left quasigroups.

Example 2.2. [4, 23, 28]
Dihedral quandle of order n. The set $\{1, 2, \ldots, n\}$ associated with the operation $x \cdot y = 2x - y \bmod n$ form a quandle of order n denoted by $\mathbf{R_n}$.
The Cayley table of $\mathbf{R_3} = (\{1, 2, 3\}, \cdot)$ is

$\cdot$	1	2	3
1	1	3	2
2	3	2	1
3	2	1	3

Trivial quandle of order n. The set $\{1, 2, \ldots, n\}$ with the operation $x \cdot y = y$ form a quandle of order n denoted by $\mathbf{T_n}$.
The Cayley table of $\mathbf{T_3} = (\{1, 2, 3\}, \cdot)$ is

$\cdot$	1	2	3
1	1	2	3
2	1	2	3
3	1	2	3

Other constructions, perhaps more used by mathematicians, arise from groups.

Let G be a group.

Core Quandle. The underlying set of G and the operation $x \cdot y = xy^{-1}x$ form a quandle, denoted by $\mathrm{Core}(G)$.

n-Conjugation Quandle. Let $n \in \mathbb{N}$. The underlying set of G and the operation $x \cdot y = x^n y x^{-n}$ form a quandle denoted by $\mathrm{Conj}_n(G)$.

Coset Quandle. Let $f \in \mathrm{Aut}(G)$ and $H \leq G$ such that $\forall x \in H \quad f(x) = x$.
The left coset G/H and the operation $xH \cdot yH = xf(x^{-1}y)H$ form a quandle, denoted by $\mathcal{Q}_{\mathrm{Hom}}(G, H, f)$.

The definition of a few of the most fundamental notions in the field of universal algebra, subalgebras, quotient structures and homomorphisms, will follow. Given the generality of the matter, the interested reader can refer to [12], for all undefined notions.

Definition 2.3. [12]
Let $\mathbf{A} = (S, \cdot, \backslash)$ be a left quasigroup. Let $S' \subseteq S$. When

$$\forall x, y \in S' \quad x \cdot y \in S' \quad \wedge \quad x \backslash y \in S',$$

that is, when S' is closed under the operations of $\mathbf{A}$, it can be associated with the restriction of the operations of $\mathbf{A}$ to S' to form $\mathbf{B} = (S', \cdot_{|S'}, \backslash_{|S'})$. Consequently, $\mathbf{B}$ is called a *subalgebra* of $\mathbf{A}$.
In general, any non-empty finite subset X of S can be expanded to be a subalgebra of $\mathbf{A}$:

$$\mathrm{Sg}(X) = \bigcap \{U : X \subseteq U \wedge U \text{ is closed under the operations of } \mathbf{A}\}$$

$(\mathrm{Sg}(X), \cdot_{|\,\mathrm{Sg}(X)}, \backslash_{|\,\mathrm{Sg}(X)})$ is a subalgebra of $\mathbf{A}$.
The set of all subalgebras of a left quasigroup $\mathbf{A}$ is denoted by $\mathrm{Sub}(\mathbf{A})$.
If, for a set $\emptyset \neq X \subseteq S$, $\mathrm{Sg}(X) = S$ then X is a generating set of $\mathbf{A}$ and its elements are called *generators*.

Given a non-empty set S, one can define a *relation* $\sim \subseteq S \times S$ on S; for two elements $x, y \in S$, $(x, y) \in \sim$ is often denoted by $x \sim y$.
Relations can have several properties. S relation is called *reflexive* when for any element $x \in S \quad (x, x) \in \ \sim$; *symmetric* when $(x, y) \in \ \sim$ implies that $(y, x) \in \ \sim$, for any two elements $x, y \in S$; or, *transitive* when the fact that (x, y) and (y, z) are both elements of $\sim$ implies that (x, z) is also an element of $\sim$. When a relation is reflexive, symmetric, and transitive, then it is called an *equivalence relation*.
Such a relation $\sim$ on a non-empty set S is particularly important because it induces a partition, usually denoted by $S/\sim$, that is a disjoint set of sets (called *equivalence classes*) such that their union is equal to S. Equivalence classes are also called *blocks of the partition* and are denoted by $[x]_\sim = \{y \in S : x \sim y\}$, for an element $x \in S$. It is possible to order the partitions of a set: for two partitions π_1, π_2 of some non-empty set S, one says $\pi_1 \leq \pi_2$ when each block of π_1 is contained in some block of π_2.

Equivalence relations that respect the operations of a left quasigroup are called *congruences* and are defined next.

Definition 2.4. [12]
Let $\mathbf{A} = (S, \cdot)$ be a left quasigroup and $\sim$ an equivalence relation on S. If

$$\forall x, y, z, t \in S \quad x \sim z \wedge y \sim t \implies x \cdot y \sim z \cdot t \quad \wedge \quad x \backslash y \sim z \backslash t,$$

that is, $\sim$ has the *compatibility property*, then $\sim$ is a *congruence relation* on $\mathbf{A}$.
The set of all congruences on an algebra $\mathbf{A}$ is denoted by $\mathrm{Con}(\mathbf{A})$.

For quandles, the blocks of the partitions induced by congruence relations, associated with the restriction to them of the operations, are subalgebras. The definiton of congruences allows the definition of a few important groups.

Definition 2.5. [9]
Let $\sim$ be an equivalence relation on a left quasigroup $\mathbf{A} = (S, \cdot)$.

- *Displacement group*, $\mathrm{Dis}(\mathbf{A})$: $\langle L_x L_y^{-1} : x, y \in \mathbf{A} \rangle$
- *Kernel relative to* $\sim$, $\mathrm{Dis}^{\sim}$: $\{h \in \mathrm{Dis}(\mathbf{A}) : h(x) \sim x \text{ for every } x \in \mathbf{A}\}$
- *Displacement group relative to* $\sim$, $\mathrm{Dis}_{\sim}(\mathbf{A})$: $\langle \{h L_x L_y^{-1} h^{-1} : x \sim y \ \wedge \ h \in \mathrm{LMlt}(\mathbf{A})\} \rangle$; in particular, if $\mathbf{A}$ is a rack, then $\mathrm{Dis}_{\sim}(\mathbf{A}) = \langle L_x L_y^{-1} : x \sim y \rangle$.

Note: $\mathrm{Dis}(\mathbf{A}) = \mathrm{Dis}_{\sim}(\mathbf{A})$ when $\sim \ = S \times S$.

It is worth pointing out that, for quandles, the actions of $\mathrm{Dis}(\mathbf{A})$ and $\mathrm{LMlt}(\mathbf{A})$ have the same orbits [9].

One can establish an algebraic structure on the set of equivalence classes obtained by partitioning the underlying set of an an algebra according to a congruence relation. This idea leads to the following definition of quotient structures.

Definition 2.6. [12]
Let $\mathbf{A} = (S, \cdot, \backslash)$ be a left quasigroup and $\sim$ a congruence relation on $\mathbf{A}$.
Let $\mathbf{B} = (S/\sim, \star, \backslash_\star)$ where

$$[x]_\sim \star [y]_\sim = [x \cdot y]_\sim, \quad \text{for any } [x]_\sim, [y]_\sim \in S/\sim$$

$$[x]_\sim \backslash_\star [y]_\sim = [x \backslash y]_\sim, \quad \text{for any } [x]_\sim, [y]_\sim \in S/\sim$$

$\mathbf{B}$ is a *quotient left quasigroup* of $\mathbf{A}$.

Quotient structures are important in their own right. In some cases, they can be used to obtain information about a superstructure and lead to the classification of the entire family of a structure. For instance, the interested reader can refer to Section 4 of [7] to see how this strategy played a fundamental role in classifying non-simple connected quandles of size pq where p and q are different prime numbers.

The definition of *homomorphisms*, particular functions that preserve the operation of the left quasigroup on which they are defined, will follow.

Definition 2.7. [12]

- Let $\mathbf{A} = (S, \cdot)$ and $\mathbf{B} = (S', \star)$ be two left quasigroups.
  Let $\phi : \mathbf{A} \to \mathbf{B}$.
  If
  $$\forall x, y \in \mathbf{A} \quad \phi(x \cdot y) = \phi(x) \star \phi(y)$$
  then ϕ is a *homomorphism*.

- Let $\phi : \mathbf{A} \to \mathbf{B}$ be a homomorphism. If ϕ is injective, ϕ is called a *monomorphism*, or an *embedding* of $\mathbf{A}$ into $\mathbf{B}$. The image of ϕ is a *subalgebra* of $\mathbf{B}$ denoted by $\phi(\mathbf{A}) \leq \mathbf{B}$.

- Let $\phi : \mathbf{A} \to \mathbf{B}$ be a homomorphism. If ϕ is surjective, ϕ is called an *epimorphism*.

- Let $\phi : \mathbf{A} \to \mathbf{B}$ be a homomorphism. If ϕ is injective and surjective, that is, ϕ is bijective, ϕ is called an *isomorphism*.
  If there exists an isomorphism $\phi : \mathbf{A} \to \mathbf{B}$, it is indicated $\mathbf{A} \cong \mathbf{B}$.
  In case $\mathbf{A} = \mathbf{B}$, ϕ is called an *automorphism*.
  The set of all automorphisms on a left quasigroup $\mathbf{A}$ is a subgroup of the symmetric group on the underlying set of $\mathbf{A}$ and it is denoted by $\mathrm{Aut}(\mathbf{A})$ [39].

There are two strong connections between homomorphisms and congruences on left quasigroups. Every homomorphism $h : \mathbf{A} \to \mathbf{B}$ induces a congruence, called *kernel*, on its domain, defined as $\ker(h) = \{(x, y) : f(x) = f(y)\}$.

As explained above, congruence relations induce a partition of the set over which they are defined. There exists a function, called *canonical projection*, that maps an element to its equivalence class relative to the congruence. The canonical projection is also an epimorphism from the original left quasigroup $\mathbf{A}$ to its quotient structure given a congruence $\sim$, $\mathbf{A}/\sim$.

