Quandle Unification Reduces to Quandle Matching

Maximus Redman
Bard College

Quandle Unification Reduces to Quandle Matching

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Maximus Redman

Annandale-on-Hudson, New York
May, 2023

Abstract

The purpose of this project is to show that quandle unification reduces to quandle matching. We show that any unification problem U can be embedded into a matching problem M. If a unifier exists for M, then the narrowing sequence N of that unifier can be refactored to N' such that all of its idempotence narrowing steps involving substitutions are done last. At the moment when the narrowing sequence N' needs only to do idempotence steps to unify M, we show that this implies a form for the embedded matching problem U such that U is guaranteed to have a unifier. This result also verifies that a quandle unification algorithm which uses the embedding technique is solving unification and not just matching.

Acknowledgments

I'd like to thank my advisors Bob McGrail and Caitlin Leverson for all the guidance they provided on this project. Without Bob, this paper would be much shorter and much less interesting. Without Caitlin, this paper may not have gotten finished at all. I would also like to thank the math and computer science departments at Bard as a whole for being so passionate about teaching, and for giving me so many different opportunities for research.

Dedication

First and foremost, I dedicate this paper to all the family members who have supported me most in my academic journey throughout the years–Helena, Jason, Jennifer, Andy, and Christine to name a few. Each of them were instrumental in shaping me to be the person who I am today, and in encouraging me to pursue higher education. I dedicate this paper to my brother Rasmus, who is one of the smartest people I know, and who always took care of me. I also dedicate this paper to all the wonderful friends and people I have met in my time at Bard, and a special thanks to my best friend Vigilance Brandon, who may be the only non-math major that actually understands what a quandle is.

1
Introduction

The topic of this paper is a proof that the quandle unification problem reduces to the quandle matching problem, which verifies the unification algorithm derived by Goldstein in 2021 [11]. The goal of chapter 2 is to establish the connection between knot theory and the quandle axioms, referencing Joyce [6], Reidemeister [9], and Nelson [7]. We also present additional quandle identities known as delta rules. The delta rules are instrumental for the term rewriting system that this paper largely focuses on–established in McGrail's 2018 paper [1]. In chapter 3 a number of key definitions from universal algebra are introduced, as well as properties of reduction and term rewriting system. Much of this chapter references Baader and Nipkow's book [2]. Chapter 4 introduces the decidable problem of syntactic unification, citing Martelli and Montanari [3]. It then presents the problems of equational matching and unification. In chapter 5, the logic programming language Prolog is briefly covered using Sterling and Shapiro's book [5] for the purposes of understanding how a quandle unification algorithm was implemented by Goldstein [11]. Finally, chapter 6 references Gallier and Snyder [4] as well as Dougherty and Johann [10] to provide a definition for narrowing and its use in the problem of equational unification. We then prove a result about refactoring narrowing sequences, as well as the general case of embedding a unification problem into a matching problem. We end by showing that we can reduce quandle unification to quandle matching.

2
Quandles and their Relationship to Knots

2.1 What is a Quandle?

Quandles are an algebraic structure that have recently become a subject of study.

Definition 2.1.1. A **quandle** is a set S with a binary operation $*$ that satisfies the following axioms [6].

1. For all $x \in S$ we have that $x * x = x$. (Idempotence)
2. For all $x, y \in S$, there exists a unique $z \in S$ such that $x = z * y$. (Right-cancellation)
3. For all $x, y, z \in S$, we have that $(x * y) * z = (x * z) * (y * z)$. (Right-distributivity)

△

Axiom 2 is equivalent to the existence of right inverses [7]. In other words, a quandle Q has an operation / where $(x * y)/y = x$ for all elements $x, y \in Q$. A quandle can also equivalently be defined with left inverses and left distributivity. While being interesting on their own, quandles are also useful at modeling other mathematical objects. For instance, quandles have a connection to knots, knot diagrams, and their isotopy-preserving transformations.

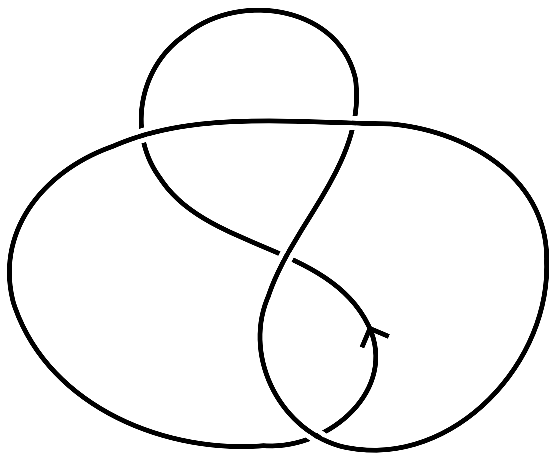

Figure 2.2.1: An example of an oriented knot diagram.

2.2 Oriented Knot Diagrams

Definition 2.2.1. A **knot** is an embedding of the circle in $\mathbb{R}^3$. In other words, it is a continuous function $K : [0,1] \to \mathbb{R}^3$ that is injective everywhere except for the endpoints [8]. △

A knot is not injective at its endpoints because they have to meet to create a loop. So given a knot $K : [0,1] \to \mathbb{R}^3$, we have

$$K(0) = K(1).$$

Definition 2.2.2. We say two knots K_1 and K_2 are **ambient isotopic**—or equivalent—if there exists a continuous function $H : \mathbb{R}^3 \times [0,1] \to \mathbb{R}^3$ such that

1. For each $t \in [0,1]$ the mapping taking $x \in \mathbb{R}^3$ to $H(x,t) \in \mathbb{R}^3$ is a homeomorphism of $\mathbb{R}^3$ onto itself

2. $H(x,0) = x$ for all $x \in \mathbb{R}^3$

3. $H(K_1,1) = K_2$).

In the case above, we call H an **ambient isotopy**. △

One useful way to represent knots is a knot diagram. A knot diagram is a projection of a knot onto a 2 dimensional plane with two constraints. As long as no strands of a knot lie directly on top of each other and no points exist where three strands cross each other, then a knot projection

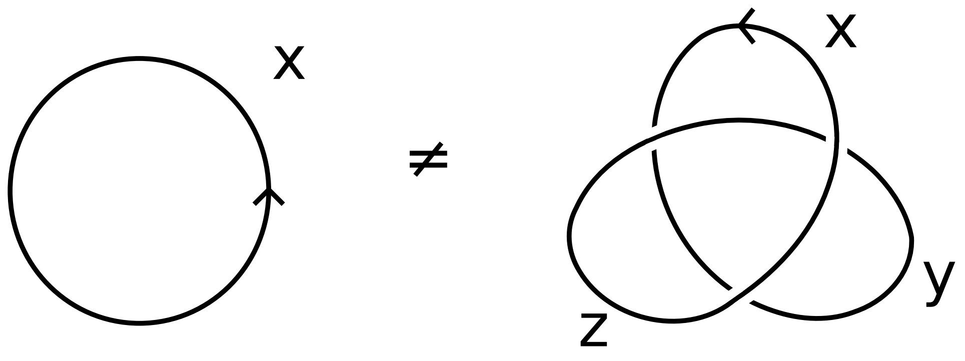

Figure 2.2.2: The trefoil knot (right) is the simplest knot which is not isotopic to the unknot (left).

is a valid knot diagram. The information contained in a knot diagram uniquely characterizes a knot up to isotopy. By recording the over/under crossings of a knot, that is enough information to determine its isotopy class. For the purposes of quandles, we deal with oriented knots. This means that for a certain strand, we give it an orientation. Since knots are embeddings of S_1, then we only need to assign a direction to one unbroken strand and follow it until it joins back with itself to determine the orientations of all the strands.

One way to know if two knots are ambient isotopic is if there exist a series of local, invariant-preserving steps that take one knot to the other. There are three of such moves, known as the Reidemeister moves [9]. If there is a series of Reidemeister moves that takes one knot to another knot, then they are ambient isotopic.

2.3 The Quandle Operations on Knot Diagrams

Before we can discuss Reidemeister moves and their connection to the quandle axioms. We first define the $*$ and / operations of quandles in the context of knot diagrams. Given some knot diagram, if a strand x crosses under a strand y from left to right, then the resulting strand is labeled $x * y$ [6]. Similarly, if a strand x crosses under a strand y from right to left, then the resulting strand is labeled x/y. Notice that if we reverse the orientation of a knot, then each $*$ operation becomes /, and vice-versa.

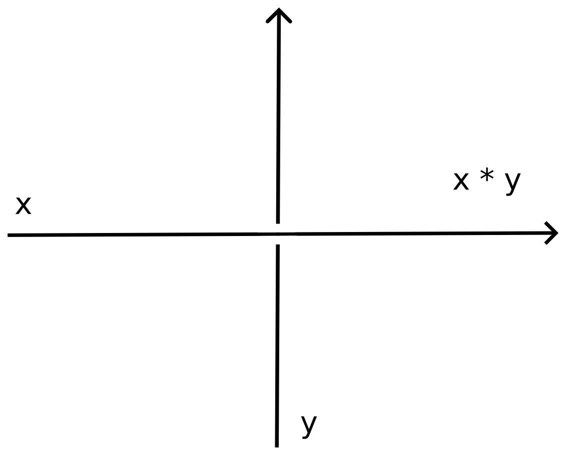

Figure 2.3.1: A crossing in an oriented knot that corresponds to the $*$ operation of a quandle.

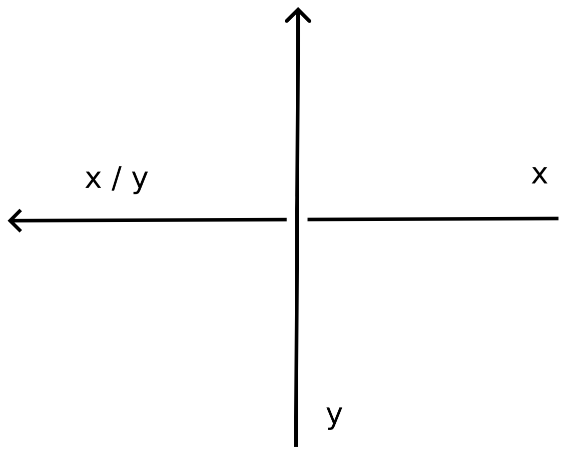

Figure 2.3.2: A crossing in an oriented knot that corresponds to the / operation of a quandle.

2.4 Type I Reidemeister Move

The Type I Reidemeister move corresponds to the first quandle axiom, also known as idempotence. On the left we see that the strand x crosses under itself from left to right, which leads to the strand y. Therefore, $x * x = y$. However, note that the end of the strand corresponding to y on the left diagram is equivalent to the end of the strand corresponding to x on the right. Therefore, $x = y$, and so

$$x * x = x. \tag{2.4.1}$$

Notice that by reversing the orientation of our diagram, we then know that

$$x/x = x. \tag{2.4.2}$$

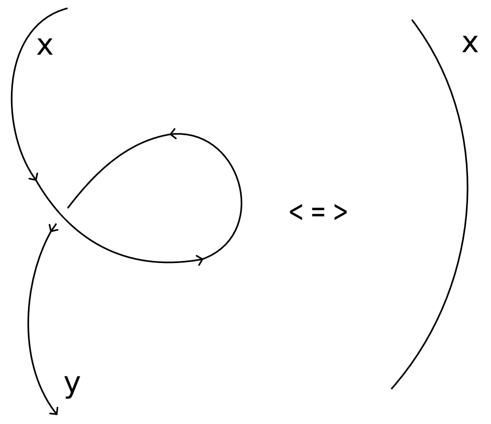

Figure 2.4.1: A Type I Reidemeister move.

The idempotence of knot theory is made very intuitive by the knot diagram. A loop formed by a strand crossing over or under itself is the same as a single strand without that loop. One can imagine pulling the strand “taught”, which would remove the loop.

