Finite Algebras on the Semantic Web

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

by
Feifan Zheng

Annandale-on-Hudson, New York
May, 2013

Abstract

This project explores information sharing in mathematical research. Currently, the major approaches to dissemination are journal articles, reviews, and graduate-level textbooks. Other less formal approaches include conference talks, interpersonal conversations, mathematical software, and static websites.

Mathematical knowledge management has as its aim to improve the sharing of mathematical research artifacts through the development of new data formats and other digital means. In this work, we consider the problem of publishing the products of research in universal algebra, specifically finite algebras and their properties, on the Semantic Web.

Contents

Abstract **1**

Dedication **6**

Acknowledgments **7**

1 Introduction **8**
1.1 Current Status of Information Sharing 8
1.1.1 Journals, Reviews, and Textbooks[13] 8
1.1.2 Conference Talks and Other Interpersonal Communication 9
1.1.3 Software 9
1.1.4 Website Repositories 10
1.2 New Ways of Sharing Mathematics Information 10
1.3 The Solution – the Semantic Web 11
1.3.1 The Traditional World Wide Web 12
1.3.2 The Modern World Wide Web 13
1.3.3 The Semantic Web 15

2 Triples and the Semantic Web **16**
2.1 Triples 16
2.2 Representing Triples 22
2.2.1 Resource Description Framework (RDF) 22
2.2.2 Terse RDF Triple Language (Turtle) 23

3 Quandles **27**
3.1 Algebra 27
3.1.1 General Definitions and Axioms 27

3.1.2 Example of Algebras 28
3.2 Quandles 29
3.2.1 Definition 29
3.2.2 Examples 29
3.2.3 Results from the ASC Lab 30

4 Quandles and Triples Transformation **32**
4.1 Breaking Down A Quandle 32
4.1.1 From operation table to "list of lists" Mathematica format 32
4.1.2 From quandle to "list of lists" Mathematica format 37
4.2 Transformation Implementation 42
4.2.1 From Mathematica list of triple lists to RDF 42

5 Conclusion **46**
5.1 Future Work 47

Bibliography **49**

List of Figures

1.3.1 12
1.3.2 13
1.3.3 13
1.3.4 14
1.3.5 14
1.3.6 15

2.1.1 16
2.1.2 17
2.1.3 17
2.1.4 17
2.1.5 18
2.1.6 18
2.1.7 19
2.1.8 19
2.1.9 19
2.1.10 20
2.1.11 21
2.2.1 22
2.2.2 23
2.2.3 23
2.2.4 24
2.2.5 24
2.2.6 24
2.2.7 24
2.2.8 24
2.2.9 25

2.2.10 25

4.1.1 34
4.1.2 34
4.1.3 35
4.1.4 35
4.1.5 35
4.1.6 36
4.1.7 37
4.1.8 37
4.1.9 38
4.1.10 39
4.1.11 40
4.1.12 41
4.2.1 42
4.2.2 42
4.2.3 44
4.2.4 44
4.2.6 44
4.2.5 45
4.2.7 45

5.0.1 46
5.0.2 47
5.0.3 47
5.0.4 48

Dedication

To my beloved parents.
献给我最挚爱的爸爸妈妈。

谁言寸草心，报得三春晖。

Acknowledgments

I would like to express my deep gratitude to Professor McGrail, my senior project advisor and a great friend, for his continuous support, patient guidance, and enthusiastic encouragement.

I would also like to thank Professor Thomas and Professor Hsiao, my senior project board members, for their useful critiques of this project.

Special thanks to my dear friends from Bard: Hsiao-Fang Lin, Xingye Zhang, Steven Wu, Yu Qiu, Hanze Song, Lei Lu, Chi-Hui Yen, Winnie Yau, Xiaohao Wu;
And my friends from back home: 门门，狐狸，没没.

Finally, I wish to thank my parents for their undivided support and encouragement throughout my study at Bard.

1
Introduction

1.1 Current Status of Information Sharing

Mathematical information for research purpose can be obtained through various sharing approaches. However, there are plenty of research artifacts that are useful to mathematicians but are not normally shared and verified. In another words, the sum total of the sharing methods leaves out much important mathematical information.