Throughout the document, and especially when dealing with homomorphisms, there will be references to so-called *invariants* under homomorphism. Contrary to what the word might suggest and to what was intended in Section 1, these *invariants* can change, but in a "predictable" way.
It is possible to distinguish two types of invariants: *global* or *element-wise*.

Global invariants.
Let p be an operator that assigns a natural number to any left quasigroup. The operator p is called a *global invariant* if $p(\mathbf{A}) \leq p(\mathbf{B})$ whenever $\mathbf{A}$ can be embedded into $\mathbf{B}$. For example:
Let $\mathrm{Nil}(G)$ indicate the nilpotency class of a group G and let $\mathbf{A}$ and $\mathbf{B}$ be two left quasigroups such that there exists a monomorphism $h : \mathbf{A} \to \mathbf{B}$.
Then $\mathrm{Nil}(\mathrm{LMlt}(\mathbf{A})) \leq \mathrm{Nil}(\mathrm{LMlt}(\mathbf{B}))$. A similar fact holds for the displacement group of the left quasigroups: let $\mathbf{A}$ and $\mathbf{B}$ be two left quasigroups such that there exists a monomorphism $h : \mathbf{A} \to \mathbf{B}$, then $\mathrm{Nil}(\mathrm{Dis}(\mathbf{A})) \leq \mathrm{Nil}(\mathrm{Dis}(\mathbf{B}))$.
Restricting the focus to quandles and isomorphisms, there also exist polynomial invariants of finite quandles such as the one presented in [32] by Sam Nelson which can help in ruling out the existence of an isomorphism between two quandles $\mathbf{A}$ and $\mathbf{B}$.

Element-wise invariants.
Let p be an operator that assigns a map $p_{\mathbf{A}} : \mathbf{A} \to \mathbb{N}$ to any left qusigroup $\mathbf{A}$.
If for every monomorphism $h : \mathbf{A} \to \mathbf{B}$ and every $x \in \mathbf{A}$ it holds that $p_{\mathbf{A}}(x) \leq p_{\mathbf{B}}(h(x))$ then p is called *element-wise* invariant.
It is worth showing two examples of what these invariants might look like for an element x of a left quasigroup $\mathbf{A} = (S, \cdot, \backslash)$:

- Number of elements $y \in S$ such that $L_x(y) = y$.
- Number of elements $y \in S$ such that $L_y(x) = x$.

When, for an element x of a left quasigroup $\mathbf{A}$, an element-wise invariant p and a left quasigroup $\mathbf{B}$, there exists no element $y \in \mathbf{B}$ such that $p(x) \leq p(y)$ then the existence of a monomorphism between $\mathbf{A}$ and $\mathbf{B}$ can be ruled out.

Chapter 3

QuandleRUN

3.1 Algorithms

The first step in manipulating something using a computer is the creation of a suitable representation: similar to the idea of adjancency matrices to represent graphs. In [19], Benita Ho and Sam Nelson introduce the idea of *quandle matrix*, that is, essentially, the Cayley table (the entry in row i and column j is $x_i \cdot x_j$): Let $\mathbf{A} = (\{x_1, x_2, \ldots, x_n\}, \cdot)$ be a finite quandle with n elements.
The quandle matrix $M_{\mathbf{A}}$ of $\mathbf{A}$ is:

$$M_{\mathbf{A}} = \begin{bmatrix} x_1 \cdot x_1 & x_1 \cdot x_2 & \ldots & x_1 \cdot x_n \\ x_2 \cdot x_1 & x_2 \cdot x_2 & \ldots & x_2 \cdot x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n \cdot x_1 & x_n \cdot x_2 & \ldots & x_n \cdot x_n \end{bmatrix}$$

These ideas can *easily* be generalised to racks and left quasigroups, so much that several of the algorithms described in this document work on all the algebraic structure defined in Chapter 2.

Without loss of generality, one can restrict discussions about quandle matrix to *integral quandle matrix*, that is, the quandle matrix of a quandle $\mathbf{A}$ with underlying set $S = \{1, 2, \ldots, n\}$ and $M_{\mathbf{A}} = \left[m_{ij}\right]$. Given the quandle matrix of any quandle $\mathbf{A} = (\{x_1, x_2, \ldots, x_n\}, \cdot)$, one can obtain an integral quandle matrix by mapping x_i to i, for $1 \leq i \leq n$ [19]. This allows QuandleRUN to have an internal representation independent of what the actual elements of the quandles might be – hence, if it makes the definition clearer, some algorithms might be described only in their "integral" form. In [19], Ho et al. determine clear methods to verify whether a given quandle matrix actually represents a quandle. However, this also takes some flexibility away from the scholar.

QuandleRUN, inspired by the treatment given by Magma to permutation groups, is the first tool trying to relax these constraints and it amortizes any additional cost that might come out of this by following the suggestion of [20]:

> *"[…] the first and principal aim before doing anything else […] is […] to move into the best possible computational situation* [Author's Note: read *representation*] *[…]."*

To do this, QuandleRUN maintains a bijective mapping from the underlying set S of the left quasigroup to $\{1..|S|\}$, called *Numbering Map* and denoted by $\mathcal{N}_S$, this was inspired by MAGMA.

3.1.1 Construction and verification of quandle axioms

Creation of a quandle

There are several known quandle constructions available in QuandleRUN and even more can easily be independently developed given the available framework. The essential components of any finite quandle are the underlying *indexed* set, the quandle operation and the bijective mapping from the underlying set and the set $\{1..n\}$, where n is the cardinality of the underlying set.
A way to create a quandle $\mathbf{A} = (S, \cdot)$ is by providing QuandleRUN with the quandle matrix $M_{\mathbf{A}}$ and the indexed set of labels.

Example 3.1.

```
QuandleFM([[7,9,8],[9,8,7], [8,7,9]], {@ 7, 8, 9 @});
// {@ 7, 8, 9 @} is the indexed set of labels the user has chosen.
// Internally, this is still represented in terms of {1, 2, 3}.
```

Another way to create a quandle $\mathbf{A} = (S, \cdot)$ is by providing QuandleRUN with an indexed version of S and an appropriate operation on S. Such a function does not seem to be available in any existing tool.

Example 3.2.

```
S := {@ "a", "b", "c" @};
M := map< car<S,S> -> S | [<<"a", "a">, "a">, <<"a", "b">, "c">, <<"a","c">,"
    b">, <<"b","a">, "c">, <<"b","b">, "b">, <<"b", "c">, "a">, <<"c", "a">,
    "b"> >, <<"c","b">, "a">, <<"c","c">, "c">]>;
M;
Mapping from: Cartesian Product<{@ a, b, c @}, {@ a, b, c @}> to SetIndx: S
    <<"a", "a">, "a">
    <<"a", "b">, "c">
    <<"a", "c">, "b">
    <<"b", "a">, "c">
    <<"b", "b">, "b">
    <<"b", "c">, "a">
    <<"c", "a">, "b">
    <<"c", "b">, "a">
    <<"c", "c">, "c">
Quandle(S,M);
[
    [ 1, 3, 2 ]
    [ 3, 2, 1 ]
    [ 2, 1, 3 ]
]
```

The quandle constructions outlined in Example 2.2 are all available in QuandleRUN.

Example 3.3.

```
R_3 := DihedralQuandle({@ 1, 2, 3 @});
T_3 := TrivialQuandle({@ 1, 2, 3 @});
G := AlternatingGroup(4);
CoreQuandleG := CoreQuandle(G);
Conj3GQuandle := ConjugationQuandle(G, 3);
AutG := AutomorphismGroup(G);
// The underlying set of the Automorphism Group of a group is not
// an iterable object (for example a Sequence), hence some
// elements need to be generated "manually".
AutSet := [x*y : x,y in Generators(AutG)];
// This creates a list of functions
// that suit the needs of the example
Fs := [f : f in AutSet | #sub<G |{ (g*f(g)^-1) : g in G }> eq #G];
CQuandle := QuandleMatrix(CosetQuandle(G, sub<G|G.0>,Fs[1]));
CQuandle;
[
    [ 1, 11, 10, 8, 9, 7, 6, 4, 5, 3, 2, 12 ],
    [ 12, 2, 8, 10, 7, 9, 5, 3, 6, 4, 11, 1 ],
    [ 10, 8, 3, 5, 4, 6, 12, 2, 11, 1, 9, 7 ],
    [ 9, 7, 6, 4, 5, 3, 2, 12, 1, 11, 10, 8 ],
    [ 9, 7, 6, 4, 5, 3, 2, 12, 1, 11, 10, 8 ],
    [ 10, 8, 3, 5, 4, 6, 12, 2, 11, 1, 9, 7 ],
    [ 3, 5, 1, 11, 2, 12, 7, 9, 8, 10, 4, 6 ],
    [ 5, 3, 2, 12, 1, 11, 10, 8, 9, 7, 6, 4 ],
    [ 5, 3, 2, 12, 1, 11, 10, 8, 9, 7, 6, 4 ],
    [ 3, 5, 1, 11, 2, 12, 7, 9, 8, 10, 4, 6 ],
    [ 12, 2, 8, 10, 7, 9, 5, 3, 6, 4, 11, 1 ],
    [ 1, 11, 10, 8, 9, 7, 6, 4, 5, 3, 2, 12 ]
]

// The construction above allows to create connected quandles
// with as many elements as the group G.
```

The construction of Coset Quandles is not cheap. QuandleRUN implements an even slower version due to technical constraints in the underlying computer algebra system, circumventable but outside of the scope of this thesis.
It follows the ideal way to implement the construction of a coset quandle:

Algorithm 1: Coset Quandle Construction

Input : Group G, $f \in \mathrm{Aut}(G)$, $H \leq G$ such that $\forall x \in H \quad f(x) = x$;
Output: $\mathcal{Q}_{\mathrm{Hom}}(G, H, f)$

`/* LeftTransversal(`G, H`) returns an indexed set of elements`
$L = (l_i)_{i \in \{1,\dots,n\}}$ `forming a left transversal for G over H; and the corresponding transversal mapping` $\phi : G \to L \quad g \mapsto l_i$ `such that` $g \in l_i H$`, for any` $g \in G$`. */`
$L, \phi \leftarrow$ `LeftTransversal(`G, H`)`
`/* Define the quandle operation by its functional graph` $G(\cdot)$ `*/`
$G(\cdot) \leftarrow \{((x, y), \phi(xf(x^{-1}y))) : x, y \in L\}$
Return: $\mathbf{A} = (L, \cdot)$

Verification of the quandle axioms

Before returning a quandle to the user, QuandleRUN checks whether the construction actually represents a quandle. Let $\mathbf{A} = (S, \cdot)$ be a quandle.