2.5 Type II Reidemeister Move

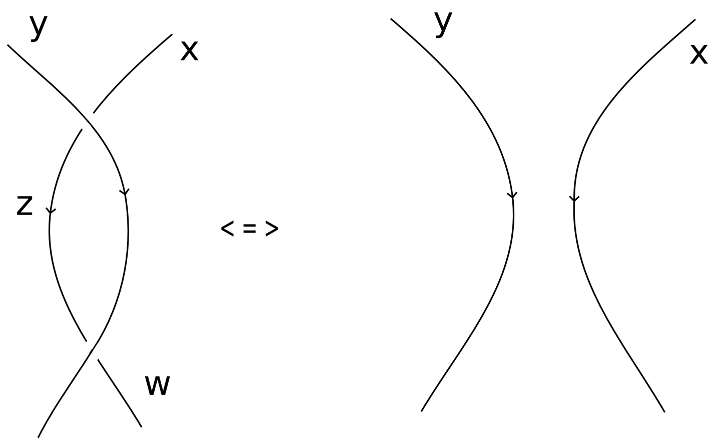

Figure 2.5.1: A demonstration of a Type II Reidemeister move.

The Type II Reidemeister move says that if two strands are stacked without any complicating crossings, then we can pull those two strands apart. This corresponds to the second quandle axiom of right-cancellation. On the left diagram, strand z is formed by x crossing under y from left to right, so $z = x * y$. Additionally, the strand w is formed by z crossing under y from right to left, and so $w = z/y = (x * y)/y$. Notice, however, that the end assigned to w on the left knot diagram is equivalently x on the right diagram. Therefore $w = x$, and by reversing the orientation we have:

$$(x * y)/y = x \tag{2.5.1}$$

$$(x/y) * y = x. \tag{2.5.2}$$

2.6 Type III Reidemeister Move

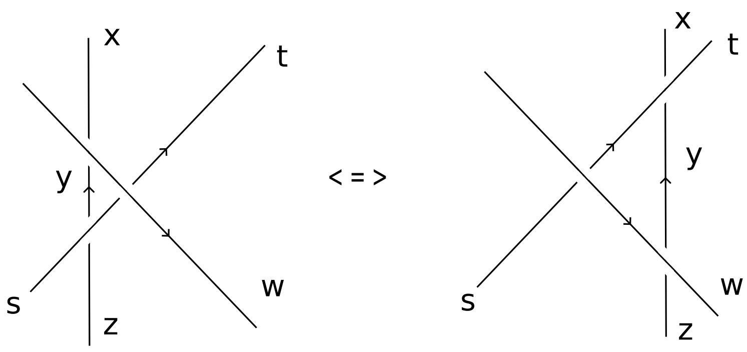

Figure 2.6.1: A demonstration of a Type III Reidemeister move.

The Type III Reidemeister move corresponds to the third quandle axiom of associativity. Notice that every under crossing in both diagrams are right to left. In other words, each operation will be /. Therefore in the left diagram we have

$$\begin{aligned} x &= y/w \\ &= (z/s)/w. \end{aligned}$$

In the right diagram, notice that $t = s/w$ and $y = z/w$. So,

$$x = (z/s)/w = (z/w)/(s/w).$$

As in the previous two cases, we can also reverse the orientation to get the following two properties:

$$(x * y) * z = (x * z) * (y * z) \tag{2.6.1}$$

$$(x/y)/z) = (x\ z)/(y/z). \tag{2.6.2}$$

The third Reidemeister move is a bit harder to understand, but it still has a fairly intuitive logic. If a strand crosses under two other strands who are also crossing, then we can move the bottom strand without affecting the relation the other two strands have.

2.7 Additional Quandle Identities

The following are direct results of the quandle axioms in Definition 2.1.1:

Theorem 2.7.1. *Let Q be a quandle. Then for all $x, y, z \in Q$, the following are true.*

1. $x * (y * z) = ((x/z) * y) * z$
2. $x * (y/z) = ((x * z) * y)/z$
3. $x/(y * z) = ((x/z)/y) * z$
4. $x/(y/z) = ((x * z)/y)/z$.

We refer to these identities as the ***delta rules****.*

Proof. These derivations are all directly from McGrail's paper, but they are short enough to put here, and it will be useful to familiarize oneself with the style of algebraic manipulations one does in quandle systems [1].

1.

$$\begin{aligned} x * (y * z) &= ((x/z) * z) * (y * z) \\ &= ((x/z) * y) * z. \end{aligned}$$

Here we use the right-cancellation axiom of quandles to substitute in $(x/z) * z$ for x, and then extract z from both subterms using right-distributivity.

2.

$$\begin{aligned} x * (y/z) &= ((x * (y/z)) * z)/z \\ &= ((x * z) * ((y/z) * z)/z) \\ &= ((x * z) * y)/z. \end{aligned}$$

Notice that each of these rules involve repeated application of introducing new terms with right-cancellation and then distributing to get helpful new representations.

3.

$$\begin{aligned} x/(y*z) &= ((x/z)*z)/(y*z) \\ &= ((((x/z)/y)*y)*z)/(y*z) \\ &= ((((x/z)/y)*z)*(y*z))/(y*z) \\ &= ((x/z)/y)*z \end{aligned}$$

4.

$$\begin{aligned} x/(y/z) &= ((x*z)/z)/(y/z) \\ &= ((((x*z)/y)*y)/z)/(y/z) \\ &= ((((((x*z)/y)/z)*z)*y)/z)/(y/z) \\ &= ((((x*z)/y)/z)*(y/z))/(y/z) \\ &= (((x*z)/y)/z). \end{aligned}$$

□

These new identities will be important for a term rewriting system for quandles. The next section covers what reduction systems are and how we can use the delta rules along with idempotence and right-cancellation identities to get a term rewriting system.

3
Term-Rewriting and Reduction Systems

This chapter will introduce the notion of term rewriting as a computational model. A term rewriting system differs from equational theory in that equations of a term rewriting system are used as directed replacement rules. For the rest of the paper, the convention unless otherwise specified for term rewriting system identities will be that the left-hand side can be replaced by the right-hand side, but not vice-versa.

3.1 Terms, Operations, and Algebras

We will be introducing a number of notation from universal algebra which will be useful for the purposes of studying quandle algebra. Universal algebra is the study of algebraic structures. The more well-known algebraic structures include groups, rings, and fields. All of the standard definitions in this section and Section 3.2 are taken from "Term Rewriting and all that" By Franz Baader and Tobias Nipkow[2].

Definition 3.1.1. A **signature** Σ is a set of function symbols, where each $f \in \Sigma$ is associated with a non-negative integer n known as its arity. The **arity** is the number of arguments that a function symbol takes. We denote the set of all $f \in \Sigma$ with arity n with $\Sigma^{(n)}$. We refer to 0-ary elements as **constants**. △

The set of n-ary functions where $n > 0$ can be thought of as symbols for actual functions, whereas all symbols for 0-ary functions can be thought of as constants in the common understanding.

Let's make a simple signature Σ for a quandle. The quandle axioms tell us that it will always have two binary operations, $*$ and $/$. Since we want elements to act on, our signature will also include constants a, b, and c. Recall that constants are simply 0-ary—sometimes called nonary—functions. Thus we have that

$$\Sigma = \{a, b, c, *, /\}.$$

Using our signature, we can construct terms such as $a * a$ or $(b/a) * (a/c)$. These aren't the only kind of terms we can make, however. We can also make terms that contain variables, which are a special kind of symbol. Let's precisely define what we mean by a term and a variable.

Definition 3.1.2. Let Σ be a signature and X a set of variables symbols such that $\Sigma \cap X = \emptyset$. We define the set of all terms $T(\Sigma, X)$ inductively. We have that

1. $x \in T(\Sigma, X)$ if $x \in X$,

2. For all $n \in \mathbb{Z}_{\geq 0}$, we have $f(t_1, \ldots, t_n) \in T(\Sigma, X)$ for all $f \in \Sigma^{(n)}$ and $t_1, \ldots, t_n \in T(\Sigma, X)$.

A **term** t is simply an element of $T(\Sigma, X)$, and a **variable** is a symbol for which any other term can be substituted in. $\triangle$

Note that in order to substitute a term in for a variable, we also need to have a signature defined.

Example 3.1.3. Let's continue with our example above and use the same signature $\Sigma = \{a, b, c, *, /\}$. Now let our variable set be $V = \{X, Y, Z\}$. Besides our constants, notice that quandles only have binary operators. Here are some examples of valid terms given our signature and variable set.

1. $a * Z$

2. $a/(b/(X * a)/a)$

3. $Y * Y * Z * (a/Z) * (Y/b)$.

Notice that every instance of a variable symbol is representing the same universally quantified object. This means if we substitute something in for a variable, we must replace every instance of that variable in the term with the substituted value. Suppose we want to apply the substitution $[Y \to (Z/c)]$ to the third example term above. Recall that variables can be instantiated for any term in $T(\Sigma, V)$, including terms with other variables. So if we apply our substitution $[Y \to (Z/c)]$, our term would become

$$(Z/c) * (Z/c) * Z * (a/Z) * ((Z/c)/b).$$

$\lozenge$

Definition 3.1.4. Let Σ be a signature and X a set of variable symbols disjoint from Σ. Then a **substitution** is a function $\sigma : X \to T(\Sigma, X)$. The set of variables that σ doesn't map to themselves again is called the **domain**. If a substitution changes a variable, we say it **instantiates** that variable. A substitution can easily be turned into a mapping from one term to another. Given our substitution σ from above, we can define a mapping $\phi : T(\Sigma, X) \to T(\Sigma, X)$ such that, for $t \in T(\Sigma, X)$, we have

$$\phi(t) = \begin{cases} \sigma(t) & \text{if } t \in X \\ f(\phi(t_1), \dots, \phi(t_n)) & \text{if } t = f(t_1, \dots, t_n). \end{cases} \tag{3.1.1}$$

Notice that if t is not a variable, then t is guaranteed to take the form of $f(t_1, \dots, t_n)$ where $f \in \Sigma^{(n)}$ and $t_1, \dots, t_n \in T(\Sigma, X)$ for some $n \in \mathbb{Z}_{\geq 0}$. $\triangle$

For the rest of the paper, when we talk about substitutions, we are usually referring to this process of extending a substitution into a mapping from a term to another term. Notice that a substitution leaves all ground sub-terms unaffected, and some variables as well. In other words, it's enough to define a substitution σ by the set of variables V where $\sigma(v) \neq v$ for all $v \in V$.

Definition 3.1.5. Let σ and ϕ be substitutions with domain $T(\Sigma, V)$ for some signature Σ and variable set V. The **composition** $\sigma\phi$ of these two substitutions is defined as

$$\sigma\phi(x) = \sigma(\phi(x))).$$

$\triangle$

Example 3.1.6. Let σ and ϕ be substitutions for $T(\Sigma, V)$ as defined in Example 3.1.3. Suppose σ only sends Y to X/b and ϕ only sends X to $a * a$. Notice that the order in which we apply these substitutions matters. In other words it is not always the case that $\sigma\phi(t) = \phi\sigma(t)$ for a general $t \in T(\Sigma, V)$. Consider the term $t = (X * b)/(Y * b)$. Notice that

$$\sigma\phi(t) = ((a * a) * b)/((X/b) * b),$$

and

$$\phi\sigma(t) = ((a * a) * b)/(((a * a)/b) * b).$$

For this reason, we need to be careful when composing substitutions. $\diamondsuit$

Remark 3.1.7. For composition of substitutions that all instantiate a single variable only, we have a special notation which proves to be very useful. Using our usual signature and variable sets, suppose we have a series of substitutions s_1, s_2, and s_3 that respectively send X to a, Y to b, and Z to c. Then we can represent the composition of these functions as

$$[X \to a, Y \to b, z \to c].$$

We apply these substitutions left to right, so this notation represents the composition $s_3 \circ s_2 \circ s_1(x)$. We frequently use this notation because narrowing steps—a method for solving unification problems—involve instantiating a single variable at each step. For more information see Chapter 6 $\diamondsuit$

Definition 3.1.8. A term is called **ground** if it contains no variables. $\triangle$

Ground terms are important because there are no substitutions which map them to a different term (since there are no variables to instantiate).

3.2 Reduction terminology.

A reduction system $(A, \to)$ consists of a set of elements A and a binary relation $\to$ on A. Instead of saying $(a, b) \in \to$, we say that $a \to b$, or "a reduces to b". We say that $a \xrightarrow{*} b$ if there exists $r_1, r_2, \dots, r_n \in A$ such that

$$a \to r_1 \to r_2 \to \dots \to r_n \to b.$$

Another way to say this is that a path of reductions exists from a to b, or a **transitively reduces** to b.