1.1.1 Journals, Reviews, and Textbooks[13]

The primary formal approaches to acquire mathematical research results are through published journal articles, reviews, and graduate-level textbooks, which traditionally appear as hard copies. In modern days, these can also be downloaded from the internet. In general, the content of a journal article only supports the main results of that work. Extensive details and examples are frequently omitted from the work. Additionally, people must follow the narrative in order to obtain the information from it. What are often missing, or, at least, difficult to extract are the countless examples in the course of mathematical discovery.

1.1.2 Conference Talks and Other Interpersonal Communication

Conference talks and interpersonal communication are also common approaches of sharing mathematical information. These methods involves real-time interactions. The receiver of information can pose further queries to the provider of information in real time. However, the information shared through these approaches might lack accuracy. The facts presented have not been peer-reviewed. Moreover, mistakes in communication might be more likely in real time.

1.1.3 Software

The uses of computers in the mathematical research community are many. In the early days of computing, mathematicians harnessed these machines for complex calculations. Nowadays, computers are employed for mathematical typesetting, visualization, communication and the archive of digital journals. Moreover, there are a myriad of systems to support mathematical research. We mention some of these below:

- "Canned" software

  Many mathematicians make use of "canned" software in their research process. One such example of a "canned" software is UACalc[14], the universal algebra calculator. UACalc allows users to input a finite algebra and determine whether that algebra has certain properties. Another example is KnotPlot[15], which is a software that visualizes and manipulates mathematical knots and braids in three and four dimensions.

- Customizable, limited-scope software

  Other mathematics researchers use many systems that are also for special circumstances that allow a limited degree of customizability. These systems allow users to define their own tests, but the scope of those tests are limited. For example,

the Mace4/Prover9[16] suite of first-order utilities allows the development of custom searches and verifications. However, those processes must be specified in first-order logic, which excludes most computable queries and tests.

- Full symbolic system

  Mathematicians also use full symbolic systems such as Mathematica[3], Maple[18], Matlab[19], and Wolfram Alpha[20] to design mathematical software. These are general programing system that are optimized for use in a wide variety of mathematical applications.

1.1.4 Website Repositories

Website repositories also contains a massive amount of mathematics information that Mathematicians often require. Information shared through this method often covers complex mathematics concepts and topics, and is maintained by experts of mathematics. These website repositories can be maintained either privately or publicly. Website repositories such as MathWorld[21] and Knot Atlas[22] are maintained by private organizations, while for websites such as Wikipedia[23], information editing is open to the public. Both privately and publicly maintained website repositories have their own flaws. For example, information from publicly maintained websites is often subject to higher variation in reliability, while mistakes that are found in private website repositories are hard for and users to correct.

1.2 New Ways of Sharing Mathematics Information

- Mathematical Knowledge Management (MKM)

  Mathematical Knowledge Management[10], also known as MKM, is an innovative field that focuses on developing new techniques to help in presenting, producing, transmitting, consuming, and managing sophisticated mathematical knowledge.

MKM involves subjects such as mathematics, computer science, library science, and scientific publishing. Areas of research in MKM include, but are not limited to:

  - Repositories of formalized mathematics;
  - Web presentation of mathematics; and
  - Collaboration tools for mathematics.

- Digital Mathematics Library (DML)

  Digital Mathematics Library[11], also known as DML, is a project that pursues a global mathematical digital library. Topics of inquiry in DML include, but are not limited to:

  - Search, indexing, and retrieval of mathematical documents;
  - Mathematical document compression; and
  - Long term archiving, data migration.

- OpenMath

  The OpenMath project[12] has the aim of formulating a standard for representing, exchanging, storing, and publishing mathematical objects along with their semantics on the World Wide Web. Topics pertinent to OpenMath include:

  - Displaying mathematical objects in a browser;
  - Exchanging mathematical objects between software systems; and
  - "Cutting and pasting" mathematical objects for use in different applications.

1.3 The Solution – the Semantic Web

The Semantic Web[7] is a project to create one giant, distributed, and linked database on the World Wide Web. This will allow people to share information beyond the boundaries

of applications and websites. In the Semantic Web, information is linked on a global scale and can be easily processed by machines. In another words, the Semantic Web functions as a globally-linked database that allows computers to seek out knowledge distributed throughout the Web, compile it, and then take actions based on it.