Axiom 1: $\forall x \in S \quad x \cdot x = x$

Algorithm 2: Verify **Axiom 1**

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$;
An indexed set $S = (x_i)_{i \in \{1..n\}} \subsetneq \mathbb{Z}$ of labels.
Output: **True** or **False**

for $i \leftarrow 1$ **to** n **do**
  if $M_{ii} \neq x_i$ **then**
    Return: False
  end
end
Return: True

Axiom 2: $\forall x \in S \quad L_x$ is a bijection

Algorithm 3: Verify **Axiom 2**

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$;
An indexed set $S = (x_i)_{i \in \{1..n\}} \subsetneq \mathbb{Z}$ of labels.
Output: **True** or **False**

for $i \leftarrow 1$ **to** n **do**
  if $M_{i\star}$ *is not a permutation of* S **then**
    Return: False
  end
end
Return: True

Axiom 3: $\forall x, y, z \in S \quad x \cdot (y \cdot z) = (x \cdot y) \cdot (x \cdot z)$

Algorithm 4: Verify **Axiom 3**

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$;
An indexed set $S = (x_i)_{i \in \{1..n\}} \subsetneq \mathbb{Z}$ of labels.
Output: **True** or **False**

$G(\mathcal{N}_S) \leftarrow \{(x_i, i) : i \in \{1..n\}\}$
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    for $t \leftarrow 1$ **to** n **do**
      if $M_{i\mathcal{N}_S(M_{jt})} \neq M_{\mathcal{N}_S(M_{ij})\mathcal{N}_S(M_{it})}$ **then**
        Return: False
      end
    end
  end
end
Return: True

It is possible to condense these three algorithms into one to save some time and computing power; here they are displayed separately with clarity in mind. Furthermore, one might decide to use only a subset of these algorithms to focus on left quasigroup and/or racks.

3.1.2 Informative groups

A left quasigroup homomorphism is a map between two left quasigroups that preserves the structure of their binary operations. Hence, it is fundamentally important to have methods that allow researchers to find these connections since it might lead to more information about the structure they started with. In some specific cases, certain sets of bijective homomorphisms coupled with function composition give rise to groups that convey important information about the left quasigroups themselves. QuandleRUN has algorithms to compute some of the most informative groups such as the left multiplication group, the automorphism group, and more.

The *left multiplication group* described in Section 2.

Algorithm 5: Left Multiplication Group Construction

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ representing a left quasigroup $\mathbf{A}$
Output: LMlt($\mathbf{A}$)

`/* The left translations of the left quasigroup are the generators of its left multiplication group. */`
$\mathsf{L} \leftarrow \{M_{i\star} : i \in [1..|M|]\}$
`/* Subgroup(`x, y`) returns the subgroup generated by` y `within` x`, thus` $y \subseteq x$ `*/`
Return: `Subgroup(`$S_{(m_{ii})_{\{1..n\}}}, \mathsf{L}$`)`

There are a few different strategies to find all the automorphisms of a left quasigroup Q. The following notions can be found in [19] referring only to quandles but they can be easily generalised to *any* binary algebra as explained in Chapter 2 of [15].

Definition 3.4 (**Definition 1** in [19])**.**
Let $\mathbf{A} = (\{x_1, x_2, \ldots, x_n\}, \cdot)$ be a left quasigroup with M its matrix, and $\sigma \in S_n$. Define

$$\sigma(M) = P_\sigma^{-1} \sigma(m_{ij}) P_\sigma$$

where P is the permutation matrix relative to σ and $\sigma(m_{ij})$ indicates the matrix arising from replacing each entry m_{ij} of M by $\sigma(m_{ij})$.

The proof of the following two statements does not rely on the axioms of racks and quandles, thus the curious reader can refer directly to [19].

Theorem 3.5 (**Theorem 4** in [19])**.**
Two matrices $M_{\mathbf{A}}, M_{\mathbf{B}}$ determine isomorphic left quasigroups of order n if and only if there exists a permutation $\sigma \in S_n$ such that $\sigma(M_{\mathbf{A}}) = M_{\mathbf{B}}$

Corollary 3.6 (**Corollary 5** in [19])**.**
The automorphism group of a finite left quasigroup $\mathbf{A}$ *of order* n *is isomorphic to the subgroup of* S_n *that fixes* $M_{\mathbf{A}}$*.*

$$\mathrm{Aut}(\mathbf{A}) \cong \{\sigma \in S_n \mid \sigma(M_{\mathbf{A}}) = M_{\mathbf{A}}\}$$

Ho et al. in [19] thus delineate a naïve method:

Algorithm 6: Naïve Automorphism Group Algorithm

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ corresponding to a left quasigroup $\mathbf{A}$
Output: $\mathrm{Aut}(\mathbf{A}) \leq S_n$

```
PermutationsList ← Permutations(M_{1⋆})
Automorphisms ← ∅
forall σ ∈ PermutationsList do
    if σ(M) ≟ M then
        Automorphisms ← Automorphisms ∪ {σ}
    end
end
Return: Aut(A) = (Automorphisms, ∘)
```

However, this quickly becomes expensive as the order of the left quasigroup grows.

n	$n!$
1	1
⋮	⋮
5	120
⋮	⋮
47	$2.59 \cdot 10^{59}$

Processing each of these permutations and executing two matrix multiplications for each of them becomes infeasible but there are ways to reduce the number of permutations that need to be processed *and* take advantage of the machinery provided by Magma. The two matrix multiplications are used as a filter to decide which bijection is an automorphism. However, it is possible to generalise Lemma 1.1 in [36] to left quasigroups which allows for a better filtering method.

Lemma 3.7.
If $\mathbf{A} = (S, \cdot, \backslash)$ *is a left quasigroup, then* $L_{h(x)} = hL_xh^{-1}$ *for every* $x \in S$ *and* $h \in \mathrm{Aut}(\mathbf{A})$*.*

Proof.
Let $\mathbf{A}$ be a left quasigroup. Let $x, y \in S$. Let $h \in \mathrm{Aut}(\mathbf{A})$.

$$\begin{aligned} hL_x(y) &= h(x \cdot y) \\ &= h(x) \cdot h(y) \quad \text{thus } hL_x(y)h^{-1} = L_{h(x)}. \\ &= L_{h(x)}h \end{aligned}$$

□

It follows from Lemma 3.7, that any automorphism h of a left quasigroup $\mathbf{A} = (X, \cdot)$ can be found inside the Normalizer of its left multiplication group, $\mathrm{Aut}(\mathbf{A}) \leq N_{S_X}(\mathrm{LMlt}(\mathbf{A})) = \{h \in S_X : h\,\mathrm{LMlt}(\mathbf{A})h^{-1} = \mathrm{LMlt}(\mathbf{A})\}$.
This, potentially, reduces the number of permutations to consider to those in $N_{S_X}(\mathrm{LMlt}(\mathbf{A}))$ *and* it is, potentially, faster to compute. As a matter of fact, in some cases, computing the normalizer of a group can be done in polynomial time [20].
Furthermore, Lemma 3.7 suggests a better way to check whether a bijective function h (such as a permutation σ) is an automorphism or not. For a left quasigroup $\mathbf{A}$, each row i of its operation matrix M represents L_{x_i} so, in case h is not automorphism, it is possible to interrupt the process much earlier than computing two matrix multiplications and the checking that the initial matrix and the resulting matrix are equal.

Algorithm 7: QuandleRUN Automorphism Group Algorithm

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ corresponding to a left quasigroup $\mathbf{A} = (X = (x_i)_{\{1..n\}}, \cdot)$
Output: $\mathrm{Aut}(\mathbf{A}) \leq S_n$

Permutations $\leftarrow N_{S_X}(\mathrm{LMlt}(\mathbf{A}))$
`/* Compute the left translations from` M `and create a map such that for` $x \in \mathbf{A} \quad x \mapsto L_x$`.`
`The map is defined by composing its functional graph. */`
$G(\mathsf{L}) \leftarrow \{(x_i, M_{i\star}) : 1 \leq i \leq n\}$
Automorphisms $\leftarrow \emptyset$
forall $\sigma \in$ Permutations **do**
  admissible $\leftarrow$ `true`
  forall $x \in M_{1\star}$ **do**
    if $\sigma^{-1}\mathsf{L}(x)\ \sigma \neq \mathsf{L}(\sigma(x))$ **then**
      admissible $\leftarrow$ `false`
      break
    end
  end
  if admissible **then**
    Automorphisms $\leftarrow$ Automorphisms $\cup \{\sigma\}$
  end
end
Return: $\mathrm{Aut}(\mathbf{A}) = (\mathsf{Automorphisms}, \circ)$

3.1.3 Subalgebras and Monomorphisms

Generators, introduced in Section 2, are important because they can speed-up some computations, especially those involving the search of homomorphisms. Hence being able to find a minimal set of generators fast is of primary importance. However, these two things do not necessarily go hand in hand – finding a minimal set of generators might take longer than finding any set of generators. Both Rig and CREAM solve the problem of finding a minimal generating set while QuandleRUN finds the minimal generating set containing element 1 of the underlying set of the quandle.