Definition 3.2.1. Let $(A, \to)$ be a reduction system. Then $\to$ is **terminating** if there does not exist some infinite chain of reductions $a_1 \to a_2 \to a_3 \to \dots$ where $\{a_1, a_2, a_3, \dots\} \subseteq A$. $\triangle$

Note that a terminating reduction does not mean that the set it acts on is finite. It is enough that for each element of some infinite set, any chain of reductions starting from that element is finite.

Definition 3.2.2. Let $(A, \to)$ be a reduction system. Then $x \in A$ is **reducible** if there exists $y \in A$ such that $x \to y$. An element of a reduction system x is in **normal form** if it is not reducible. We also sometimes say that x is **reduced**. $\triangle$

Note that if a rewrite system is terminating, that means each of its elements has a normal form. Normal forms are not inherently unique, however. For that, we need a stronger constraint.

Definition 3.2.3. Let $(A, \to)$ be a reduction system. Then $a \in A$ is **confluent** if and only if for all distinct $b, c \in A$ such that $x \xrightarrow{*} b$ and $a \xrightarrow{*} c$, there exists $d \in A$ such that $b \xrightarrow{*} d$ and $c \xrightarrow{*} d$. If a term rewriting system is terminating and confluent, we say that it is **convergent**. $\triangle$

Importantly, a convergent term rewriting system is enough to guarantee unique normal forms for its elements.

Theorem 3.2.4. *Let $(A, \to)$ be a reduction system. If it is convergent, then each element in A will have a unique normal form.*

Proof. Let $(A, \to)$ be a terminating an confluent reduction system. Let $a \in A$. Since $\to$ is terminating, we know that a will have normal forms. Let m and n be normal forms of a. For a contradiction, suppose $m \neq n$. As m and n are normal forms of a, we know $a \xrightarrow{*} m$ and $a \xrightarrow{*} n$. By definition 3.2.3, m and n being distinct implies there exists $l \in A$ such that $m \xrightarrow{*} l$ and $n \xrightarrow{*} l$. Since m and n are normal forms, they are not reducible, and therefore we have a contradiction. Thus we know

$$m = n.$$

□

With unique normal forms, we are closer to being able to create a decision procedure for evaluating the equality of two quandle terms.

Definition 3.2.5. A **rewrite rule** is an identity $l = r$ for terms l and r where an instance of a certain variable in r implies it also exists in l. In other words, the simplification from l to r doesn't introduce any new variables. A **term rewriting system** is a set of rewrite rules. A **redex** is an instance of a subterm that matches with the left-hand side of a rewrite rule. △

Note that, given some term rewriting system T, the reduction system $\to_T$ will be well-defined [2]. Therefore when we say things like "T is confluent", we are actually referring to the reduction system induced by T. Also, terms can be in normal form—or reduced—relative to T, since we just mean the reduction system induced by T.

3.3 A Quandle Term Rewriting System

Here we present a terminating and confluent rewriting system for the first order equational theory of quandles defined in Definition 2.1.1. This rewriting system was first proposed by Mcgrail et al. in 2018 [1].

Definition 3.3.1. Let Q be a quandle. Let x, y, z be elements of Q. Then our term rewriting system R consists of the following rules.

R.1 $x * x \to x$

R.2 $x/x \to x$

R.3 $(x * y)/y \to y$

R.4 $(x/y) * y \to x$

R.5 $x * (y * z) \to ((x/z) * y) * z$

R.6 $x * (y/z) \to ((x * z) * z)/z$

R.7 $x/(y * z) \to ((x/z)/y) * z$

R.8 $x/(y/z) \to ((x * z)/y)/z$.

Recall that the first four rules correspond to the idempotence and right-cancellation axioms of quandles in Definition 2.1.1. The remaining four are the delta rules from Theorem 2.7.1. Unless specified otherwise, when we refer to R we are referring to the above set of rewrite rules.

Theorem 3.3.2. *The rewrite system R is both terminating and confluent, and so it is convergent* [1]*. Therefore it generates unique normal forms for all quandle terms.*

△

Example 3.3.3. Let's use our quandle Q defined above to develop intuition for how R simplifies terms to their normal form. Our signature and variable set will be the same Σ and V from earlier examples. Given the term

$$(((c * c)/(A * b)) * (A * b) * (a * c)),$$

we can outline the reductions that our rewrite system R will make to carry it to a normal form.

$$\begin{aligned}
(((c * c)/(A * b)) * (A * b)) * (a * c) &= ((c/(A * b)) * (A * b)) * (a * c) && \text{(R.1)}\\
&= c * (a * c) && \text{(R.4)}\\
&= ((c/c) * a) * c && \text{(R.5)}\\
&= (c * a) * c. && \text{(R.2)}
\end{aligned}$$

Notice that each step reduces the size of the overall term except for the delta rule. The first four rules corresponding to idempotence and right-cancellation reduce the total number of operators in the term, however the four delta rules actually increase the number of operators. The size of this increase has an exponential relationship to the number of operators in the right term of a delta rule. As an example, let's look at the normal form of the term $(a * (b * (c/X)))$. Notice that each left subterm is a constant whereas—until we've reached X—the right subterms are all binary operations on other terms. We call this form a **right-leaning** term due to the fact that all of the closing parentheses appear on the right end of the term. In this case, we only need to successively apply delta rules to fully simplify.

$$\begin{aligned}(a * (b * (c/X))) &= ((a/(c/X) * b)) * (c/X) && \text{(R.5)}\\ &= ((((a * X)/c)/X) * b) * (c/X) && \text{(R.8)}\\ &= ((((((a * X)/c)/X) * b) * X) * c)/X && \text{(R.6)}\end{aligned}$$

The original term, $(a*(b*(c/X)))$, has 3 operators and the resulting term has 7. In general, if we have n operators in a right-leaning term that requires no other reductions other than those from delta rules, then the number of operators after carrying out the delta rules will be $2^n - 1$. ♢

Now we have some intuition for how our term rewriting system arrives at a normal form. While the idempotence and right cancellation rules are intuitively simplifying since they reduce the total number of operators, it is not clear why the delta rules are simplifying. For this reason, we give a simplified version of the argument that Mcgrail used to show that the term rewriting system R will eventually always terminate. Please see their full paper for more details [1]. Note that we will be using $\circ$ as an unspecified quandle operator for the remainder of this paper. In other words, $\circ \in \{*, /\}$ and furthermore

$$\circ' = \begin{cases} * & \text{if } \circ = / \\ / & \text{if } \circ = *. \end{cases} \tag{3.3.1}$$

Theorem 3.3.4. *The term rewriting system R in Definition 3.3.1 is terminating.*

Proof. Let S be a set of nonary function symbols. Let T our set of terms over the signature $T_Q = \{*, /\} \cup S$. Now we define a measure function $m : R \to \mathbb{N}$ such that

$$m(t) = \begin{cases} 0, & \text{if } t \text{ is a variable or constant,} \\ 1 + m(t_1) + 3m(t_2), & \text{if } t = t_1 \circ t_2. \end{cases} \tag{3.3.2}$$

Now we want to show that, given some term t, the overall measure of t will reduce with each rewrite step. If we can show this, then the fact that the measure of a term is bounded from below by 0 tells us that the measure of any sequence of rewrite steps on t will eventually converge to some value. At this point we will be unable to simplify further, and so any sequence of reduction steps will be finite. It is enough to show that the right-hand side of each rewrite rule has measure smaller than the left-hand side. For idempotence and right cancellation rules, showing the left-hand side has smaller measure is trivial. Thus we only need to consider the general form of the delta rule

$$r \circ_1 (s \circ_2 t) \to ((r \circ_2^{-1} t) \circ_1 s) \circ_2 t.$$

for arbitrary terms r, s, and t. So,

$$\begin{aligned} m(r \circ_1 (s \circ_2 t)) &= 1 + m(r) + 3m(s \circ_2 t) \\ &= 4 + m(r) + m(s) + m(t), \end{aligned}$$

and

$$\begin{aligned} m(((r \circ_2^{-1} t) \circ_1 s) \circ_2 t) &= 1 + m((x \circ_2^{-1} z) \circ_1 s) + 3m(z) \\ &= 3 + m(r) + m(s) + m(t). \end{aligned}$$

Thus each delta rule reduction decreases the overall measure of whatever term is being delta-reduced. Now we have that each reduction step strictly reduces the measure, and so any sequence of reduction steps on a term is strictly decreasing. Since it is also bounded from below by zero, we now know that any sequence of reduction steps will be finite, and thus R is terminating. □

Recall that, in Example 3.3.3, when our term rewrite system terminated it ended up pushing all the opening parentheses to the left. Only in this form—the opposite of the right-leaning term—are we unable to do any more delta rules.

Definition 3.3.5. A term t is **delta reduced** if it is of the form

$$t = (\ldots(t_0 \circ_1 t_1) \circ_2 t_2) \ldots) \circ_n t_n,$$

where $t_1, t_2, \ldots, t_n$ are all either variables or constants. Note that the left ellipsis represents $n-3$ open parentheses symbols. The right ellipsis, on the other hand, represents the application of $n-3$ more operators and terms, beginning with $\circ_3 t_3$ up to $\circ_{n-1} t_{n-1}$. We will use this convention of ellipses in reduced terms for the remainder of the paper. We say this term is delta reduced because we cannot simplify it further through the use of delta rules. A reduced term is by definition also delta reduced. $\triangle$

3.4 A Term Rewriting Decision Procedure

A term rewriting system that guarantees unique normal forms is useful because it creates a simple method for determining if two expressions are in the same equivalence class under some theory. For a precise definition of a theory, see Definition 4.2.1.

Algorithm 3.4.1. Let T_1 and T_2 be ground quandle terms. Suppose we are asked to determine whether or not T_1 equals T_2 under the quandle theory. Our decision procedure is as follows:

1. In T_1, search for a subterm T_s that matches the left-hand side of a rewrite rule in R.
2. Replace T_s with the right-hand side of the same rewrite rule in R.
3. Repeat steps 1 and 2 until there are no more possible reductions to be made to T_1.
4. Repeat steps 1-3 for T_2, at which point T_1 and T_2 will both be in normal form.
5. If T_1 is syntactically identical to T_2, then we return true. Otherwise we return false.

$\diamondsuit$

Notice that steps 3 and 5 of this algorithm rely on the existence of normal forms for expressions and their uniqueness respectively. Note that this decision procedure is only guaranteed to work if there are no variables. The presence of variables in one or both of the expressions will make this procedure unusable.

4 Unification

To show the ways in which variables can complicate deciding whether or not two terms are equal to each other, we will consider a simple example with terms from our usual signature and variable set Σ and V.

Example 4.0.1. Let us look at the expression

$$(a/(b*c))*X = ((a*b)/(a*b))/b$$

and try to determine if it is false or not. If we simplify both terms to their normal forms, the expression becomes

$$(((a/c)/b)*c)*X = a.$$

Clearly, the decision procedure in Algorithm 3.4.1 would say that these two are not identical, however we still have the variable X that we can instantiate. Notice that if we apply the substitution $[X \to (b*c)]$, the left-hand side of the expression becomes $(((a/c)/b)*c)*(b*c)$, which can be simplified further. So,

$$\begin{aligned} (((a/c)/b)*c)*(b*c) &= (((((a/c)/b)*c)/c)*b)*c && ((\text{R.5})) \\ &= (((a/c)/b)*b)*c && ((\text{R.3})) \\ &= (a/c)*c && ((\text{R.4})) \\ &= a. && ((\text{R.4})) \end{aligned}$$

Thus if we allow substitutions, then we know that the expression $(a/(b*c))*X = ((a*b)/(a*b))/b$ is true. $\Diamond$

For this reason, we will need a new decision procedure if we want to handle problems that involve variables, also known as unification.

Definition 4.0.2. The **unification problem** is about finding a set of substitutions for variables such that two terms are equationally equal to each other. If no such substitution set exists, we want to be able to determine this as well. Note that the reduced form of Example 4.0.1 is not an instance of general unification, but a subproblem known as **matching** where variables appear only on one side of the expression. $\triangle$

Before we look at equational unification, let us first look at an easier form of unification–one that does not allow any term rewriting.