The Semantic Web is a somewhat utopia vision of the World Wide Web. The ultimate difference between the traditional web and the Semantic Web is described by Tim Berners-Lee, the the inventor of the World Wide Web, as below:

> "If HTML and the Web made all the online documents look like one huge book, RDF, schema, and inference languages will make all the data in the world look like one huge database"[2].

1.3.1 The Traditional World Wide Web

The original interaction model for the traditional World Wide Web, also referred as the Web, proceeds as follows:

1. The web browser makes a request specified by a uniform resource locator (URL) to the web server. The URL specifies a server, a path on that server, and the document of interest. See Figure 1.3.1.

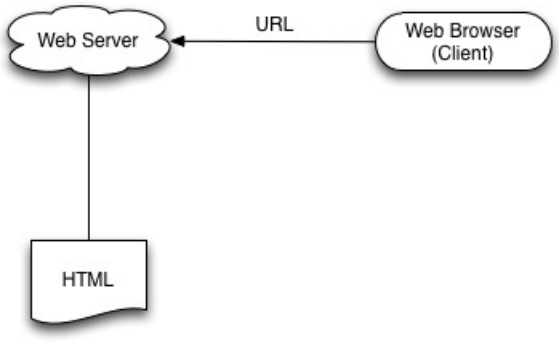

Figure 1.3.1.

2. The web server retrieves the requested document in HyperText Markup Language (HTML) format. See Figure 1.3.2.

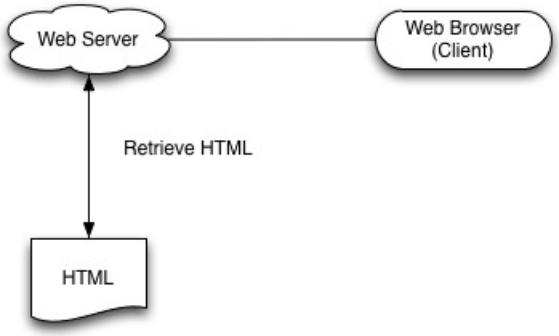

Figure 1.3.2.

3. The web server serves the HTML document to the web browser. And the web browser renders the HTML to the users. See Figure 1.3.3.

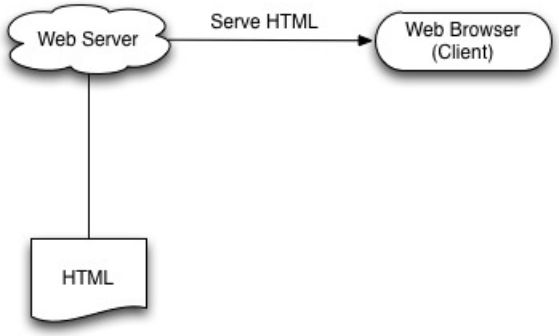

Figure 1.3.3.

1.3.2 The Modern World Wide Web

The modern World Wide Web can be viewed through the model-view-controller (MVC) model. The steps are as follows:

1. As before, the web browser requests information from the web server via a URL. This time the requested document may represent an online application rather than just a static HTML file. Fee Figure 1.3.4.

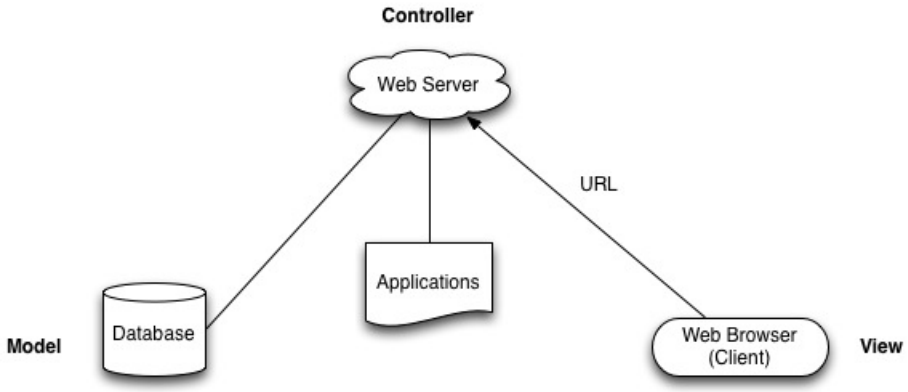

Figure 1.3.4.