QuandleRUN's approach is closer to the approach taken by the developers of CREAM but it has been adapted in order to tend to quandles more. The approach taken by CREAM is particularly interesting because it highlights some important aspects connected to the problem of finding a monomorphism between two quandles, analysed at the end of this subsection.

Inspired by [30], the developers of CREAM start the search of a generating set with the computation of 17 element-wise invariants and partition the set accordingly (if two elements have the same invariant values, they are placed in the same equivalence class). Then they look for a minimal generating set, favouring, where possible, elements from equivalence classes with less elements. Partitioning the underlying set is not particularly helpful in the search of generators but it is extremely useful when discussing the problem of finding a monomorphism between two left quasigroups, analysed at the end of this subsection; furthermore, choosing generators for smaller equivalence classes of the partition helps in the search for automorphisms [14].

QuandleRUN does not compute these element-wise invariants at all, furthermore, QuandleRUN skips the first round of CREAM's computation because any element of a quandle is idempotent, thus computing the subalgebras generated by one element will not yield any bigger subalgebra. Ultimately, QuandleRUN tries to exploit the fact that all the elements of the superquandle are known beforehand by using a lookup table to avoid some computations. The definitions and computations mentioned above give rise to three algorithms.

Algorithm 8: Expand a subalgebra by one element - $\texttt{Sg}(M, S, x)$

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ representing a quandle $\mathbf{A} = (\{1..n\}, \cdot, \backslash)$;
A subalgebra S of $\mathbf{A}$;
$x \in X$;
Output: $\mathrm{Sg}(S \cup \{x\})$

```
LookupTable[i] ← i ∈? S
LookupTable[x] ← true
ToBeAdded ← {x}
while ToBeAdded ≠ ∅ do
    x ← Pop(ToBeAdded)
    for y ∈ S do
        for z ∈ {M_xy, M_yx} do
            if ¬LookupTable[z] then
                ToBeAdded ← ToBeAdded ∪ {z}
                LookupTable[z] ← true
            end
        end
    end
    S ← S ∪ {x}
    if |S| + |ToBeAdded| = n then
        Return: M_1⋆
    end
end
Return: S
```

Algorithm 8 is similar to Algorithm 18 in [14] but it has been adapted to try to be faster in practice.
The algorithm to compute $\mathrm{Sg}(S)$ given a set $S \subseteq X$ for a quandle $\mathbf{A} = (X, \cdot)$ is naïve so it will not be outlined here.

Algorithm 9: QuandleRUN's generating set algorithm - `Generators(`M`)`

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ representing a quandle $\mathbf{A} = (\{1..n\}, \cdot, \setminus)$;
Output: S such that $\mathrm{Sg}(S) = \mathbf{A}$

```
Subalgebra ← {1}
S ← {1}
LargestSoFar ← {1}
TemporaryGenerators ← {1}
PotentialGenerators ← {2..n}
while |Subalgebra| ≠ n do
    while PotentialGenerators ≠ ∅ do
        x ← Pop(PotentialGenerators)
        TemporarySubalgebra ← Sg(M, Subalgebra, x)
        if TemporarySubalgebra > LargestSoFar then
            LargestSoFar ← TemporarySubalgebra
            TemporaryGenerators ← S ∪ x
            if |LargestSoFar| = n then
                Return: TemporaryGenerators
            end
        end
        PotentialGenerators ← PotentialGenerators \ TemporarySubalgebra
    end
    Subalgebra ← LargestSoFar
    S ← TemporaryGenerators
    PotentialGenerators ← {x : x ∈ M_{1⋆}}\ Subalgebra
end
Return: S
```

The problem of finding all the subalgebras of a given quandle is also connected to these matters. The approaches taken in Rig and CREAM are wildly different. Rig finds all subalgebras in a naïve fashion. CREAM finds all subalgebras of a quandle $\mathbf{A}$ by computing all the subalgebras with 1 generator, expanding each of them by one element from each orbit of $\mathrm{Aut}(\mathbf{A})$ and then, by applying the group action of $\mathrm{Aut}(\mathbf{A})$, it computes all the other subalgebras generated by the same amount of generators; the algorithm finishes when no subalgebra shows up other than $\mathbf{A}$ itself.

QuandleRUN tries to combine these approaches: it tries to find all the subalgebras naïvely like Rig but, at each round, computes the generated subalgebra like CREAM. It uses hash tables to speed up some computations – it could even use hash tables where each bucket allows for only one element but given the widespread shortcomings of all systems when it comes to hashing sets, this is not possible. For example, the sets $\{1, 5, 9, 13\}$ and $\{2, 6, 10, 14\}$ are hashed to the same value in Magma which means that one of them would be ignored, if they were both subalgebras of a left quasigroup $\mathbf{A}$.

For clarity, the algorithm that uses hash tables will be shown.

Algorithm 10: QuandleRUN's Subalgebras algorithm - `Sub(A)`

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ representing a quandle $\mathbf{A} = (\{1..n\}, \cdot, \backslash)$;
Output: Sub($\mathbf{A}$), the set of subalgebras of $\mathbf{A}$, as defined in [12]

```
Subalgebras ← [ ]
HashTable ← [ ]
for x ∈ M_{1⋆} do
    Subalgebra ← {x}
    Append(Subalgebras, Subalgebra)
    hash ← Hash(Subalgebra)
    /* IsDefined checks whether there are entries for a specified
       hash.                                                      */
    if ¬IsDefined(HashTable, hash) then
        HashTable[hash] ← [Subalgebra]
    else
        Append(HashTable[hash], Subalgebra)
    end
end
index ← 1
while index ≤ |Subalgebras| do
    x ← Subalgebras[index]
    for y ∈ M_{1⋆}\x do
        if |x| < n then
            Subalgebra ← Sg(M, x, y)
            hash ← Hash(Subalgebra)
            if ¬IsDefined(HashTable, hash) then
                HashTable[hash] ← [Subalgebra]
                Append(Subalgebras, Subalgebra)
            else
                if Subalgebra ∉ HashTable[hash] then
                    Append(HashTable[hash], Subalgebra)
                end
            end
        end
    end
    index ← index +1
end
Return: Subalgebras
```

An interesting decision problem with regards to subalgebras is:

> Given a quandle $\mathbf{A}$ of order n and a quandle $\mathbf{B}$ of order $m \leq n$.
> Is $\mathbf{B}$ a subalgebra of $\mathbf{A}$? Or rather, *Is there a monomorphism from* $\mathbf{B}$ *to* $\mathbf{A}$*?*

According to Definition 2.7, one needs to be able to be able to find a monomorphism from $\mathbf{B}$ to $\mathbf{A}$. QuandleRUN only works with left quasigroups of finite order, so an algorithm to find a monomophirsm allows to solve two problems:

- Find a monomorphism between two left quasigroups, if one exists.
- Find an isomorphism between two left quasigroups, if one exists.

It is appropriate, for this section, to restrict the focus on quandles alone.
RIG does have an algorithm to compute homomorphisms between two quandles, but it acts naïvely: tries all possible images for the values in the domain and stores all those mappings that are homomorphisms and then returns this list.

QuandleRUN's algorithm is very similar to that of CREAM, except for some preliminary steps. Practically speaking, the author has found that the element-wise invariants proprosed in [14] are of no help for quandles; and especially connected quandles. The author has computed the invariants suggested by [14] for the entire database of 791 connected quandles available in [38] and for several disconnected quandles and all lead to the partition with only one equivalence class. These are the reasons why the author has decided to save some time and computing power by cutting the computation of invariants out of the process.

Should effective element-wise invariants for quandles be found – the framework to support them is mostly already in place. The following two pages will outline the three main components in the search for a monomorphism between two quandles, since these procedure do not rely on any property of quandles or racks, they can be applied directly to left quasigroups.

Algorithm 11: Monomorphism Algorithm Part 1 - `Monomorphism1(A,B)`

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ representing a left quasigroup $\mathbf{A} = (\{1..n\}, \cdot, \backslash)$;
A $m \times m$ matrix F over $\{1..n\} \subset \mathbb{Z}$ representing a left quasigroup $\mathbf{B} = (\{1..m\}, \star, \backslash_\star)$;
Output: A list $[h_1, h_2, \ldots, h_n]$ such that, for an element $x \in \{1..n\}$ $f(x) = h_x$, where f is a monomorphism from $\mathbf{A}$ to $\mathbf{B}$, if one exists. An empty list [ ], otherwise.

if $n > m$ **then**
  Return: [ ]
end
GeneratorsA $\leftarrow$ `Generators(`M`)`
PreImages $\leftarrow \emptyset$
Images $\leftarrow [0_1, 0_2, \ldots, 0_n]$
Return: `Monomorphism2(`M, F, GeneratorsA, *(*Images, PreImages*)*`)`

Algorithm 12: Monomorphism Algorithm Part 2 - `Monomorphism2(A,B,G,P)`

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ representing a left quasigroup $\mathbf{A} = (\{1..n\}, \cdot, \backslash)$;
A $m \times m$ matrix F over $\{1..n\} \subset \mathbb{Z}$ representing a left quasigroup $\mathbf{B} = (\{1..m\}, \star, \backslash_\star)$;
A set G of generators of $\mathbf{A}$;
A tuple (Images,Preimages);

Output: A list $[h_1, h_2, \ldots, h_n]$ such that, for an element $x \in \{1..n\}$ $f(x) = h_x$, where f is a monomorphism from $\mathbf{A}$ to $\mathbf{B}$, if one exists. An empty list [ ], otherwise.

```
if G = ∅ then
 | Return: Monomorphism3(M, F,(Images,Preimages))
end
x ← Pop(G)
AvailableImages ← [y ∈ M_{1⋆} | y ∉ Images]
for y ∈ AvailableImages do
 | (Images2,Preimages2) ← P
 | Images2[x] ← y
 | Preimages2 ← Preimages2 ∪ {x}
 | Monomorphism ← Monomorphism2(M, F, G, (Images2,Preimages2))
 | if Monomorphism ≠ [] then
 | | Return: Monomorphism
 | end
end
```

The third part of the algorithm can be found at the end of this section.
The algorithm returns a monomorphism, if one exists.