4.1 Syntactic Unification

Definition 4.1.1. Syntactic unification is the process of trying to unify two expressions to make them syntactically identical. Expressions are **syntactically identical** if they contain exactly the same symbols in each position. $\triangle$

Example 4.1.2. Suppose we look at expressions using our usual signature $\Sigma = \{a, b, c, *, /\}$ and variable set $V = \{A, B, C\}$. Imagine that we are given the equation

$$X * x = Y * x$$

and asked if the two expressions are syntactically identical. In this case, we do not think of x and y as elements of some quandle, because under the problem of syntactic unification we cannot use the quandle axioms to simplify terms, and so the only option we have is to try and instantiate X and Y such that both sides are syntactically identical. Now consider the substitution

$$[Y \rightarrow X].$$

Then we have $X * x = (X) * x$, and now we can substitute in any term for X and these two expressions will be syntactically identical. Let $[X \to z]$, then $(z) * x = (z) * x$. The series of substitutions in order to make two terms syntactically identical is known as a **syntactic unifier**. In this case, our syntactic unifier is

$$[Y \to X, X \to z].$$

Importantly, the substitutions are read left to right. So we first substitute in X for Y, and then z for x. The astute among you will have noticed that the substitution $[Y \to X]$ leads to the expression $X * x = (X) * x$, which is already syntactically unified. In some sense, this unifier is "better", because it leaves us with a variable X to instantiate at some later point. The unifier $[Y \to X]$ is what is known as the most general unifier. $\Diamond$

Definition 4.1.3. Let $E_1 = E_2$ be some expression denoted E where E_1 and E_2 both contain variables. A unifier σ for E is said to be a **most general unifier** if, for all other unifiers ρ of E, there exists a unifier τ such that

$$\tau(\sigma(E)) = \rho(E).$$

$\triangle$

A useful feature of syntactic unification is that it is decidable [3].

Definition 4.1.4. A problem is **decidable** if there exists an algorithm guaranteed to solve it in finite time. $\triangle$

Logic programming languages like Prolog rely on the fact that syntactic unification is decidable, since they use syntactic unification to answer queries within programs [5]. Not only is it decidable, but linear time algorithms have been found for syntactic unification [3].

A sub-problem of syntactic unification is syntactic matching, where only one of the expressions contains variables. While we have been using the conventional operators and elements of quandles, syntactic unification problems function identically, irrespective of the symbols used to represent their different elements. What we want, then, is to make use of quandle axioms in order to simplify expressions.

4.2 Equational Unification

Definition 4.2.1. Equational Unification is the problem of trying to unify two terms such that they are equal under some theory. A **theory** is a fixed set of identities. △

Example 4.2.2. Suppose we had a theory E containing a commutative identity $f(s,t) = f(t,s)$. Now let $a, b \in S$ and let X be a variable. Suppose we were given the equation

$$f(a,b) = f(b,X)$$

and asked to solve it for syntactic unification first and equational unification second. While we cannot syntactically unify these two terms, we can make them equal under our above theory. Consider the substitution $[X \to a]$. Then

$$f(a,b) =_E f(b,(a)).$$

◊

An example of a theory is the quandle theory in Definition 2.1.1. We can use idempotence, right cancellation, and right distributivity identities to alter terms while still keeping them in the same equational equivalence class. Due to these new identities, equational unification is usually harder than syntactic unification, though that depends on the theory. Oftentimes it is much more likely that equational unification problems are not decidable.

Theorem 4.2.3. *The general form for a quandle unification problem that cannot be immediately rewritten to a matching problem is*

$$(\ldots(t_0 \circ_1 t_1) \circ_2 t_2) \circ_3 \ldots) \circ_n t_n = B,$$

where t_0 and B are variables and t_1 through t_n are either constants or variables.

Proof. Let $E_1 = E_2$ be a quandle expression such that E_1 and E_2 both contain variables. We can use the rewrite system R to make both terms reduced, and so our expression becomes

$$(\ldots(t_0 \circ_1 t_1) \circ_2 t_2) \circ_3 \ldots) \circ_n t_n = (\ldots(s_0 \circ_1 s_1) \circ_2 s_2) \circ_3 \ldots) \circ_m s_m.$$

where each s and t term is either a variable or a constant. Notice that we can isolate t_0 by right canceling the outermost term on the left-hand side. So,

$$t_0 = (\dots(s_0 \circ_1 s_1) \circ_2 s_2) \circ_3 \dots) \circ_m s_m) \circ_1^{-1} t_1) \circ_2^{-1} t_2) \circ_3^{-1} \dots) \circ_n^{-1} t_n.$$

If t_0 is a constant, then only the right-hand side contains variables, and so this expression reduces to a matching problem. We can similarly isolate s_0, and so if it is a constant then the expression will also reduce to a matching problem. Therefore, the general form for a quandle unification problem will be

$$(\dots(t_0 \circ_1 t_1) \circ_2 t_2) \dots) \circ_i t_i = B,$$

where t_0 and B are variables and t_1 through t_i are either constants or variables. $\square$

Note that all the identities in our term rewriting system R in Definition 3.3.1 come directly from the axioms of the quandle theory, and so we can use R to rewrite terms in the quandle unification problem. This still, however, does not tell us how to find substitutions for a unifier. Before we get to that, however, we briefly have to discuss the some implementation details of the quandle unification algorithm created by Goldstein in 2021 [11].

5 Prolog

As Prolog was the language that Goldstein chose to implement a unification algorithm, we will briefly cover its basics in this chapter [11]. Prolog is a logic programming language, and one of the earliest [5]. It is dynamically typed, and unlike most other programming languages it is primarily declarative. This means programming in Prolog involves designing a program such that it describes the logical constraints of some computation, as opposed to directly manipulating the state of a program through commands. A program in Prolog consists of a series of relations, either facts or rules. A computation is a query on these relations. Prolog only has a single data structure: the term. The definitions in this chapter are taken from "The Art of Prolog" by Leon Sterling and Ehud Shapiro. Please see the full book for more details [5].

5.1 Basic Definitions

Prolog consists of using terms and statements, which is taken from formal logic. The simplest statement in Prolog is a fact. A fact says that some relation holds between some number of objects. Some examples of facts are:

```
food(pizza).
less(two, five).
siblings(leah, collin, james).
```

Note that a fact can define a relation for any number of objects, including just one. Another name for a relation is a predicate. In this case, food, less, and siblings are all predicates. The objects predicates refer to are also known as atoms. The atoms in this case are pizza, two, five, leah, collin, and james. Importantly, predicates and atoms begin with lowercase letters. This is because objects beginning in uppercase letters are reserved for variables in Prolog. A finite collection of facts is known as a program. It can also be thought of as a database. Here is an example of a program:

```
father(lyle,james).
father(michael,noah).
father(michael,alice).
father(michael,anna).
father(ben,lyle).
father(ben,herbert).
father(ben,michael).
male(michael).
male(lyle).
male(noah).
male(james).
male(herbert).
male(ben).
female(sarah).
female(alice).
female(anna).
mother(sarah,james).
```

The query is the second kind of statement in Prolog. A query in Prolog asks if a relation holds between objects. If we take the above program to be our fact-base, then an example of a query we might ask is:

```
?- father(ben,michael).
true.
```

When a query is made in relation to a program, the query is the statement is a logical consequence of the program. In this case, father(ben, michael) is a fact in our program, and as such is immediately true. Suppose we wanted to know all the children of ben. To do that we can use variables. Variables are stand-ins for a single but unspecified value. Using variables we can pose the following query:

```
?- father(ben, X).
X = lyle ;
X = herbert ;
X = michael.
```

In this case, the query father(ben, X)? is asking whether or not there exists some value for X in the program that would make the statement true. We can also use variables to define universal facts. If we wanted to define multiplication for natural numbers with zero and without using variables, we would have to write

```
times(zero, one, zero).
times(zero, two, zero).
times(zero, three, zero).
```

and so on. With variables, we can summarize this infinite list of facts with a single fact:

```
times(zero, X, zero).
```

This fact reads *for all X, X times zero is zero*, so in other words the variable X is universally quantified. Now that we have defined variables, we can define terms.

Definition 5.1.1. A **term** is either a variable, constant, or a compound term. A compound term consists of a functor—which is an atom—and a number of terms as its arguments. △

Examples of compound terms are

```
country(somalia).
plus(two, three, five).
foo(X).
times(zero, X, zero).
times(2, times(2, 2), 8).
```

As stated previously, terms are the only data structure of Prolog, and they are very flexible for that reason. Now we can talk about the most powerful kind of statement in Prolog: the rule.

5.2 Rules, Backtracking, and Declarative Programming

Suppose we had some fact P. in our program. Notice that we could pose it as the query P? as well. Without specifying it as a fact or query, we refer to P on its own as a **goal**.

Definition 5.2.1. A **rule** in Prolog is a statement of the form

$$S \leftarrow G_1, G_2, \ldots, G_n.$$

Where the goal S is known as the **head**, and the **body** of the rule is made up of the goals $G_1, G_2, \ldots, G_n$ where $n > 0$. △

When combined with variables, rules can be used to capture more complicated logical relationships. For instance, we could define a daughter rule as:

```
daughter(X, Y) :- father(Y, X), female(X).
```

In words, this rule says that X is the daughter of Y if X is the father of Y and X is a female. The head of a rule is satisfied only if all the goals in the body can be satisfied as well. Note that a rule is the same as any other goal, and can be used in the body of other rules. Note that if we wanted to define a grandparent rule, it would take four different rules with alternating uses of the predicates father and mother. Instead, we can define a more general parent rule.

```
parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).
```

Now we have a much simpler time defining a grandparent rule:

```
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

Using our family program from above, we can now use rules to make more advanced queries.

```
?- grandparent(X, Y).
X = ben,
Y = james ;
X = ben,
Y = noah ;
X = ben,
Y = alice ;
X = ben,
Y = anna ;
```

Here our program found four instances where ben was a grandparent, but how Prolog searches the program space to find these solutions is also important. This is where we get to the topic of **backtracking**. Prolog will first search the program space for a constant with which to instantiate X, and then for some constant to instantiate Y such that the grandfather rule is true. In this case, the fact father(michael, noah). is the first fact from top-to-bottom of our program where X can be instantiated such that the first goal of the grandfather rule would be true. In other words, the first substitution Prolog makes is [$X \rightarrow$ michael]. The problem then becomes finding a substitution for Y such that the goal parent(michael, Y). is satisfied. The first substitution for Y we would get is $[T \rightarrow noah]$, giving us parent(michael, noah). The program will be unable to find a Z such that parent(noah, Z) is true, and so it will backtrack and try to find another instantiation of Y. At some point, it will run out of instantiations for Y and backtrack another step. At this point, it will find the substitution $[X \to \text{ben}]$. It is with this instantiation of X that we find our four solutions. Backtracking is an extremely powerful aspect of Prolog, and enables us to solve much more complicated logic problems.

5.3 Evaluation Algorithm

The simplest part of the unification algorithm is the sub-process of **evaluation**, which corresponds to the term rewrite system R for quandles [11]. The evaluation process consists of searching some expression for subterms that match the left-hand side of some rewrite rule and replacing them with the right-hand side of that same rewrite rule. This part of the algorithm seeks to simplify expressions as much as possible without instantiating any variables. It does this by searching within a term for subterms that match the redex of a rewrite rule and replacing it with the reduct of said rewrite rule.

In our program, evaluation is done with the rule eval/2. We have two definitions for this rule.