2. The web server loads the online application, which then determines the response. The application may request information from a back-end database. See Figure 1.3.5.

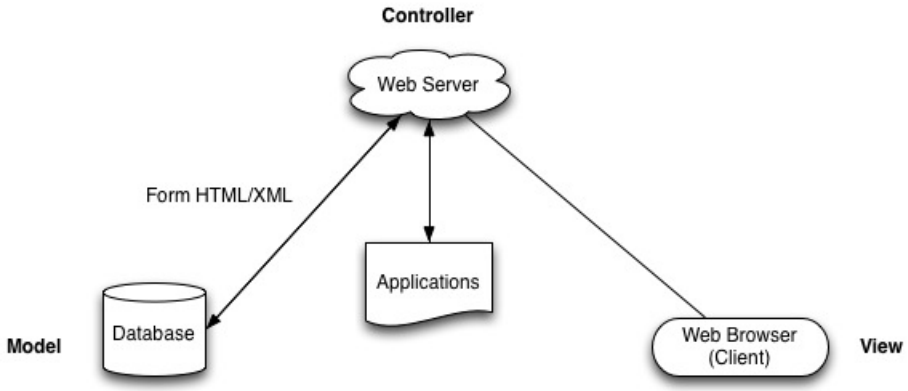

Figure 1.3.5.

3. The web server serves the response to the web browser. However, the response may come in XML format rather just HTML format. See Figure 1.3.6.

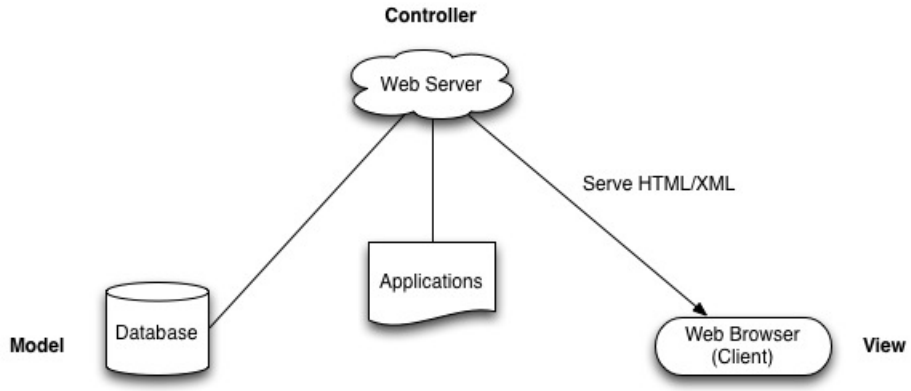

Figure 1.3.6.

1.3.3 The Semantic Web

The modern architecture described above can support the Semantic Web. The enhancement of the Semantic Web case is the incorporation of Resource Description Framework (RDF) files in the process. To be more specific, the response that the application produces is in RDF format, which is a form of XML. Details about how information is stored in the Semantic Web and how RDF file is formulated will be discussed in Chapter 2.

Sharing mathematics information on the Semantic Web can overcome the drawbacks of the previously mentioned sharing methods such as the lack of details, accuracy, and versatility. This project will focus on representing and publishing finite algebras, which is a narrow field in mathematics, on the Semantic Web.

2
Triples and the Semantic Web

The Semantic Web, which serves as a massive, globally-linked, distributed database, allows users to store and query information from the World Wide Web. In order to generalize the Semantic Web, those who enter information into this database will have to adhere to a common standard syntax.

2.1 Triples

Information in the Semantic Web is decomposed and stored into small pieces, called triples[1]. A triple consists of a subject, a predicate and an object. The subject denotes the resource, and the object denotes the target. The predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. For example, given a simple statement:

"I own my apartment."

Figure 2.1.1.

We can decompose it as:

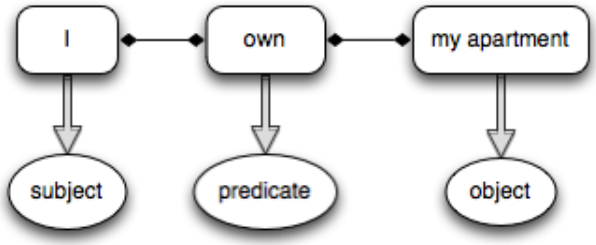

Figure 2.1.2.