If a monomorphism exists and the domain has just as finitely many elements as the codomain, this suggest that the function is bijective and thus, an isomorphism.
Although element-wise invariants have proven to be of little help, it is possible to exploit global invariants to rule out the existence of monomorphisms and thus avoid the search altogether.

Algorithm 13: Monomorphism Algorithm Part 3 - `Monomorphism3(A,B,P)`

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ representing a left quasigroup $\mathbf{A} = (\{1..n\}, \cdot, \backslash)$;
A $m \times m$ matrix F over $\{1..n\} \subset \mathbb{Z}$ representing a left quasigroup $\mathbf{B} = (\{1..m\}, \star, \backslash_\star)$;
A tuple (Images,Preimages);

Output: A list $[h_1, h_2, \ldots, h_n]$ such that, for an element $x \in \{1..n\}$ $f(x) = h_x$, where f is a monomorphism from $\mathbf{A}$ to $\mathbf{B}$, if one exists. An empty list [ ], otherwise.

```
NewElements ← {x : x ∈ Preimages}
OldElements ← {}
LookupTable[i] ← i ∈? Images
while NewElements ≠ ∅ do
    for x, y ∈ NewElements do
        Pairs ← Pairs ∪ {(x, y)}
    end
    for x ∈ OldElements do
        for y ∈ NewElements do
            Pairs ← Pairs ∪ {(x, y), (y, x)}
        end
    end
    results ← ∅
    for (x,y) ∈ Pairs do
        z ← M_xy
        Hx ← Images(x)
        Hy ← Images(y)
        Hz ← Images(z)
        if Hz = 0 ∧ ¬LookupTable[F_HxHy] then
            Images(z) ← F_HxHy
            Preimages ← Preimages ∪ {z}
            LookupTable[F_HxHy] ← true
        else
            if Hz ≠ F_HxHy then
                Return: [ ]
            end
        end
    end
    OldElements ← OldElements ∪ NewElements
    NewElements ← results
end
Return: Images
```

3.1.4 Congruences

The formation of quotient structures is one of the fundamental constructions in algebra. By definition 2.6, quotient structures are described in terms of congruence relations. Given a congruence and a left quasigroup, computing the quotient left quasigroup relative to that congruence is simple.

Algorithm 14: Construction of a Quotient Left Quasigroup

Input : A left quasigroup $\mathbf{A} = (S, \cdot, \backslash)$;
A congruence π on $\mathbf{A}$;
Output: $\mathbf{A}/\pi = (S/\pi, \cdot/\pi)$

$\mathsf{UnderlyingSet} \leftarrow \{\texttt{Representative}(\pi_i)\}_{\pi_i \in \pi}$
`/* Define the operation by its functional graph` $G(\cdot/\pi)$ `*/`
$G(\cdot/\pi) \leftarrow \{((x, y), x \cdot y) : x, y \in \mathsf{UnderlyingSet}\}$
Return: $\mathbf{A}/\sim = (\mathsf{UnderlyingSet}, \cdot/\pi, \backslash/\sim)$

But how does one get the congruences without processing each partition of the underlying set? For sets endowed with unary operations, called *unary algebras*, Ralph Freese in [17] proposes a very efficient algorithm to compute the smallest congruence such that the two elements of the pair are in the same block, such a congruence is called *principal congruence* generated by a pair. In [17], Ralph Freese already pointed out that the same algorithm could be "adapted" to work on binary algebras, such as left quasigroups, by "unarifying" them.

The process of converting a binary operation into a set of unary operations is simple; it consists of considering each row and each column as a unary operation, putting them in list and removing all duplicates. It described by the following algorithm:

Algorithm 15: Unarifying a binary operation

Input : A $n \times n$ left quasigroup matrix M over $\{1..n\} \subset \mathbb{Z}$ corresponding to a left quasigroup $\mathbf{A} = (X = (x_i)_{\{1..n\}}, \cdot)$
Output: S, the set of unary operations induced by the left quasigroup matrix

```
UnaryOperations ← ∅
for i = 1 to n do
    row ← [ ]
    column ← [ ]
    for j = 1 to n do
        Append(row, M_ij)
        Append(column, M_ji)
    end
    UnaryOperations ← UnaryOperations ∪ {row, column}
end
Return: UnaryOperations
```

For completeness, the algorithm described in [17] will be shown. It is important to note that in order to speed-up computations, [17] represents partitions differently from the "traditional" set of sets.
Any partition π of $\{1..n\}$ is represented by a list of length n where the i-th item $L[i]$ is defined as follows:

$$L[i] = \begin{cases} -|\pi_b| & \text{if } i = \min(\pi_b) \\ \min(\pi_b) & \text{otherwise} \end{cases}$$

where π_b is the block of π where i is.
For example, the partition $\{\{1\}..\{n\}\}$ is described by $[-1,\ldots,-1]$ and the partition $\{\{1,2\},\{3,4\},\ldots,\{n-1,n\}\}$ is described by $[-2,1,-2,3,\ldots,-2,n-1]$.

Algorithm 16: Principal congruence generated by pair x, y - `Cg(L, x, y)`

Input : A list L of unary operations corresponding to a left quasigroup $\mathbf{A} = (X = (x_i)_{\{1..n\}}, \cdot)$;
A pair $(x, y) \in X \times X$

Output: The partition induced by $\mathrm{Cg}^{\mathbf{A}}(x, y)$, the principal congruence on $\mathbf{A}$ generated by (x, y)

```
Pairs ← Queue({(x, y)})
Partition ← [−1_1, −1_2, ..., −1_n]
/* JoinBlocks(P, a, b) joins the blocks of partition P containing a
   and b                                                          */
Partition ← JoinBlocks(Partition, x, y)
while Pairs ≠ Queue(∅) do
    (x,y) ← Dequeue(Pairs)
    for f ∈ L do
        /* Root(P, a) returns the root (the least element according
           to [14] and the greatest element according to [17]) of the
           block in P in which a is.                              */
        z ← Root(Partition,f(x))
        t ← Root(Partition,f(y))
        if z ≠ t then
            Partition ← JoinBlocks(Partition, z, t)
            Pairs ← Enqueue(Pairs, (z,t))
        end
    end
end
Return: Partition
```

Then, following the procedure laid out by the developers of CREAM, one can compute all principal congruences of a left quasigroup.

Algorithm 17: All principal congruences - `PC(L)`

Input : A list L of unary operations corresponding to a left quasigroup $\mathbf{A} = (\{1..n\}, \cdot)$
Output: S, the set of all partitions induced by principal congruences on $\mathbf{A}$.

```
S ← ∅
for i = 1 to n − 1 do
    for j = i + 1 to n do
        Congruence ← Cg(L, (i, j))
        S ← S ∪ {Congruence}
    end
end
Return: S
```

Now all the components are in place to compute all congruences of a left quasigroup.

Algorithm 18: All congruences

Input : A list L of unary operations corresponding to a left quasigroup $\mathbf{A}$
Output: The set of all partitions induced by $\Theta(\mathbf{A})$, the set of all congruences on $\mathbf{A}$.

```
PrincipalCongruences ← PC(L)
Congruences ← PrincipalCongruences
index ← 1
while index < |Congruences| do
    for i = 1 to |PrincipalCongruences| do
        if PrincipalCongruences[i] ⋠ Congruences[index] then
            /* Join(A, B) returns the smallest partition containing
               both A and B                                          */
            congruence ← Join(Congruences[index], PrincipalCongruences[i])
            Congruences ← Congruences ∪ {congruence }
        end
    end
    index ← index + 1
end
```

Algorithm 15 to Algorithm 18 are all described in [14] and they are also implemented in QuandleRUN; using them, QuandleRUN is able to compute the congruences on all left quasigroups whose left multiplication group is not transitive.
Technically speaking, these algorithm are able to compute the congruences of any binary algebra; however, the author has found a way to implement Lemma 1.5 in [8], which allows to harness Magma's power and to obtain the congruences in a much quicker way, at least for connected left quasigroups.

Before presenting Lemma 1.5, it is appropriate to introduce the notion of *block system* with respect to the action of a group G. A block system is a partition π such that the action of G on each block $\pi_b \in \pi$ either maps the block to itself $\pi_b^G = \pi_b$ or to another block in the same partition, thus out of itself $\pi_b^G \cap \pi_b = \emptyset$.

Lemma 3.8 (**Lemma 1.5** in [8])**.**
Let $\mathbf{A}$ *be a left quasigroup and* $\sim$ *an equivalence relation on* $\mathbf{A}$*. The following are equivalent:*

1. $\sim$ *is a congruence.*
2. $\sim$ *is a block system of* $\mathrm{LMlt}(\mathbf{A})$ *and* $\mathrm{Dis}_{\sim}(\mathbf{A}) \leq \mathrm{Dis}^{\sim}(\mathbf{A})$.

By expanding the algorithm for finding blocks of imprimitivity due to [33, 34], or even the slower [3], one can constructs all the block systems of the left multiplication group of the left quasigroup. This construction acts as a filter, retaining only the partitions induced by equivalence relations that might turn out to be congruences of the left quasigroup. Lemma 3.8 warrants the creation of methods to compute $\mathrm{Dis}_{\sim}$ and $\mathrm{Dis}^{\sim}$ which will be found in the next page.

Algorithm 21 would be enough but there is the "problem" of computing the kernel and the displacement group relative to the congruence relation. Why compute both when it is enough $x \notin \mathrm{Dis}^{\sim}$, for an element $x \in \mathrm{Dis}_{\sim}$, to interrupt the computation earlier? This already suggests new ways to make the process faster but one can go a step further and optimize the process even more such that that neither the kernel nor the displacement group relative to the congruence relation need to be computed.