```
eval(E1, E2) :-
eval_one_step(E1, E3),
!,
eval(E3, E2).
```

and

```
eval(E1, E1).
```

The latter definition can be thought of as our base case. If eval fails to find any more simplifications, it will return itself. The former definition is a recursive method of successively applying single evaluation steps. The argument $E1$ is the input expression and $E2$ will be the final simplified expression. In sentence form, this rule says that $E1$ *evaluates to* $E2$ *if* $E1$ *evaluates to* $E3$ *in a single step, and* $E3$ *then evaluates to* $E2$. This recursive call to eval will cause $E3$ to evaluate to some $E4$ in a single step, and this process continues until it cannot be simplified any further, at which point this rule will not be able to be satisfied and the program will find the base case definition of eval. In the terminology of a reduction system, the first eval rule is known as the **transitive closure**, and the second eval rule is known as the reflexive closure. Our eval rule is meant to search for every simplification it can make under the term rewriting system, and therefore eval_one_step must be defined for each of these simplifications. For the idempotence rules, we have

```
eval_one_step(star(E, E), E).
eval_one_step(divi(E, E), E).
```

Similarly, right-cancellation gives us

```
eval_one_step(star(divi(E1, E2), E2), E1).
eval_one_step(divi(star(E1, E2) E2), E1).
```

We now need to have each delta rule as well:

```
eval_one_step(star(E1, star(E2, E3)),
        star( star(divi(E1, E3), E2), E3)).

eval_one_step(star(E1, divi(E2, E3)),
        divi( star( star(E1, E3), E2), E3)).

eval_one_step(divi(E1, star(E2, E3)),
        star(divi( divi(E1, E3), E2), E3)).

eval_one_step(divi(E1, divi(E2, E3)),
        divi(divi( star(E1, E3), E2), E3)).
```

Finally, we need to have rules that can simplify the subterms of a term. We do this by noting that a complex term is rooted at an operator, either star or divi. Then we just have to look at the subterms on either side of this operator. So this gives us

```
eval_one_step( star(E1,E2), star(E3, E2)) :-
eval_one_step(E1,E3).

eval_one_step( star(E1, E2), star(E1, E3)) :-
eval_one_step(E2, E3).

eval_one_step( divi(E1, E2), divi(E3, E2)) :-
eval_one_step(E1, E3).