If we complicate the statement in Figure 2.1.1 and turn it to:

"I own my apartment, which has a balcony."

Figure 2.1.3.

We can enhance the relation in Figure 2.1.2 to:

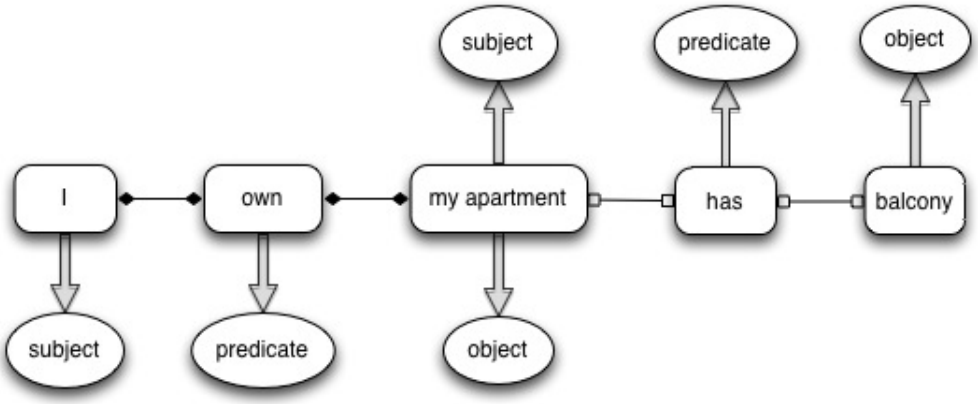

Figure 2.1.4.

As we can see, there are two triples in Figure 2.1.4. The first one is what we have in Figure 2.1.4. It has "I" as the subject, "my apartment" as the object and "own" as the predicate to indicate the relation between "I" and "my apartment". The second triple has "my apartment" as the subject, "has" as the predicate and "balcony" as the object. Notice that these two triples share a piece of information, which is "my apartment". The

"my apartment" "I" "own" is the same "my apartment" that "has" "balcony". So "my apartment" is both the object of the first triple and the subject of the second triple. Thus, in Figure 2.1.4, the "my apartment" unit points to both subject and object tags. If we have a table storing the information such as the table in Figure 2.1.5, we are able to retrieve the complete information of the statement in Figure 2.1.3.

Subject	**Predicate**	**Object**
I	own	my apartment
my apartment	has	balcony

Figure 2.1.5.

If we keep complicating the statement in Figure 2.1.3, we can update the table in Figure 2.1.5 by appending more triples that capture the new information. In fact, if we think in an opposite direction, any complex statements can be decomposed to a collection of simple statements. The examples we mentioned above are naturally subject-verb-object structured, so it is obvious to see how we can reduce it. For statements that are not naturally subject-verb-object structured, the approach is not obvious. For example, let us look at the following statement:

"Bob gives a computer to Lily."

Figure 2.1.6.

In the above statement, we have "Bob" as the subject, "gives" as the verb, "a computer" as the object and "to Lily" as the indirect object. Since indirect object is involved here, it is not likely to translate it directly to triples without losing information. The solution is to give the statement an identifier and take it as the subject in triples. Then for each segment of the statement, we take the type of the segment as the predicate and the content of the segment as the object. So let us name the statement in Figure 2.1.6 as SEN1. Then we can store the information of SEN1 as below:

Subject	Predicate	Object
SEN1	subject	Bob
SEN1	verb	gives
SEN1	object	a computer
SEN1	indirect object	to Lily

Figure 2.1.7.

In this way, we manage to decompose a non-subject-verb-object-structured statement to a list of triples. Thus, we can transform any single arbitrary statement to a list of triples while the information of that statement is preserved within the triples. If we have a collection of statements, we can still get the list of triples of the complete information by joining each list of triples from each statement. Therefore, the claim is that any database can be reduced to a collection of triples.

So let us keep complicating the statement in Figure 2.1.3:

"I own my apartment, which has a balcony. My friend Susie also owns her apartment, which also has a balcony."

Figure 2.1.8.