Algorithm 19: Displacement group relative to $\sim$ - `DisB`$(M, S/\sim)$

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ corresponding to a connected rack $\mathbf{A} = (S = \{1..n\}, \cdot)$;
The partition $S/\sim$ induced by the congruence relation $\sim$ on $\mathbf{A}$
Output: $\mathrm{Dis}_{\sim}$, the displacement group of $\mathbf{A}$ relative to the congruence relation $\sim$

$\mathsf{L} \leftarrow \emptyset$
for block $\in S/\sim$ **do**
  $\mathsf{L} \leftarrow \mathsf{L} \cup \{M_{i\star}M_{j\star}^{-1} : i, j \in \mathsf{block}\}$
end
`/* Subgroup(`x, y`) returns the subgroup generated by` y `within` x`, thus` $y \subseteq x$ `*/`
Return: `Subgroup(`S_n, L`)`

Algorithm 20: Kernel relative to $\sim$ - `DisT`$(M, S/\sim)$

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ corresponding to a connected left quasigroup $\mathbf{A} = (S = \{1..n\}, \cdot)$;
The partition $S/\sim$ induced by the congruence relation $\sim$ on $\mathbf{A}$

Output: Dis$^{\sim}$, the kernel relative to the congruence relation $\sim$

```
kernel ← ∅
Dis ← DisB(M, {{1..n}})
for h ∈ Dis do
    admissible ← true
    for element = 1 to n do
        Ha ← h(element)
        round ← false
        for block ∈ S/∼ do
            if Ha ∈ block ∧ element ∈ block then
                round ← true
                break
            end
        end
        admissible ← admissible ∧ round
        if ¬admissible then
            break
        end
    end
    if admissible then
        kernel ← kernel ∪ { h }
    end
end
```

Algorithm 21: All congruences of a connected left quasigroup

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ corresponding to a connected left quasigroup $\mathbf{A} = (S, \cdot)$

Output: The set of all partitions induced by $\Theta(\mathbf{A})$, the set of all congruences on $\mathbf{A}$.

```
LMlt ← LMlt(M)
Congruences ← {{{x} : x ∈ S}, S}
L ← {M_{i⋆} : i ∈ [1..|M|]}
/* AllPartitions(G) constructs all non-trivial block systems of
   the transitive permutation group G. [11]                    */
BlockSystems ← AllPartitions(LMlt)
for π ∈ BlockSystems do
    if DisB(M, π) ≤ DisT(M, π) then
        Congruences ← Congruences ∪ {π}
    end
end
Return: Congruences
```

Lemma 3.9.
Let $\mathbf{A} = (S, \cdot, \backslash)$ *be a connected quandle,* $\sim$ *be a block system of* $\mathrm{LMlt}(\mathbf{A})$ *and consider* $[a_0] \in \sim$ *for some* $a_0 \in S$.
Define $H = \{L_b L_{a_0}^{-1} : b \in [a_0]_\sim\}$. *If* $H \subset \mathrm{Dis}^\sim(\mathbf{A})$ *then* $\mathrm{Dis}_\sim(\mathbf{A}) \leq \mathrm{Dis}^\sim(\mathbf{A})$.

Proof.
Assume that $H \subset \mathrm{Dis}^\sim(\mathbf{A})$.
Given that $[a_0]_\sim$ is a block, $\forall x \in S \quad \exists h \in \mathrm{Dis}^\sim(Q) \quad [x]_\sim = h([a_0]_\sim)$.
For any $x \sim y \in S$, there exist $x_0, y_0 \in [a_0]_\sim$ and $h \in \mathrm{Dis}(\mathbf{A})$ such that $x = h(x_0)$ and $y = h(y_0)$:

$$\begin{aligned} L_x L_y^{-1} &= L_{h(x_0)} L_{h(y_0)}^{-1} \\ &= h L_{x_0} h^{-1} h L_{y_0}^{-1} h^{-1} \\ &= h L_{x_0} L_{y_0}^{-1} h^{-1} \\ &= h \underbrace{L_{x_0} L_{a_0}^{-1}}_{\in \mathrm{Dis}^\sim(\mathbf{A})} \underbrace{L_{a_0} L_{y_0}^{-1}}_{\in \mathrm{Dis}^\sim(\mathbf{A})} h^{-1} \in \mathrm{Dis}^\sim(\mathbf{A}) \end{aligned}$$

□

Theorem 3.10.
Let $\mathbf{A}$ *be a connected quandle. The set* Congruences *returned by Algorithm 22 is the the set of all partitions induced by the congruences of* $\mathbf{A}$.

Proof.
One can assume that $P =$ `AllPartitions(LMlt)` where `LMlt=`$\mathrm{LMlt}(\mathbf{A})$ returns the expected result since it is based on [33, 34] and it is implemented in Magma. Take an arbitrary $a_0 \in \mathbf{A}$.
Take an arbitrary $\pi \in P$. Let $\sim$ be the equivalence relation inducing π. Define $H = \{L_x L_{a_0}^{-1} : x \in [a_0]_\sim\}$. From Definition 2.5, it follows that $H \subset \mathrm{Dis}_\sim(\mathbf{A})$ and that $H \subset \mathrm{Dis}(\mathbf{A})$. In quandles, $\mathrm{LMlt}(\mathbf{A})$ and $\mathrm{Dis}(\mathbf{A})$ have the same orbits [**Proposition 2.1(iv)** in [21]]. Thus, for any element $h \in H$ and $x \in \mathbf{A}$, either $h(x) \in [x]_\sim$ and so $\forall y \in [x]_\sim \quad h(y) \in [x]_\sim$ (case 1) *or* $h(x) \notin [x]_\sim$ and so $\forall y \in [x]_\sim \quad h(y) \notin [x]_\sim$ (case 2). In case 2, $h \notin \mathrm{Dis}^\sim(\mathbf{A})$ so $H \not\subset \mathrm{Dis}^\sim(\mathbf{A})$ and π is not added to Congruences. Case 1 justifies checking only a representative for each block. If case 1 holds for any $h \in H$, then $H \subset \mathrm{Dis}^\sim(\mathbf{A})$.
Therefore, assume that $H \subset \mathrm{Dis}^\sim(\mathbf{A})$.
Given Lemma 3.9, $\mathrm{Dis}_\sim(\mathbf{A}) \leq \mathrm{Dis}^\sim(\mathbf{A})$, thus π is added to the set Congruences.

□

Algorithm 22: All congruences of a connected quandle

Input : A $n \times n$ matrix M over $\{1..n\} \subset \mathbb{Z}$ corresponding to a connected quandle $\mathbf{A} = (\{1..n\}, \cdot)$

Output: The set of all partitions induced by $\Theta(\mathbf{A})$, the set of all congruences on $\mathbf{A}$.

```
LMlt ← LMlt(M)
Congruences ← {{{x} : x ∈ S}, S}
L ← [M_{i⋆} : i ∈ [1..|M|]]
/* AllPartitions(G) constructs all non-trivial block systems of
   the transitive permutation group G. [11]                      */
BlockSystems ← AllPartitions(LMlt)
a0 ← 1
for π ∈ BlockSystems do
    admissible ← true
    A ← {L[x]L[a0]^{-1} : x ∈ π_1}
    for π_b ∈ π do
        y ← Representative(π_b)
        for a ∈ A do
            if a(y) ∉ π_b then
                admissible ← false
                break
            end
        end
    end
    if admissible then
        Congruences ← Congruences ∪ {π}
    end
end
Return: Congruences
```

3.2 Applications

3.2.1 Visualising subalgebras

Let L be a set of left quasigroups.
It is possible to define a transitive relation $\mathsf{sub} = \{(x, y) \in L \times L : y \text{ is a subalgebra of } x\}$. The transitivity of sub outlines an elegant way to visualise the relation for any left quasigroup $\mathbf{A}$:

1. Define a directed graph $G = (V = \mathrm{Sub}(\mathbf{A}), E = \emptyset)$.
2. Define $I = (\mathrm{Sub}(\mathbf{A}) \setminus \{\mathbf{A}\}) \setminus \bigcup_{x \in \mathrm{Sub}(\mathbf{A})} (\mathrm{Sub}(x) \setminus \{x\})$.
3. $E \cup \bigcup_{x \in I} (x, \mathbf{A})$
4. Repeat steps 2 and 3 for each element $x \in I$.

Step 2 is needed to avoid duplicate paths going from a vertex representing a smaller left quasigroup to one representing a bigger left quasigroup: the set I defined for a left quasigroup $\mathbf{A}$ is the set of subalgebras that can only be embedded in $\mathbf{A}$ and in none of its proper subalgebras.
Since quandles are left quasigroups, one can use the strategy outlined above to visualise sub on a large set of quandles. This can help to immediately identify quandles with a certain characteristic; for example strictly simple quandles, quandles with no proper subquandle with more than one element, discussed in [5].

In Figure 1, one can see a graph of subquandles for all connected quandles up to order 12. The isolated vertices and the vertex corresponding to the quandle with only one element have been removed. The edges going from quandles of smaller order to those of greater order are meant to represent the monomorphisms going from subalgebra to superalgebra.