eval_one_step( divi(E1, E2), divi(E1, E3)) :-
eval_one_step(E2, E3).
```

In sentence form, the first rule says "star(E1, E2)) simplifies to (star(E3, E2)) if we can simplify E1 to E3 in a single step". Notice that the second rule does this for the right subterm, and the next two do the same for the divi operation.

5.4 The Cut Operator and Unnecessary Backtracking

The ! is known as the cut operator in Prolog, and is used to prevent backtracking. What this means is that if eval_one_step cannot be satisfied, it will not try to backtrack to a previous single evaluation step and change it. Notice that our evaluation algorithm is reducing any quandle expression to its normal form within our term rewriting system. As McGrail showed that this term rewriting system is confluent, we should be able to find our normal form by applying term-rewrites in any order [1]. Therefore if we are unable to simplify further, we know that we have reached a normal form and backtracking is worthless.

Remark 5.4.1. If n is the number of operators in some expression E, then the complexity of eval without backtracking is $O(n)$ and the complexity of eval with backtracking is $O(n!)$. ◊

Proof. Notice that eval_one_step looks at each subterm. This means if n is the number of operators, in the worst case it will look at n complex subterms and $n+1$ non-complex subterms (constants or variables). In other words it is $O(n)$. Now suppose we have some expression where we can make rewrites A, B, C, and D in any order. Confluence tells us that any sequence should arrive at the same normal form. In other words, $A \to B \to C \to D = A \to B \to D \to C$ and so forth. If we allow backtracking, we have to check every permutation of these four rewrites. Thus our complexity becomes $O(n!)$. □

The cut operator should be used sparingly since it comes close to non-declarative programming. Here we use it because we can prove that confluence of our term rewriting system tells us the search space of a solution to our problem does not require backtracking.

5.5 The Quandle Unification Algorithm

The inspiration for this paper is to verify the result of a previous senior project. In 2021, Goldstein gave a unification algorithm for quandles [11]. In it, the algorithm takes a quandle unification problem $E_1 = E_2$ and converts it into a matching problem like so.

$$E_1 = E_2$$

$$a * E_1 = a * E_2$$

$$(a * E_1)/E_2 = a,$$

where a is some constant outside of the signature for quandle terms E_1 and E_2. The algorithm then attempts to unify $(a * E_1)/E_2$ and a. If it finds such a unifier, then that implies $E_1 = E_2$, and we have solved our unification problem. This implication, however, is not as clear as it seems. Suppose that in our unifier for $(a * E_1)/E_2$ and a, we instantiate some variables inside E_1 or E_2 to include a. Notice that this now does not immediately correspond to a unifier of $E_1 = E_2$, because that unification problem does not include a in its signature. Thus, in order for Goldstein's unification algorithm to be true, we must show that a solution to the converted matching problem does in fact lead to a solution for the unification problem in all cases. We now pick up where we left off at the end of Chapter 4 by discussing narrowing.

6 Narrowing and the Unification-Matching Reduction

Narrowing drives the bulk of the algorithm used by Goldstein in their quandle unification algorithm, and it will be a very useful tool for showing that unification reduces to matching for quandles. For more details on how narrowing was implemented in the Prolog algorithm, see [11].

6.1 Narrowing

Definition 6.1.1. Given a term t and a theory E, a single **narrowing step** involves the following [10].

1. Pick a compound subterm $t_1 \in t$ that contains at least one variable X.
2. Let $L = R$ be an identity in E. Search for a unifier for L and t_1 of the form $\sigma = [X \to Y]$ where Y is any term.
3. If such a unifier σ exists, then we apply the substitution $\sigma(t)$ and, within it, we replace the subterm $\sigma(t_1)$ with $\sigma(R)$.

A narrowing step is **substitution free** if we do not instantiate any of the variables in t_1. **Narrowing** is the process of continually composing narrowing steps on some expression $E_1 = E_2$ until a unifier has been found. $\triangle$

The process of narrowing on an expression $E_1 = E_2$ is a method for enumerating all possible E-unifiers for that expression. In other words, if a unifier exists for some expression then repeatedly searching the space of composition of narrowing steps will return that unifier [4]. The only problem that occurs, however, is if the narrowing process is not guaranteed to terminate at some point.

Definition 6.1.2. Let t be a term. A **narrowing sequence** on t is a sequence of narrowing steps

$$N = [s_0, s_1, s_2, \ldots, s_n]$$

where s_i is a single narrowing step involving instantiation of some variable in t, unification of some subterm t_1 with the reduct of a rewrite rule, and finally full reduction of t using rewrite system R. So,

$$N = s_n(s_{n_1}(\ldots(s_1(s_0(t))))).$$

Importantly, in a narrowing sequence we fully reduce t after each narrowing step. This is because a variable instantiated in some narrowing step can occur many times throughout the term t, and thus provides many new opportunities to simplify the term. △

Example 6.1.3. As an example of why we want to fully reduce between narrowing steps, consider

$$t = ((((a * t_2)/t_1) * t_0) * t_1)/t_2,$$

where t_1 is a variable. Suppose we do the following narrow step n.

$$[t_1 \to t_2] \tag{n}$$

$$((((a * t_2)/(t_2)) * t_0) * (t_2))/t_2$$

$$((a * t_0) * t_2)/t_2.$$

Notice, however, that the instantiation of t_1 has also given us another opportunity to right cancel. If we fully reduce this term, we get $a * t_0$. This is why a narrowing step of a narrowing sequence is always followed by reduction of the term, and in practice we will always be starting our narrowing sequence either with a reduced term, or by reducing it before any narrowing steps. Notice that Definition 6.1.1 tells us that a simple rewrite step is also a narrowing step–it only requires variable instantiation in the identity of the term rewriting system, and not the term itself. So, a narrowing sequence is a sequence of narrowing steps where we try to do as many simple rewrite narrowing steps—steps that do not require variable instantiation in t—in between each narrowing step s_i which requires instantiation of some variable in t. Therefore, if we had a narrowing sequence consisting of only n we would have the following.

$$[t_1 \rightarrow t_2] \tag{n}$$

$$((((a * t_2)/(t_2)) * t_0) * (t_2))/t_2$$

$$((a * t_0) * t_2)/t_2.$$

$$a * t_0.$$

$\Diamond$

Narrowing sequences for quandles have a unique property that we can exploit in service of showing that quandle unification reduces to quandle matching.

Theorem 6.1.4. *Let t be a quandle term. For every narrowing sequence N of t, there exists a narrowing sequence N' which does all of its delta rule narrowing steps first, right cancel rules second, and idempotence rules third such that*

$$N(t) = N'(t).$$

Proof. Let t be a term in reduced form. Let N be a narrowing sequence for t that contains subsequences $I = [i_1, i_2, \ldots, i_{n_1}]$, $R = [r_1, r_2, \ldots, r_{n_2}]$, and $D = [d_1, d_2, \ldots, d_{n_3}]$. Note We will be using i to denote an instance of an idempotence narrowing step, r for an instance of a right cancellation step, and d for a delta rule step. Furthermore, we have that

$$length(N) = length(I) + length(R) + length(D).$$

Note that N will be of the form $N = [s_1, s_2, \ldots, s_i, s_{i+1}, \ldots, s_n]$ Where $1 \leq i < n$ and s_i and s_{i+1} are two different types of narrowing steps. Then each case we must look at corresponds to each different pair of narrowing steps. Recall that we are using the $\circ$ symbol to denote an instance of some quandle operator, either $*$ or $/$.

Case 1: (Delta rules and right cancellation)

Suppose we look at right cancellation and delta steps as the first pair. Then either can come first, and t is constrained to two possible forms. Either

$$t = (\dots(E_1 \circ_1 A) \circ_1^{-1} Y) \circ_2 \dots) \circ_3 Y,$$

or

$$t = (\dots(E_1 \circ_1 Y) \circ_1^{-1} A) \circ_2 \dots) \circ_3 Y$$

where E_1 is some expression in normal form, Y is a variable, and A is a variable or a constant. Note that we only need to consider these two cases WLOG because these are the only forms of t where r and d share a variable involved in both steps–namely Y. In each case where right cancel and idempotence steps do not share a variable, it is trivial to show that you can apply either narrowing step first and arrive at the same result.

1.1 Let $t = (\dots(E_1 \circ_1 A) \circ_1^{-1} Y) \circ_2 \dots) \circ_3 Y$. We want to consider the case where we apply right cancellation first. Notice that, if we want to do right cancellation and then a delta rule, A must be a variable because otherwise we would apply the substitution $[Y \to A]$ for our initial right cancellation, which would make the outer Y in both cases a constant and we would no longer be able to narrow with a delta rule. Suppose that $N = [s_1, s_2, \dots, s_i, r, d, s_{i+1} \dots, s_n]$. Thus, our narrowing sequence will be

$$[A \to Y] \tag{r}$$

$$(\dots(E_1 \circ_1 (Y)) \circ_1^{-1} Y) \circ_2 \dots) \circ_3 Y$$

$$(\dots(E_1 \circ_2 \dots) \circ_3 Y$$

$$[Y \to (Y_1 \circ_4 Y_2)] \tag{d}$$

$$(\dots(E_1 \circ_2 \dots) \circ_3 (Y_1 \circ_4 Y_2)$$

$$(\dots(E_1 \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2.$$

Now we need to show that we arrive at the same normal form for t by applying some series of delta rules first and then possibly right cancellation rules. Recall that A still must be a variable, since we are working with the same term t. Then our sequence is

$$[Y \to (Y_1 \circ_4 Y_2)] \qquad (d)$$

$$(\dots(E_1 \circ_1 A) \circ_1^{-1} (Y_1 \circ_4 Y_2)) \circ_2 \dots) \circ_3 (Y_1 \circ_4 Y_2)$$

$$(\dots(E_1 \circ_1 A) \circ_1^{-1} (Y_1 \circ_4 Y_2)) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\dots(E_1 \circ_1 A) \circ_4^{-1} Y_2) \circ_1^{-1} Y_1) \circ_4 Y_2) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$[A \to (Y_1 \circ_4 Y_2)] \qquad (d')$$

$$(\dots(E_1 \circ_1 (Y_1 \circ_4 Y_2)) \circ_4^{-1} Y_2) \circ_1^{-1} Y_1) \circ_4 Y_2) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\dots(E_1 \circ_4^{-1} Y_2) \circ_1 Y_1) \circ_4 Y_2) \circ_4^{-1} Y_2) \circ_1^{-1} Y_1) \circ_4 Y_2) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\dots(E_1 \circ_4^{-1} Y_2) \circ_1 Y_1) \circ_1^{-1} Y_1) \circ_4 Y_2) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\dots(E_1 \circ_4^{-1} Y_2) \circ_4 Y_2) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\dots(E_1 \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2.$$

Note that the sequence of narrowing steps $[r, d]$ results in the same final substitution as the sequence $[d, d']$.Therefore we can construct

$$N' = [s_1, s_2, \dots, s_i, d, d', s_{i+1}, \dots, s^n],$$

where $N'(e) = N(e)$ for all terms e.

1.2 Now suppose $t = (\dots(E_1 \circ_1 Y) \circ_1^{-1} A) \circ_2 \dots) \circ_3 Y$. We again begin with right cancellation first. The reasoning is almost identical to that of the previous case. Let N be the same form as in section 1.1, and notice that A must still be a variable for the same reasons. So our narrowing sequence is as follows.

$$[A \to Y] \qquad (r)$$

$$(\dots(E_1 \circ_1 Y) \circ_1^{-1} (Y)) \circ_2 \dots) \circ_3 Y$$

$$(\ldots(E_1 \circ_2 \ldots) \circ_3 Y$$

$$[Y \to (Y_1 \circ_4 Y_2)] \tag{d}$$

$$(\ldots(E_1 \circ_2 \ldots) \circ_3 (Y_1 \circ_4 Y_2)$$

$$(\ldots(E_1 \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2.$$

Similar to the previous case, we now need to show that we can reach the same reduced form with delta rules first and possibly some right cancellations after. Then,

$$[Y \to (Y_1 \circ_4 Y_2)] \tag{d}$$

$$(\ldots(E_1 \circ_1 (Y_1 \circ_4 Y_2)) \circ_1^{-1} A) \circ_2 \ldots) \circ_3 (Y_1 \circ_4 Y_2)$$

$$(\ldots(E_1 \circ_1 (Y_1 \circ_4 Y_2)) \circ_1^{-1} A) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\ldots(E_1 \circ_4^{-1} Y_2) \circ_1 Y_1) \circ_4 Y_2) \circ_1^{-1} A) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$[A \to (Y_1 \circ_4 Y_2)] \tag{d'}$$

$$(\ldots(E_1 \circ_4^{-1} Y_2) \circ_1 Y_1) \circ_4 Y_2) \circ_1^{-1} (Y_1 \circ_4 Y_2)) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\ldots(E_1 \circ_4^{-1} Y_2) \circ_1 Y_1) \circ_4 Y_2) \circ_4^{-1} Y_2) \circ_1^{-1} Y_1) \circ_4 Y_2) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\ldots(E_1 \circ_4^{-1} Y_2) \circ_1 Y_1) \circ_1^{-1} Y_1) \circ_4 Y_2) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\ldots(E_1 \circ_4^{-1} Y_2) \circ_4 Y_2) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\ldots(E_1 \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2.$$

Therefore, in either case, we can find narrowing steps $[d, d']$ that results in the same final form of t as the sequence $[r, d]$, and so given our narrowing sequence with the form

$$N = [s_1, s_2, \ldots, s_i, r, d, s_{i+1} \ldots, s_n],$$

we have shown we can construct a new sequence

$$N' = [s_1, s_2, \ldots, s_i, d, d', s_{i+1}, \ldots, s^n]$$

where $N(e) = N'(e)$ for all terms e. Now we know that if we arrive at a contiguous subsequence S of a narrowing sequence consisting only of i delta rules and j right cancellation rules with $i, j \in \mathbb{N}$, there exists a contiguous subsequence S' with $i + j$ delta rules that results in the same end forms as S for any term.

Case 2: (Delta and idempotence) Now we look at delta and idempotence narrowing steps. Notice again that the only non-trivial cases are one where the same variable is involved in the idempotence and delta rule steps. Thus we have two cases for what a term can look like. Either

$$t = (\ldots (A \circ_1 Y) \circ_2 \ldots) \circ_3 Y$$

or

$$t = (\ldots (Y \circ_1 A) \circ_2 \ldots) \circ_3 Y.$$

2.1 Suppose $t = (\ldots (A \circ_1 Y) \circ_2 \ldots) \circ_3 Y$. We first consider the case of the idempotence step first. Notice that A must be a variable and not a constant, otherwise our idempotence step would involve sending Y to a constant. This would then make it impossible to carry out a delta rule step with the outer Y, now having been instantiated as a constant. Let $N = [s_1, s_2, \ldots, s_j, i, d, s_{j+1}, \ldots, s_n]$ be a sequence of narrowing steps. Then our contiguous subsequence $[i, d]$ applied to t will be

$$[A \to Y] \tag{i}$$

$$(\ldots ((Y) \circ_1 Y) \circ_2 \ldots)$$

$$(\ldots (Y \circ_2 \ldots) \circ_3 Y$$

$$[Y \to (Y_1 \circ_4 Y_2)]$$

$$(\ldots ((Y_1 \circ_4 Y_2) \circ_2 \ldots) \circ_3 (Y_1 \circ_4 Y_2)$$

$$(\ldots ((Y_1 \circ_4 Y_2) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2.$$

Now we want to reach the same form given application of delta rules first and potentially idempotence rules second. Notice that we can do the following sequence

$$[Y \to (Y_1 \circ_4 Y_2)] \tag{d}$$

$$(\ldots(A \circ_1 (Y_1 \circ_4 Y_2)) \circ_2 \ldots) \circ_3 (Y_1 \circ_4 Y_2))$$