The graphical presentation of the above statement in Figure 2.1.9. Here we represent the

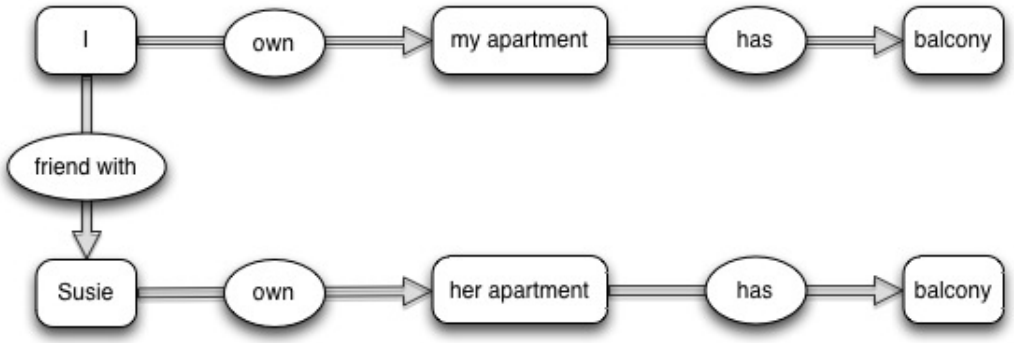

Figure 2.1.9.

relation differently from Figure 2.1.2 and Figure 2.1.4. In Figure 2.1.9, subject and object

units are denoted by rectangles and predicate units by ellipses. An arrow indicates a triple relation. The beginning of the arrow connects to the subject unit of the triple relation and the end of the arrow points to the object unit. The predicate unit of the triple relation is attached to the middle of the arrow. In Figure 2.1.9, we have five triples.

Then we run into a problem. In Figure 2.1.9, there are two object units that are both named "balcony" but refer to two different object. If we encounter a "balcony" object when we are retrieving the information form the triples, we would not be able to tell which "balcony" it really points to. Similarly, having two units named "my apartment" and "her apartment" would be vague and problematic. The solution is to number the same type of things in order, so that we don't get confused when we retrieve information back. The updated graphical representation is as below:

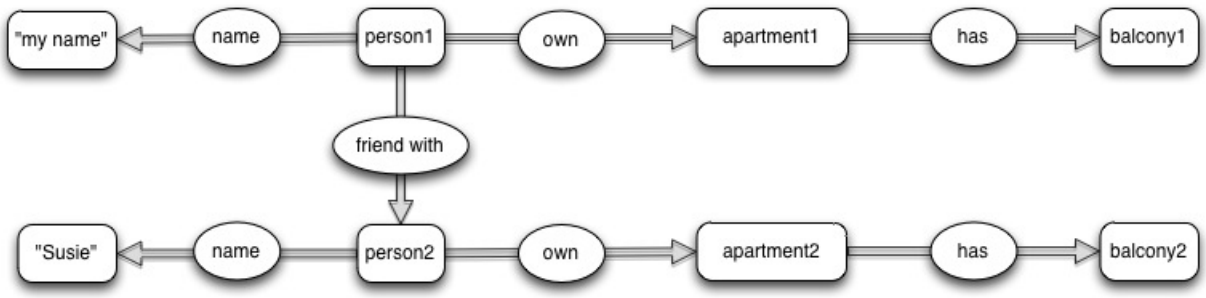

Figure 2.1.10.

Figure 2.1.10 will fix the problem we mentioned above. However, Figure 2.1.10 is still problematic. I name my apartment as "apartment1" and Susie's apartment as "apartment2" so that in my local database, these two apartments can be distinguished. However, someone else might call his apartment "apartment1" in his local database. There would be no confusion if both of our databases stay disconnected. But if both of us submit our databases to the Semantic Web, then we would run into the problem of ambiguity again. To fix this problem completely, we replace all the names we define to Uniform Resource Identifiers.

Uniform Resource Identifier, in short URI, is the generic term for all types of names and addresses that refer to resources on the World Wide Web. It is consisted of a string of characters. Below are a few examples of URIs:

- http://example.org/stuff/1.0/homePage
- ftp://example.org/resource.txt
- http://www.asclab.org/zheng

Specifically, an URI starts with a protocol "http://", followed by the server name, a unix path name, and an optional string representing a unix filename or application.