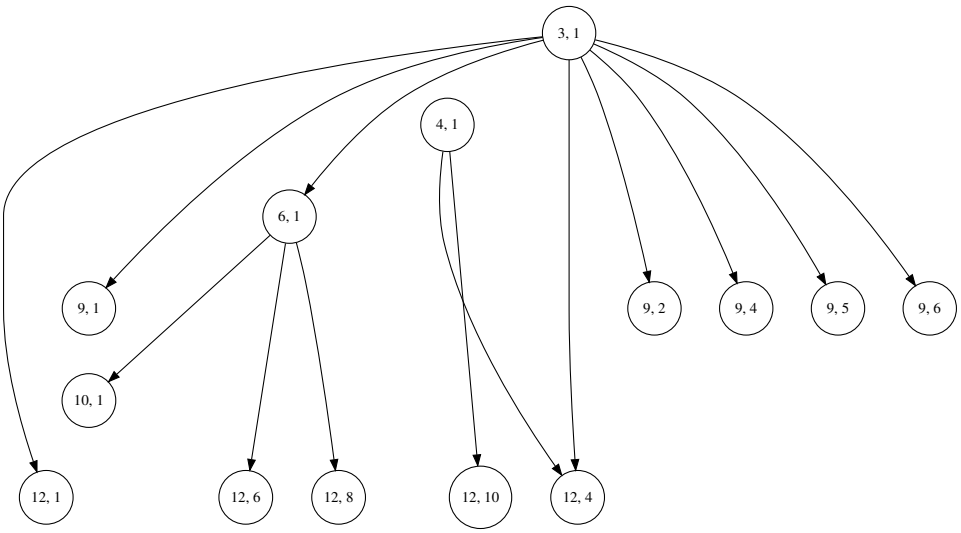

Figure 1

3.2.2 Visualising congruence lattices of left quasigroups

The algorithms to compute the congruences of a left quasigroup available in QuandleRUN can be used to draw the congruence lattice of any left quasigroup. The commutator theory for racks and quandles is described in [9], it is built on the commutator theory for universal algebra developed by Smith in [35] and expanded by Freese in [18].

Restricting the focus on racks, it is possible to distinguish whether a congruence $\sim$ on a rack $\mathbf{A}$ is central, or abelian, according to group-theoretic properties of $\mathrm{Dis}_{\sim}(\mathbf{A})$ [9]. The action of a group H is $\sim$-*semiregular* for a congruence $\sim$ on a rack $\mathbf{A} = (S, \cdot)$ if for every $h \in H$, if $h(x) = x$ then $h(y) = y$ for any two elements $x, y \in S$ such that $X \sim y$. The congruence $\sim$ is called *central* if and only if the action of $\mathrm{Dis}(\mathbf{A})$ is $\sim$-semiregular on $\mathbf{A}$ and $\mathrm{Dis}_{\sim}(\mathbf{A})$ is central in $\mathrm{Dis}(\mathbf{A})$ [9]. The congruence $\sim$ is called *abelian* if and only if the action of $\mathrm{Dis}_{\sim}(\mathbf{A})$ is $\sim$-semiregular on $\mathbf{A}$ and $\mathrm{Dis}_{\sim}(\mathbf{A})$ is abelian [9].

Figure 2 shows three possible congruence lattices of a locally strict simple quandle, one whose all proper subquandles are strictly simple [7]. The vertices coloured in gold (■) indicate that the congruence is central, the vertices coloured in light green (■) indicate that the congruence is abelian.

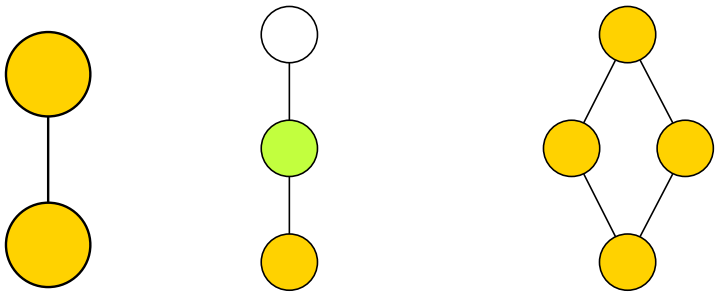

Figure 2: (a) Simple (b) Subdirectly Irreducible (c) Subdirectly reducible

The following, more interesting, examples are the congruence lattices of the 10th connected quandle of order 36 and of the 42nd connected quandle of cardinality 36.

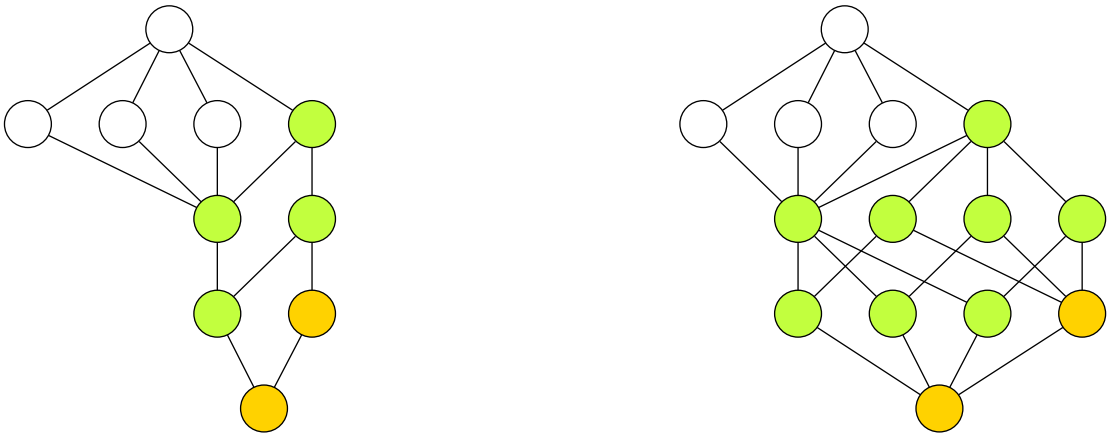

Figure 3

The congruence lattice of a left quasigroup can convey information about the original structure. In the case of quandles, a congruence lattice as in Figure 2(c) indicates that the quandle is a direct product of a pair of quandles and congruence lattice as in Figure 2(b) might indicate a connected non-abelian locally strict simple quandle [7].

3.2.3 Visualising quotient structures

Following this line of thought, one can visualise the factors of a quandle.
In Figure 4, one can see the graph of quotient quandles for all connected quandles up to order 12 where the vertex representing the quandle of order 1 was removed. The edges going from the quandles of greater order to those of smaller order are meant to represent the epimorphisms going from structure to quotient structure.

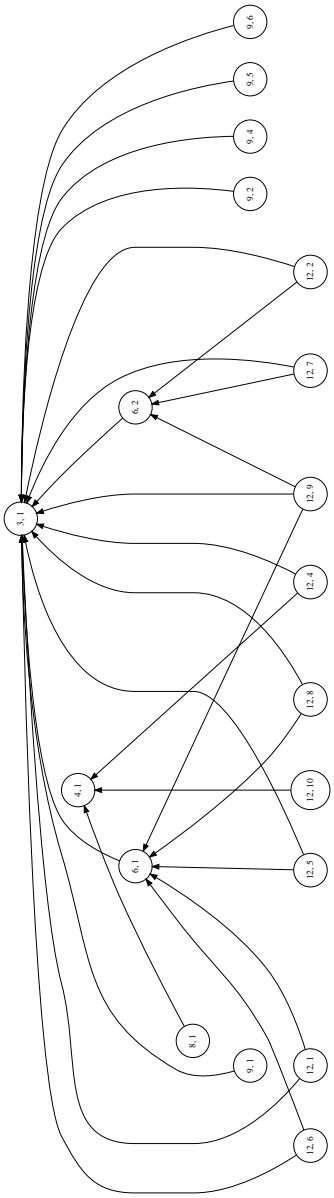

Figure 4

3.3 Comparison

This section will show a comparison between QuandleRUN, Rig and CREAM in terms of running times on the following tasks: computation of the automorphisms group, search of a generating set, search of all subalgebras, computation of all congruences.

Each command has been run on a shared university terminal running Ubuntu 20.04.4 LTS (Focal Fossa) with AMD EPYC 7502P 32-Core Processor and a 64GB RAM. The algorithms have been tested on the full database of 791 connected quandles available in Rig. The values displayed are the medians: the median time taken by the software to carry out the tasks mentioned above for a connected quandle of order $x \in [X, Y] \subset \mathbb{N}$. The author decided to display the median instead of the traditional mean because, due to a few peculiar quandles in the database, the results would not have reflected the time taken to carry out the tasks mentioned above for most connected quandles, they would have been heavily skewed towards high untruthful values.

Order	QuandleRUN(s)	Rig(s)	CREAM(s)
1-10	0.000	0.000	0.000
11-20	0.000	0.001	0.002
21-30	0.000	0.004	0.004
31-40	0.000	0.007*	0.006
41-47	0.000	✗	0.009

Table 1: Running times to find a generating set.

According to Table 1, when finding a generating set, CREAM is very fast but slightly slower than QuandleRUN. This is due to the fact that QuandleRUN skips the computation of invariants, *and* the computations of subalgebras generated by one element by exploiting the idempotence of quandles. Rig is slower than both because first it computes all the subsets of the underlying set of the quandle of cardinality 1 and checks whether any of them generates the quandle; if not, it moves to all the subsets of cardinality 2 and checks whether any of them generates the quandle and so on until a generating set is found.

Order	QuandleRUN(s)	Rig(s)	CREAM(s)
1-10	0.000	0.000	0.001
11-20	0.020	0.022	0.003
21-30	0.060		0.007
31-40	0.165		0.011
41-47	0.300		0.016

Table 2: Running times to compute the automorphism group.

According to Table 2, which refers to the computation of the automorphism group, one can see that, while QuandleRUN is faster than Rig, the only software available explicitly

*The computation was interrupted while working on the last quandle of order 36; it reached the pre-set memory limit.

dedicated to the study of quandles, CREAM is definitely the winner of this competition. This is interesting since the strategies used by Rig and CREAM are very similar and the only differences between them are that CREAM looks for a generating set before looking for the automorphisms, since that might reduce the number of trials necessary to find valid homomorphisms, as explained in Section 3.1.3. Furthermore, most of the computation of CREAM is written in the programming language `C` rather than the language of the computer algebra system GAP. Rig naïvely tries to assign different images to the values of the domain until all valid automorphisms have been found. The use of `C`, combined with a different strategy, on the other hand, explains the advantage of CREAM over QuandleRUN.

Order	QuandleRUN(s)	Rig(s)	CREAM(s)
1-10	0.000	0.000	0.001
11-20	0.001	0.006	0.005
21-30	0.005	0.042	0.016
31-40	0.170	0.149	
41-47	0.250	0.316	

Table 3: Running times to find all subalgebras.