$$(\ldots(A \circ_1 (Y_1 \circ_4 Y_2)) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\ldots(A \circ_4^{-1} Y_2) \circ_1 Y_1) \circ_4 Y_2) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$[A \to (Y_1 \circ_4 Y_2)] \tag{d'}$$

$$(\ldots((Y_1 \circ_4 Y_2) \circ_4^{-1} Y_2) \circ_1 Y_1) \circ_4 Y_2) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\ldots((Y_1 \circ_1 Y_1) \circ_4 Y_2) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\ldots(Y_1 \circ_4 Y_2) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2.$$

2.2 Now we consider $t = (\ldots(Y \circ_1 A) \circ_2 \ldots) \circ_3 Y$. Note that N is of the same form as in section 2.1, and A is still a variable for the same reasons. Doing idempotence first results in the sequence

$$[A \to Y] \tag{i}$$

$$(\ldots(Y \circ_1 (Y)) \circ_2 \ldots) \circ_3 Y$$

$$(\ldots(Y \circ_2 \ldots) \circ_3 Y$$

$$[Y \to (Y_1 \circ_4 Y_2)] \tag{d}$$

$$(\ldots((Y_1 \circ_4 Y_2) \circ_2 \ldots) \circ_3 (Y_1 \circ_4 Y_2)$$

$$(\ldots((Y_1 \circ_4 Y_2) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2.$$

Now, notice that we can reach the same end form with the sequence

$$[Y \to (Y_1 \circ_4 Y_2)] \tag{d}$$

$$(\ldots((Y_1 \circ_4 Y_2) \circ_1 A) \circ_2 \ldots) \circ_3 (Y_1 \circ_4 Y_2)$$

$$(\ldots((Y_1 \circ_4 Y_2) \circ_1 A) \circ_2 \ldots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$[A \to (Y_1 \circ_4 Y_2)] \qquad (d')$$

$$(\dots((Y_1 \circ_4 Y_2) \circ_1 (Y_1 \circ_4 Y_2)) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\dots((Y_1 \circ_4 Y_2) \circ_4^{-1} Y_2) \circ_1 Y_1) \circ_4 Y_2)) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\dots((Y_1 \circ_1 Y_1) \circ_4 Y_2)) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2$$

$$(\dots((Y_1 \circ_4 Y_2)) \circ_2 \dots) \circ_4^{-1} Y_2) \circ_3 Y_1) \circ_4 Y_2.$$

Therefore, in either case, we can find narrowing steps $[d, d']$ that results in the same final form of t as the sequence $[i, d]$, and so given our narrowing sequence with the form

$$N = [s_1, s_2, \dots, s_i, i, d, s_{i+1} \dots, s_n],$$

we have shown we can construct a new sequence

$$N' = [s_1, s_2, \dots, s_i, d, d', s_{i+1}, \dots, s^n]$$

where $N(e) = N'(e)$ for all terms e.

Case 3: (Right cancellation and idempotence) The final case we need to consider is the idempotence-right cancellation pair. Thus our narrowing sequence will look like

$$N = [s_1, s_2, \dots, s_j, i, r, s_{j+1}, \dots, s_n].$$

We must show that we can construct a contiguous sequence that is equivalent to the sequence $[i, r]$, but where all right cancellation steps are done before idempotence steps. Our term can be of the form

$$t = (\dots(Y \circ_1 A) \circ_2 \dots) \circ_3 B) \circ_3^{-1} Y,$$

where Y is a variable. Notice that at least one of B or A must be a variable, otherwise we would not be able to carry out an idempotence and then a right cancellation or vice versa. We don't have to consider the cases where the leftmost Y and A are flipped or where the rightmost Y and B are flipped because idempotence and right cancellation are commutative. If we can prove it works for this form then we know it works in all four different cases.

3.1 Let A be a variable, and so the narrowing sequence of $[i, r]$ is constrained to

$$[Y \to A] \tag{i}$$

$$(\ldots((A) \circ_1 A) \circ_2 \ldots) \circ_3 B) \circ_3^{-1} (A)$$

$$(\ldots(A \circ_2 \ldots) \circ_3 B) \circ_3^{-1} A$$

$$[A \to B] \tag{r}$$

$$(\ldots((B) \circ_2 \ldots) \circ_3 B) \circ_3^{-1} (B)$$

$$(\ldots(B \circ_2 \ldots).$$

We could have sent A to Y as our first idempotence step as well, but this would not change the end result since we could have simply sent Y to B in that case. Notice, however, that we can instead do the sequence

$$[Y \to B] \tag{r'}$$

$$(\ldots((B) \circ_1 A) \circ_2 \ldots) \circ_3 B) \circ_3^{-1} (B)$$

$$(\ldots((B) \circ_1 A) \circ_2 \ldots)$$

$$[A \to B] \tag{i'}$$

$$(\ldots(B \circ_1 (B)) \circ_2 \ldots)$$

$$(\ldots(B \circ_2 \ldots).$$

3.2 Suppose now that B is a variable. This case is nearly identical to the above so it does not need repeating of the algebra. For both our original $[i, r]$ and flipped $[r', i']$

sequences, we could keep our first substitutions, and the second substitutions would be flipped. For both of them, we now send B to A.

Therefore we now know that, given some sequence of narrowing steps $[i, r]$, we can find an equivalent sequence $[r', i']$.

Now that we have covered each individual pair, we finally can move onto our general case. Let $N = [s_1, s_2, \ldots, s_i, s_{i+1}, \ldots, s_n]$ be a sequence of narrowing steps as defined at the beginning of this proof. Recall that it contains subsequences $I = [i_1, i_2, \ldots, i_{n_1}]$, $R = [r_1, r_2, \ldots, r_{n_2}]$, and $D = [d_1, d_2, \ldots, d_{n_3}]$, and that

$$length(N) = length(I) + length(R) + length(D).$$

Let d be the first delta rule step that appears in N after a step of some other kind, either idempotence or right cancellation. So $N = [s_1, \ldots, s_{i-1}, s_i, d, s_{i+1} \ldots, s_n]$ where s_i is not a delta rule. In either case, we have shown that there exists d' such that the contiguous subsequence $[d, d']$ gives the same result as $[s_i, d]$. Thus we know that the narrowing sequence $S = [s_1, \ldots, s_{i-1}, d, d', s_i \ldots, s_n]$. will give the same end result as N when applied to any term. Now we repeat the process for S and look at $[s_{i-1}, d]$. If s_{i-1} is not a delta rule step, then we can do the same transformation to get a new narrowing sequence, moving d back another step. Eventually, we will encounter a delta rule to the left of d and the process will stop. Note that at this point everything to the left of d will be a delta rule, since it was the first to appear after either an idempotence or right cancellation step. Now we repeat the process, finding the first delta rule to appear after some non-delta rule and repeatedly swapping it back a step until we cannot. This will result in a narrowing sequence

$$N' = [d^1, d^2, \ldots, d^{n_3}, s'_1, s'_2, \ldots, s'_m],$$

equivalent to N, where each of $s'_1, s'_2, \ldots, s'_m$ is either a right cancellation or idempotence step and $D' = [d^1, d^2, \ldots, d^{n_1}]$ is a contiguous subsequence of delta rules.

We now want to generate a new narrowing sequence from N' that does all its right cancellation steps before any idempotence steps, but after all of its delta rule steps. Let $[i, r]$ be the first contiguous subsequence in N' where a right cancellation step appears after an idempotence step. We showed in case 3 above that there exists narrowing steps r', i' such that the sequence $[r', i']$ is equivalent to $[i, r]$ when applied to any term. Thus, we can generate a new narrowing sequence that is equivalent to N', except that the contiguous subsequence $[i, r]$ has been replaced with $[r', i']$. Just like with delta rules, we can repeat this process—swapping right cancellation steps with idempotence steps until it reaches either another right cancellation step or a delta rule step—until we create a narrowing sequence that does every right cancellation step before any idempotence step. Let this narrowing sequence be N'' Notice that our delta rule steps remain unaltered from N' to N''. Let $R' = [r^1, r^2, \ldots, r^{n_1}]$, and $I' = [i^1, i^2, \ldots, i^{n_3}]$ be subsequences in N''. Then, due to our construction, we know that

$$N'' = [d^1, d^2, \ldots, d^{n_1}, r^1, r^2, \ldots, r^{n_2}, i^1, i^2, \ldots, i^{n_3}].$$

Notice that—at each step from N to N''—we proved that every narrowing sequence produced the same final result as N for any term we applied it to. Thus, we know that

$$N(t) = N''(t)$$

for all terms t. □

6.2 The Unification-Matching Reduction

Now that we have the above result that any narrowing sequence can be refactored such that all of its delta rule narrowing steps are done first, right cancels are done second, and idempotence rules third, we are close to being able to reduce unification to matching. The last thing we need to do is present the general form for what an embedding of a unification problem into a matching problem will look like.

Lemma 6.2.1. *Consider the quandle term $T = a * (\ldots(t_0 \circ_1 t_1) \circ_2 t_2) \circ_3 \ldots) \circ_n t_n)$. where the subterm $(\ldots(t_0 \circ_1 t_1) \circ_2 t_2) \circ_3 \ldots) \circ_n t_n)$ is delta reduced. Then*

$$T \xrightarrow{*} (\ldots(a \circ_n^{-1} t_n) \circ_{n-1}^{-1} t_{n-1}) \ldots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2) \circ_3 \ldots) \circ_n t_n.$$

Note that this resulting term is also delta reduced, so the ellipsis on far left of the equation corresponds to repeated open parentheses.

Proof. Let $s_0 = t_0$ and $s_{n+1} = s_n \circ_{n+1} t_{n+1}$, for all $n \in \mathbb{N}$, where t_n is some quandle term. We show by induction that

$$a * s_n \overset{*}{\to} (\dots(a \circ_n^{-1} t_n) \circ_{n-1}^{-1} t_{n-1}) \dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2) \circ_3 \dots) \circ_n t_n$$

for all terms a.

As a base case, note that $a * s_0 = a * t_0$ which is already in reduced form.

Now for our inductive hypothesis, suppose that

$$a * s_n \overset{*}{\to} (\dots(a \circ_n^{-1} t_n) \circ_{n-1}^{-1} t_{n-1}) \dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2) \circ_3 \dots) \circ_n t_n$$

for some n, for all terms a. Then

$$\begin{aligned} a * s_{n+1} &= a * (s_n \circ n + 1 t_{n+1}) \\ &= ((a \circ_{n+1}^{-1} t_{n+1}) * s_n) \circ_{n+1} t_{n+1}. \end{aligned}$$

Now we let $x_a = (a \circ_{n+1}^{-1} t_{n+1})$. Notice that our inductive hypothesis tells us

$$x_a * s_n \overset{*}{\to} (\dots(x_a \circ_n^{-1} t_n) \circ_{n-1}^{-1} t_{n-1}) \dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2) \circ_3 \dots) \circ_n t_n,$$

and so

$$\begin{aligned} a * s_{n+1} &= (x_a * s_n) \circ_{n+1} t_{n+1} \\ &\overset{*}{\to} (\dots(x_a \circ_n^{-1} t_n) \circ_{n-1}^{-1} t_{n-1}) \dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2) \circ_3 \dots) \circ_n t_n) \circ_{n+1} t_{n+1} \\ &= (\dots(a \circ_{n+1}^{-1} t_{n+1}) \circ_n^{-1} t_n) \dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2) \circ_3 \dots) \circ_n t_n) \circ_{n+1} t_{n+1} \end{aligned}$$

□

With this general form of the embedded unification problem, we can show that unification reduces to matching.

Theorem 6.2.2. *The quandle unification problem reduces to the quandle matching problem.*

Proof. We want to show that, given any quandle unification problem, we can convert it to a quandle matching problem whose solution corresponds to a solution for the quandle unification problem. The other direction is trivial to show, since unification is harder than matching. Let Σ be some quandle signature containing the constant a and V be a variable set. Let $\Sigma' = \Sigma - \{a\}$. Consider the quandle unification problem with the signature and variable set (Σ', V). Recall from Theorem 4.2.3that the general form of a unification problem that is not reducible to a matching problem will be

$$(\ldots(t_0 \circ_1 t_1) \circ_2 t_2) \ldots) \circ_n t_n = B, \tag{6.2.1}$$

where $t_1, \ldots, t_n \in \Sigma' \cup V$ and $t_0, B \in V$. Notice that if a unification problem is simply in the form of a matching problem (or can be converted to one using quandle axioms), then obviously the solution to one corresponds to the other. Now, we transform this expression to a matching problem with signature Σ by adding $a * \ldots$ to both sides of the expression and moving B to the left side with right cancellation. So our matching problem will look like

$$(a * (\ldots(t_0 \circ_1 t_1) \circ_2 t_2) \ldots) \circ_n t_n)/B = a.$$

By Lemma 6.2.1, we know this problem can be recast to

$$((\ldots(a \circ_n^{-1} t_n) \circ_{n-1}^{-1} t_{n-1}) \ldots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2) \circ_3 \ldots) \circ_n t_n)/B = a. \tag{6.2.2}$$

Now that we have our original unification problem and the unification problem encoded into a matching problem, we first show that a right-cancellation or delta rule narrowing step on the matching side corresponds to an equivalent operation on the unification side.

Case 1: Right cancellation

Suppose we look at the narrowing step that does right cancellation between terms t_i and t_{i+1} in the expression.

$$(\ldots(a \circ_n^{-1} t_n) \ldots) \circ_{i+1}^{-1} t_{i+1}) \circ_i^{-1} t_i) \ldots) * t_0) \ldots) \circ_i t_i) \circ_{i+1} t_{i+1}) \ldots) \circ_n t_n)/B = a.$$

In order for a right cancel to be possible, we know that $\circ_i = \circ_{i+1}^{-1}$. Suppose we somehow unify t_i and t_{i+1} with narrowing step u in order to get a right cancel at $\ldots \circ_i t_i) \circ_{i+1} t_{i+1}) \ldots$.

Notice, however, that $\circ_i = \circ_{i+1}^{-1}$ tells us that $\circ_i^{-1} = (\circ_{i+1}^{-1})^{-1} = \circ_{i+1}$. Therefore we also can immediately right cancel at $\ldots) \circ_{i+1}^{-1} t_{i+1}) \circ_i^{-1} t_i) \ldots$ as well. So, overall, we have the sequence

$$[t_i = t_{i+1}] \tag{u}$$

$$(\ldots(a \circ_n^{-1} t_n)\ldots) \circ_{i+1}^{-1} t_{i+1}) \circ_i^{-1} (t_{i+1}))\ldots) * t_0)\ldots) \circ_i (t_{i+1})) \circ_{i+1} t_{i+1})\ldots) \circ_n t_n)/B$$

$$(\ldots(a \circ_n^{-1} t_n)\ldots) \circ_{i+1}^{-1} t_{i+1}) \circ_i^{-1} t_{i+1})\ldots) * t_0)\ldots) \circ_{i-1} t_{i-1}) \circ_{i+2} t_{i+2})\ldots) \circ_n t_n)/B$$

$$(\ldots(a \circ_n^{-1} t_n)\ldots) \circ_{i+2}^{-1} t_{i+2}) \circ_{i-1}^{-1} t_{i-1})\ldots) * t_0)\ldots) \circ_{i-1} t_{i-1}) \circ_{i+2} t_{i+2})\ldots) \circ_n t_n)/B.$$

Recall that our unification problem is of the form in Equation 6.2.1. Notice that we are guaranteed to be able to apply our narrowing step u on the unification side as well. We will get

$$[t_i = t_{i+1}] \tag{u}$$

$$(t_0 \circ_1 t_2) \circ_2 \ldots) \circ_i (t_{i+1}) \circ_{i+1} t_{i+1})\ldots) \circ_n t_n.$$

$$(t_0 \circ_1 t_2) \circ_2 \ldots) \circ_{i-1} t_{i-1}) \circ_{i+2} t_{i+2})\ldots) \circ_n t_n.$$

Therefore a right cancellation narrowing step u in the unification problem corresponds to two right cancellation narrowing steps in matching–the same step u and then another right cancel involving the same two terms. This means that both problems eliminate terms using right cancellation in the same way. If we suppose that t_{i+1} was the only instance of that specific term, then we have now successfully eliminated it from both problems. Importantly, since right cancellation is the only way to eliminate constants, we now know that if a constant is eliminated from the matching problem it will be eliminated from the unification problem and vice-versa. Notice that this happens because the form of the matching problem has a kind of symmetry. Starting from t_0 and until we reach a on the left and B on the right, each term one step further away from t_0 is the same, and its operator is the inverse of the other. If we can maintain this symmetry, then we know that right cancellation rules will continue to have a direct correspondence. In order to keep the property that constant elimination is mirrored across matching and unification, we must maintain some symmetrical window about t_0.