Thus, we can modify the triples in Figure 2.1.10 to:

Subject	**Predicate**	**Object**
http://www.asclab.org/zheng/example/-person/1	name	"my name"
http://www.asclab.org/zheng/example/-person/1	own	http://www.asclab.org/zheng/example/-apartment/1
http://www.asclab.org/zheng/example/-person/1	friend with	http://www.asclab.org/zheng/example/-person/2
http://www.asclab.org/zheng/example/-apartment/1	has	http://www.asclab.org/zheng/example/-balcony/1
http://www.asclab.org/zheng/example/-person/2	name	"Susie"
http://www.asclab.org/zheng/example/-person/2	own	http://www.asclab.org/zheng/example/-apartment/2
http://www.asclab.org/zheng/example/-apartment/2	has	http://www.asclab.org/zheng/example/-balcony/2

Figure 2.1.11.

With URIs, we would no longer have the problem of ambiguity because every item is represented by a unique URI.

2.2 Representing Triples

In the following section, we will introduce two different representations of triples and discuss the features of each representation.

2.2.1 Resource Description Framework (RDF)

Resource Description Framework (RDF)[9] is the standard framework for encoding metadata and describing other resources on the Semantic Web. RDF is an abstract model, a way to break down knowledge into discrete pieces. While being designed to be read and understood by computers, it is not meant to be displayed to people. RDF identifies things using URIs, which we have discussed the advantage of doing so. The common format of storing RDF is XML. It describes resources(subjects) with properties(predicates) and property values(objects) in the following basic structure:

<?xml version="1"? >

<RDF>
  <Description about="*subject*" >
    < *predicate*1 > *object*1 < /*predicate*1 >
    < *predicate*2 > *object*2 < /*predicate*2 >
  < /Description>
< /RDF>

Figure 2.2.1.

Notice that all the triples with the same subject are included under the same parent "Description" tag. Each triple forms a child tag with the predicate as the tag name and the object as the content of the tag. If we have multiple subjects, we append more "Description" tags under the root "RDF" tag.

The triples in Figure 2.1.11 would look like the following in RDF format:

```
<?xml version="1"? >

<RDF>
    <Description about="http://www.asclab.org/zheng/example/person/1">
        <name>"my name"< /name>
        <own>http://www.asclab.org/zheng/example/apartment/1< /own>
        <friend_with>http://www.asclab.org/zheng/example/person/2< /friend_with>
    < /Description>
    <Description about="http://www.asclab.org/zheng/example/apartment/1">
        <has>http://www.asclab.org/zheng/example/balcony/1< /has>
    < /Description>
    <Description about="http://www.asclab.org/zheng/example/person/2">
        <name>"Susie"< /name>
        <own>http://www.asclab.org/zheng/example/apartment/2< /own>
    < /Description>
    <Description about="http://www.asclab.org/zheng/example/apartment/2">
        <has>http://www.asclab.org/zheng/example/balcony/2< /has>
    < /Description>
< /RDF>
```

Figure 2.2.2.

RDF format can store every detail information of triples. So when RDF files are passed to computers, those computers obtain the full-scale database that they can retrieve information from. However, the RDF-XML rules of structuring and tedious syntax make it very hard for people to read and understand the relations between resources.

2.2.2 Terse RDF Triple Language (Turtle)

Terse RDF Triple Language (Turtle)[8] is a plain-text RDF representation, which is representing information using "triples". Each triple has the basic format:

$$< subject >< predicate >< object > .$$

Figure 2.2.3.

Each statement ends with a period. The subject element and the predicate element in the triple are URIs. The object element, however, can be represented in a form of string, number or URI. "," is introduced for statements with the same subject and predicate.

we can make the document shorter and more readable by combining the statements and separating the objects by one or more commas.

$$< subject >< predicate >< object1 >, < object2 >, < object3 > .$$

Figure 2.2.4.

";" is introduced for statements with the same subject but different predicates and objects. We combine them and separate the predicate-object parts with a semicolon.

$$\begin{aligned} &< subject > \\ &\quad < predicate1 >< object1 >; \\ &\quad < predicate2 >< object2 > . \end{aligned}$$

Figure 2.2.5.

Besides the basic format, prefixes are allowed by adding following string to the top of the Turtle file:

$$\text{@prefix } pref\text{: } < URI\ prefix > .$$

Figure 2.2.6.