According to Table 3, QuandleRUN and Rig are the winners when computing subalgebras. CREAM needs to compute the automorphisms group first and then, for any subalgebras it finds "naïvely", it applies the action of the automorphisms group to find all the other ones generated by the same amount of generators until it can only find the underlying set of the superquandle. CREAM is very fast in computing the set of automorphisms (as can be seen above) but it applies the automorphisms naïvely and does not use hash tables to check whether it has already seen a quandle or not. These are all contributing factors that make it the slowest of the three in the long run, while Rig, with its simplicity, manages to almost keep up with QuandleRUN and even be much faster in a handful of instances. For example, computing all the subalgebras of the 45th quandle of order 45 in QuandleRUN takes over 8 hours whereas Rig needs slightly less than 3 hours. This advantage might be explained, in minimal part, by technical differences of the underlying systems, GAP and Magma: GAP allows the user to add elements to the list at the base of a for-loop whereas, in Magma, once a for-loop over a list is defined, elements can be added to the list but the for loop will stop once it reaches the elements that was the last element when the for loop was defined. This means that to achieve a similar behaviour, QuandleRUN has to use a while loop that needs to check the loop condition at each iteration which slows the process down.

Order	QuandleRUN(s)	CREAM(s)
1-10	0.000	0.000
11-20	0.000	0.000
21-30	0.000	0.002
31-40	0.000	0.007
41-47	0.000	0.013

Table 4: Running times to compute all congruences

Finally, the system are compared based on how they perform when computing all congruences. RIG cannot compute the congruences of a quandle. The difference in speed between QuandleRUN and CREAM can be explained by the difference in strategy. QuandleRUN relies on MAGMA to constructs all non-trivial partitions preserved by the left multiplication group of the connected quandle. This ensures speed, since QuandleRUN will only need to check the second condition of Lemma 3.8, and guarantees that QuandleRUN will not have to deal with possible duplicates. CREAM, on the other hand, needs to unarify the algebra, construct the congruences "manually" and make sure that there are no duplicates, which takes time.

Chapter 4

Conclusions

This document aimed to introduce QuandleRUN, a modern tool for research in the theory of quandles. It shows how, by incorporating existing ideas, developing new ones based on recently discovered theory, or by looking at the theory from a different perspective, one can provide the foundations for an open and easily extendable research tool. Furthermore, being fully written in Magma, it ensures that researchers have at their disposal the best instruments to improve upon it.

Just like Magma, QuandleRUN uses the language of sets and maps and, thus provides a natural way to any mathematician of describing computations both to improve their own tool or to explore new ideas in quandle theory. Furthermore, QuandleRUN provides a common language for quandle theorists and a common interface to a library of algorithms; all things bound to improve communication and favour computational experimentation. This document hopes to stimulate a more experimental way of researching and hopes that QuandleRUN can be a tool that, quoting [10]:

> *"[...] provides a compelling way to generate understanding and insight; to generate and confirm or confront conjectures; and generally to make mathematics* [Author's note: read *Quandle theory*] *more tangible, lively, and fun for both the professional researcher and the novice."*

The source code of QuandleRUN along with the examples of Section 4 can be found on GitHub.

4.1 Future Direction & Open Problems

- Adapt [3] or [33] to work on non-transitive groups. This would make Algorithm 21 work on any left quasigroup.
- Attempt to define the theoretical complexity of Algorithms 21 and 22 and compare it with Algorithm 17.
- In order to make algorithms such as Algorithm 10 faster, hashing functions with better collision rates for sets of integers are needed.
- Ways to reduce the search space for automorphisms of a left quasigroup are needed, this would make Algorithm 7 even faster.
- Implement `LeftTransversal(`G`,` H`)` mentioned in Algorithm 1, to make the construction of coset quandles faster.

Bibliography

[1] Adams, C., Devadoss, J., Elhamdadi, M., and Mashaghi, A. Knot theory for proteins: Gauss codes, quandles and bondles. *Journal of Mathematical Chemistry 58*, 8 (2020), 1711–1736.

[2] Andruskiewitsch, N., and Grana, M. From racks to pointed hopf algebras. *Advances in Mathematics 178*, 2 (2003), 177–243.

[3] Atkinson, M. D. An algorithm for finding the blocks of a permutation group. *Mathematics of Computation 29*, 131 (1975), 911–913.

[4] Bianco, G., and Bonatto, M. On connected quandles of prime power order. *Beiträge zur Algebra und Geometrie/Contributions to Algebra and Geometry 62*, 3 (2021), 555–586.

[5] Bonatto, M. Principal and doubly homogeneous quandles. *Monatshefte für Mathematik 191*, 4 (2020), 691–717.

[6] Bonatto, M. Mal'cev classes of left quasigroups and quandles. *Quasigroups and Related Systems 29*, 2 (2021), 177–192.

[7] Bonatto, M. Connected quandles of size pq and 4p. *Osaka Journal of Mathematics 59*, 1 (2022), 145–175.

[8] Bonatto, M. Medial and semimedial left quasigroups. *Journal of Algebra and its applications 21*, 02 (2022), 2250021.

[9] Bonatto, M., and Stanovsky, D. Commutator theory for racks and quandles. *Journal of the Mathematical Society of Japan 73*, 1 (2021), 41 – 75.

[10] Borwein, J. M., Bailey, D. H., and Girgensohn, R. *Experimentation in mathematics: Computational paths to discovery*. AK Peters/CRC Press, 2004.

[11] Bosma, W., Cannon, J., and Playoust, C. The magma algebra system i: The user language. *Journal of Symbolic Computation 24*, 3 (1997), 235–265.

[12] Burris, S., and Sankappanavar, H. *A Course in Universal Algebra*. Graduate Texts in Mathematics. Springer New York, 1981.

[13] Crans, A. S. *Lie 2-algebras*. PhD thesis, 2004.

[14] de Ara'ujo, J. M., Pereira, R. B., Bentz, W., Chow, C., Ramires, J. H. V., Sequeira, L., and Sousa, C. Cream: a package to compute [auto, endo, iso, mono, epi]-morphisms, congruences, divisors and more for algebras of type $(2^n, 1^n)$.

[15] Elhamdadi, M., and Nelson, S. *Quandles*, vol. 74. American Mathematical Soc., 2015.

[16] Etingof, P., Soloviev, A., and Guralnick, R. Indecomposable set-theoretical solutions to the quantum yang–baxter equation on a set with a prime number of elements. *Journal of Algebra 242*, 2 (2001), 709–719.

[17] Freese, R. Computing congruences efficiently. *Algebra universalis 59*, 3 (2008), 337–343.

[18] Freese, R., McKenzie, R., et al. *Commutator theory for congruence modular varieties*, vol. 125. CUP Archive, 1987.

[19] Ho, B., and Nelson, S. Matrices and finite quandles. *Homology, Homotopy and Applications 7*, 1 (2005), 197–208.

[20] Holt, D. F., Eick, B., and O'Brien, E. A. *Handbook of computational group theory*. Chapman and Hall/CRC, 2005.

[21] Hulpke, A., Stanovskỳ, D., and Vojtěchovskỳ, P. Connected quandles and transitive groups. *Journal of Pure and Applied Algebra 220*, 2 (2016), 735–758.

[22] Jones, V. F. R. On knot invariants related to some statistical mechanical models. *Pacific Journal of Mathematics 137*, 2 (1989), 311 – 334.

[23] Joyce, D. A classifying invariant of knots, the knot quandle. *Journal of Pure and Applied Algebra 23*, 1 (1982), 37–65.

[24] Kauffman, L., Fenn, R., and Manturov, V. Virtual knot theory–unsolved problems. *Fund. Math 188* (2005), 293–323.

[25] Kauffman, L. H. Statistical mechanics and the jones polynomial.

[26] Kauffman, L. H. Introduction to virtual knot theory. In *Introductory Lectures On Knot Theory: Selected Lectures Presented at the Advanced School and Conference on Knot Theory and Its Applications to Physics and Biology* (2012), World Scientific, pp. 502–541.

[27] Linton, S. Gap: Groups, algorithms, programming. *ACM Commun. Comput. Algebra 41*, 3 (sep 2007), 108–109.

[28] Lopes, P., and Roseman, D. On finite racks and quandles. *Communications in Algebra® 34*, 1 (2006), 371–406.

[29] Matveev, S. V. Distributive groupoids in knot theory. *Matematicheskiĭ Sbornik. Novaya Seriya 119*, 1 (1982), 78–88.

[30] Nagy, G. P., and Vojtechovskỳ, P. Loops: Computing with quasigroups and loops in gap. *Online: http://web. cs. du. edu/petr/loops* (2007).

[31] Nelson, S. Signed ordered knotlike quandle presentations. *Algebraic & Geometric Topology 5*, 2 (2005), 443–462.

[32] Nelson, S. A polynomial invariant of finite quandles. *Journal of Algebra and Its Applications 7*, 02 (2008), 263–273.

[33] Schönert, M., and Seress, Á. Finding blocks of imprimitivity in small-base groups in nearly linear time. In *Proceedings of the international symposium on Symbolic and algebraic computation* (1994), pp. 154–157.

[34] Seress, . *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003.

[35] Smith, J. D. H. Mal cev varieties. *Lecture Notes in Mathematics 554* (1976).

[36] Stanovský, D. *Left Distributive Left Quasigroups*. PhD thesis, Charles University, Prague, 2004.

[37] Turaev, V. The yang-baxter equation and invariants of links. *New Developments in the Theory of Knots 11* (1990), 175.

[38] Vendramin, L. Rig, a gap package for racks, quandles and nichols algebras.

[39] Warner, S. *Modern algebra*. Courier Corporation, 1990.

[40] Wu, F. Y. The yang-baxter equation in knot theory. *International Journal of Modern Physics B 7*, 20n21 (1993), 3737–3750.