We still have one corner case to address. Suppose that $\circ_n = *$. This means we can instantiate B in order to get a right cancel rule with t_n on the matching side.

$$[B \to t_n]$$
$$((\dots(a \circ_n^{-1} t_n) \circ_{n-1}^{-1} t_{n-1}) \dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2) \dots) \circ_n t_n)/(t_n) = a$$
$$((\dots(a \circ_n^{-1} t_n) \circ_{n-1}^{-1} t_{n-1}) \dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2) \dots) \circ_{n-1} t_{n-1} = a.$$

Now, if we were to be able to right cancel $\circ_n^{-1} t_n$ with $\circ_{n-1}^{-1} t_{n_1}$, then it seems we have introduced an asymmetry. To address this, let us see how this operates in the unification problem.

Notice that if we instantiate B on the unification side to be t_n, we can eliminate t_n on the left side by right cancellation. So,

$$[B \to t_n]$$
$$(\dots(t_0 \circ_1 t_1) \circ_2 t_2) \dots) \circ_n t_n = (t_n)$$
$$(\dots(t_0 \circ_1 t_1) \circ_2 t_2) \dots) \circ_n t_n) \circ_n^{-1} t_n = t_n \circ_n^{-1} t_n$$
$$(\dots(t_0 \circ_1 t_1) \circ_2 t_2) \dots) \circ_{n-1} t_{n-1} = t_n.$$

Now suppose we right cancel $\circ_n^{-1} t_n$ with $\circ_{n-1}^{-1} t_{n_1}$ on the matching side, implying that t_n and t_{n-1} have been unified. Notice that this lets us do the same trick as with B. We can right cancel t_{n-1} on the left-hand side, and it collapses back to a single simple term on the right-hand side due to idempotence. Notice that, as right cancellation functions based on the symmetry we first outlined, we have shown that all cases of right cancellation result in a mirrored elimination of terms across unification and matching.

Case 2: Delta rule

This case turns out to be relatively simple. Again, we just have to check that a delta rule narrowing step maintains symmetry about some term t_0. Let us begin in the unification problem, and suppose we have a narrowing step d that does the following.

$$[t_i \to (F * G)] \tag{d}$$

$$(\ldots(t_0 \circ_1 t_1) \circ_2 t_2)\ldots) \circ_{i-1} t_{i-1}) \circ_i (F * G)\ldots) \circ_n t_n$$

$$(\ldots(t_0 \circ_1 t_1) \circ_2 t_2)\ldots) \circ_{i-1} t_{i-1})/G) \circ_i F) * G)\ldots) \circ_n t_n.$$

If we apply the same narrowing step d in our matching case, we get

$$[t_i \to (F * G)] \tag{d}$$

$$(\ldots(a \circ_n^{-1} t_n)\ldots) \circ_{i+1}^{-1} t_{i+1}) \circ_i^{-1} (F * G))\ldots) * t_0)\ldots) \circ_i (F * G)) \circ_{i+1} t_{i+1})\ldots) \circ_n t_n)/B$$

$$(\ldots(a \circ_n^{-1} t_n)\ldots) \circ_{i+1}^{-1} t_{i+1})/G) \circ_i^{-1} F) * G)\ldots) * t_0)\ldots) \circ_i (F * G)) \circ_{i+1} t_{i+1})\ldots) \circ_n t_n)/B$$

$$(\ldots(a \circ_n^{-1} t_n)\ldots) \circ_{i+1}^{-1} t_{i+1})/G) \circ_i^{-1} F) * G)\ldots) * t_0)\ldots)/G) \circ_i F) * G) \circ_{i+1} t_{i+1})\ldots) \circ_n t_n)/B.$$

Notice that this preserves the same symmetry we were looking for in the right cancellation case. In other words, this form still tells us that term elimination—most importantly constants—is mirrored across both problems.

Once again, we must cover our corner case with the asymmetry induced by right cancelling with B. On the matching side, our potentially problematic case will be

$$((\ldots(a \circ_n^{-1} t_n) \circ_{n-1}^{-1} t_{n-1})\ldots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2)\ldots) \circ_{n-1} t_{n-1} = a$$

$$[t_n \to (F * G)]$$

$$((\ldots(a \circ_n^{-1} (F * G)) \circ_{n-1}^{-1} t_{n-1})\ldots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2)\ldots) \circ_{n-1} t_{n-1} = a$$

$$((\ldots(a/G) \circ_n^{-1} F) * G) \circ_{n-1}^{-1} t_{n-1})\ldots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2)\ldots) \circ_{n-1} t_{n-1} = a.$$

Notice that the only case for a right cancel is between G and t_{n-1}. Note this implies that $* = \circ_{n-1}$, and so $/ = \circ_{n-1}^{-1}$. Let us see what this case looks like in the unification side.

$$(\dots(t_0 \circ_1 t_1) \circ_2 t_2)\dots) \circ_{n-1} t_{n-1} = t_n$$

$$[t_n \to (F * G)]$$

$$(\dots(t_0 \circ_1 t_1) \circ_2 t_2)\dots) \circ_{n-1} t_{n-1} = (F * G)$$

$$(\dots(t_0 \circ_1 t_1) \circ_2 t_2)\dots) \circ_{n-1} t_{n-1})/G = F$$

$$[G \to t_{n-1}]$$

$$(\dots(t_0 \circ_1 t_1) \circ_2 t_2)\dots) \circ_{n-1} t_{n-1})/(t_{n-1}) = F$$

$$(\dots(t_0 \circ_1 t_1) \circ_2 t_2)\dots) \circ_{n-2} t_{n-2} = F.$$

Doing this right cancellation on the matching side will give the form

$$((\dots(a/G) \circ_n^{-1} F) * G) \circ_{n-1}^{-1} t_{n-1})\dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2)\dots) \circ_{n-1} t_{n-1} = a$$

$$[G \to t_{n-1}]$$

$$((\dots(a/(t_{n-1})) \circ_n^{-1} F) * (t_{n-1})) \circ_{n-1}^{-1} t_{n-1})\dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2)\dots) \circ_{n-1} t_{n-1} = a$$

$$((\dots(a/t_{n-1}) \circ_n^{-1} F) \circ_{n-2}^{-1} t_{n-2})\dots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \circ_2 t_2)\dots) \circ_{n-1} t_{n-1} = a.$$

Notice that F and t_{n-2} cancelling in the matching problem was handled in the previous case–this corresponds to a right cancelling and idempotence rule on the unification side. If, however, F and t_{n-1} unify to right cancel on the matching side, then that corresponds to a simple subsitution $[F \to t_{n-1}]$ with no reduction on the unification side. Notice, then, that the asymmetric right cancellation of $(\dots) \circ_{n-2} t_2)) \circ_{n-1} t_{n-1}$ on the outside of the matching problem would again correspond to the right cancellation and idempotence trick on the unification side. Besides these values, our symmetry window stays the same, and so we know we continue to have a correspondence of term elimination.

Case 3: Idempotence

Idempotence is a bit more complicated than the first two cases. While we can do unification idempotence on the matching side in a way the preserves symmetry, notice that we can potentially have a narrowing step for matching that sends t_n to a, leading to an idempotence rule. Since a does not exist in the signature of the unification side, this leads to a problem. We will be unable to do some equivalent idempotence step on the unification side. We first deal the easy case and then outline our solution to the more difficult one. Because our procedure relies on Theorem 6.1.4 to allow us to consider narrowing sequences that do all idempotence narrowing steps last, notice that we only have to handle the case of simple idempotence narrowing steps, or idempotence steps that do not require variable instantiation to perform. We handle the idempotence narrowing steps at the very end of this proof.

3.1 (Unification to matching) In the unification problem, we only have a single place to do simple idempotence, and that is the subterm $(t_0 \circ_1 t_1)$. So our narrowing sequence will be as follows.

$$[t_0 = t_1]$$

$$(\ldots((t_1) \circ_1 t_1) \circ_2 t_2)\ldots) \circ_n t_n$$

$$(\ldots(t_1 \circ_2 t_2) \circ_3 t_3)\ldots) \circ_n t_n.$$

Now suppose $\circ_1^{-1} = /$ WLOG. If we apply the same narrowing sequence of i to the matching side, then we get

$$[t_0 = t_1]$$

$$(\ldots) \circ_2^{-1} t_2) \circ_1^{-1} t_1) * (t_1)) \circ_1 t_1) \circ_2 t_2)\ldots$$

$$(\ldots) \circ_2^{-1} t_2) \circ_1 t_1) \circ_2 t_2)\ldots.$$

Notice that if $\circ_1$ were equal to $/$ instead, we would get right cancellation on the right side. This form still preserves the symmetry of the matching side, and so our term elimination principle still holds. Notice that this case also completes the right cancellation case. Before, we had not addressed what would happen to the unification problem if we did a narrowing step on the matching side that involved right cancelling with t_0. Now we know that this leads to idempotence on the unification side.

3.2 (Matching to unification)

The only place we can do simple idempotence on the matching side is the subterm $(a \circ_n^{-1} t_n)$. Though it is tempting to assume that we know t_n will never be a, and so we will never have a simple idempotence case, recall from Theorem 6.1.4 that our solution for the delta rule and idempotence case was to actually embed simple idempotence *within* the second delta rule. Thus we cannot rule out the possibility that the embedded simple idempotence will have been with a. So, our simple idempotence narrowing step on the matching side will be as follows.

$$[t_n = a]$$
$$((\ldots(a \circ_n^{-1} (a)) \circ_{n-1}^{-1} t_{n-1}) \ldots) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \ldots) \circ_{n-1} t_{n-1}) \circ_n (a))/B$$
$$((\ldots(a \circ_{n-1}^{-1} t_{n-1}) \ldots) \circ_1^{-1} t_1) * t_0) \circ_1 t_1) \ldots) \circ_{n-1} t_{n-1}) \circ_n a)/B.$$

Notice that we now still have a symmetrical window up to t_{n-1} from either side of t_0 that corresponds to our unification problem.

If we end up in this case, then we know that t_n was either a term or a subterm for which a variable was instantiated, and that t_n became a. For the unification side, suppose that we kept that anonymous variable instantiation, but instead sent t_n to X, a fresh variable that does not appear anywhere else in the unification problem. Now the left-hand side of our unification problem will be of the form

$$(t_0 \circ_1 t_1) \circ_2 t_2) \ldots) \circ_n X$$

with the stipulation that, from now on, we cannot instantiate X until the very end of our procedure. If we did, we would not have a corresponding operation on the matching side. Notice that our corner case with the asymmetry caused by sending something to B is unaffected in this case. We still send whatever we would have sent to a to X. With this, we have managed to maintain our narrowing step correspondence between matching and unification. If t_{n-1} gets sent to a for right cancellation on the matching side, we send t_{n-1} to X for right cancellation on the unification side. Our delta rules function the same, and notice that our window of symmetry starting at t_0 going to t_{n-1} on either side has been preserved. Thus, for every operation, we know that term elimination is mirrored in both problems.

Now let N be a narrowing sequence that solves M, a matching problem of the form in Equation 6.2.2 which encodes a unification problem U of the form described by Equation 6.2.1. By Theorem 6.1.4, there exists narrowing sequence N' which solves M, but does all delta rule and right cancel narrow steps before idempotence. Let M' be the form of M after all delta and right cancel steps have been applied. If we can only now do idempotence steps to reach a unifier, we know that M' is of the form

$$(\ldots(a \circ_1 s_1) \circ_2 s_2) \ldots) \circ_n s_n) = a,$$

where s_i is either a variable or the constant a for all $1 \leq i \leq n$. Note that right cancellation is the only step that can eliminate constants, and so—after we have done all our right cancellation steps in N'—we know that M' cannot have constants other than a on the left-hand side if it unifies with a through only idempotence narrowing steps. Now let U' be the form of U after having also performed all delta and right cancel steps in N'. As proved in the cases above, if M' has eliminated all constants beside a at this point in the narrowing sequence, then the same is true for U'. Notice that, at each time we would have substituted in a for the matching side, we instead substituted in the variable X— a fresh variable that did not occur anywhere else in the unification problem—and stipulated that we could not instantiate it. In essence, treating it like a new constant. So, U' is of the following form

$$(t_0 \circ_1 t_1) \circ_2 t_2) \ldots) \circ_m t_m = E,$$

Where t_i is either the variable X or some other variable for $0 \leq i \leq m$ and E is some term. Recall that, in the matching and unification side, the variable B in Equation 6.2.2 and Equation 6.2.1 occurred in special cases of right cancellation for both problems, and so it is possible that E could be a constant. On the matching side, we immediately have a solution that corresponds to sending every variable in $\{s_1, \ldots, s_n\}$ to a and using a series of idempotence steps to collapse the entire left-hand side to a. For the unification side, we importantly are guaranteed that every term on the left-hand side will be a variable. Therefore, we send all variables—X included—to the term E and let the left-hand side collapse from idempotence rules as well.

Thus, for any quandle unification problem, we can embed it into a quandle matching problem, whose solution implies the existence of a solution to the embedded unification problem, and so quandle unification reduces to quandle matching. $\square$

7 Conclusion

7.1 Results

This paper shows that the quandle unification problem reduces to the quandle matching problem. We show that any unification problem can be embedded into a matching problem, and that the existence of a solution for that matching problem implies the existence of a solution for the embedded unification problem. This proves the claim of Elliott Goldstein, who previously created a quandle unification algorithm by solving an embedded matching problem. As an intermediate result, this paper also showed that a narrowing sequence for a quandle term can be refactored to partition its different types of narrowing steps. This result allowed for insights into what the final form of a solution looked like for both matching and unification cases, and will be very useful for future work.

7.2 Future Work

What remains to be shown is a bound on the number of delta rule narrowing steps in a unifier. Without this bound, the quandle unification algorithm cannot be proven to terminate at some point. More generally, the future work will be on determining the decidability of quandle unification. As we have shown quandle unification reduces to matching, it will be easiest to either show that unification is undedicable or that matching is decidable.

Bibliography

[1] AJ Tripathi, T.T. Nguyen, T.T. Tran Trang, and Robert McGrail, *A Terminating and Confluent Term Rewriting System for the Pure Equational Theory of Quandles* (2018).

[2] Franz Baader and Tobias Nipkow, *Term rewriting and all that*, 1st ed., Cambridge University, Cambridge, MA, 1998.

[3] Alberto Martelli and Ugo Montanari, *An efficient unification algorithm*, ACM Transactions on Programming Languages and Systems **4** (1982), 258–282.

[4] Jean Gallier and Wayne Snyder, *Complete sets of transformations for general E-unification*, Theoretical Computer Science **67** (1989), 203–260.

[5] Leon Sterling and Ehud Shapiro, *The Art of Prolog*, MIT press, 1986.

[6] David Joyce, *A Classifying Invariant of Knots, the Knot Quandle*, Pure and Applied Algebra **23** (1982), 37–65.

[7] Sam Nelson and Mohammed Elhamdad, *Quandles: An Introduction to the Algebra of Knots*, Vol. 74, American Mathematical Society, 2015.

[8] M. Armstrong, *Basic Topology*, Springer, New York, NY, 2011.

[9] H. Reidemeister, *Knoten un Gruppen*, Abh. Math. Sem. (1926).

[10] Daniel Dougherty and Patricia Johann, *An Improved General E-Unification Method*, Journal of Symbolic Computation **14** (1992), 303–320.

[11] Elliot Goldstein, *A Unification Algorithm For The First Order Theory of Quandles*, Senior Projects Spring 2021 **152** (2021).