Turtle also allows us to define the type relationship, which is a relationship between a thing and a category of things, by a special keyword **a**.

$$< subject > \text{ a } type\_name.$$

Figure 2.2.7.

When we don't want to define an URL for something, we can use an anonymous object in the object area in Turtle. Anonymous objects are defined with square brackets "[ ]" around one or more predicate-object pairs, defining the object, separated by semicolons. We can define all kinds of relationships that that object is the subject of.

$$< subject1 >< predicate1 > [< predicate2 >< object2 >; < predicate3 >< object3 >].$$

Figure 2.2.8.

The triples in Figure 2.1.11 should be presented as following in the Turtle format:

```
@prefix : <http://www.asclab.org/zheng/example/>.
@prefix person: <http://www.asclab.org/zheng/example/person/>.
@prefix apartment: <http://www.asclab.org/zheng/example/apartment/>.
@prefix balcony: <http://www.asclab.org/zheng/example/balcony/>.

person:1
      :name “my name”;
      :own apartment:1;
      :friend_with person:2.

apartment:1 :has balcony:1.

person:2
      :name “Susie”;
      :own apartment:2.

apartment:2 :has balcony:2.
```

Figure 2.2.9.

Figure 2.2.9 can be further simplified as below:

```
@prefix : <http://www.asclab.org/zheng/example/>.
@prefix person: <http://www.asclab.org/zheng/example/person/>.
@prefix balcony: <http://www.asclab.org/zheng/example/balcony/>.

person:1
      :name “my name”;
      :own [:has balcony:1];
      :friend_with [:name “Susie”; :own [:has balcony:2]].
```

Figure 2.2.10.

As we can see, Turtle format is good at presenting resource information to readers. The syntax is more concise and readable. For example, the representation in Figure 2.2.9 is much more straightforward and pleasant to look at than the representation in Figure 2.2.2. It also allows us to reconstruct its structure so the relationships among different resources are more obvious and focused to readers. For example, Figure 2.2.10 gives readers a better

understanding of how each resource is related to “person:1”.

As for the downside of Turtle format, the rules of structuring a Turtle file is more complex. Also, some information might be excluded while we are simplifying the file. For example, we lost the information of “apartment:1”, “apartment:2” and “person:2” in Figure 2.2.10. We only know “person:1” “own” somethings that “has” “balcony:1” for instance.

3
Quandles

This chapter defines algebras and quandles in mathematics. It also introduces some of the research projects concerning quandles, which shows the need of sharing quandles and their properties on the Semantic Web.

3.1 Algebra

3.1.1 General Definitions and Axioms

Definition. An **operation** f on a set s is a function $f : S \times S \times \cdots \times S \to S$.

$*$ is said to be a **binary operation** if $*$ is a function:

$$* : S \times S \to A.$$

And A is said to be **closed** under the binary operation $*$. Usually we denote $*(a, b)$ by $a * b$ where $a, b \in S$. We say $*$ is **associative** if

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

for all $a, b, c \in S$. And we say $*$ is **commutative** if

$$a * b = b * a$$

for all $a, b, c \in S$. If there exists an element $e \in S$ such that

$$e * a = a = a * e$$

for all $a \in S$, then e is called the **identity**. If for each $a \in S$, there exists $a^{-1} \in S$ such that

$$a * a^{-1} = e = a^{-1} * a,$$

then a^{-1} is called the **inverse** of a.

Definition. An finite **algebra** A is a pair (S, F), where S is a finite set and F is a set of finitary operations on S.

3.1.2 Example of Algebras

One of the common types of algebra is group.

Definition. A **group** $G = (S, \cdot, ()^{-1}, e)$ is a set S with a binary operation $\cdot$ on A if the following axioms holds:

- (Associativity:) For all $a, b, c \in S$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- (Identity:) There exists an element $e \in S$ with $e \cdot a = a = a \cdot e$ for all $a \in S$.
- (Inverse:) For each $a \in S$, there exists $a^{-1} \in S$ such that $a^{-1} \cdot a = e = a \cdot a^{-1}$.

One example of a group is $G = (I, +, (-)^{-1}, 0)$, where

- I is the set of integers.
- $+$ is the operation of addition.
- 0 is the identity.
- the inverse operation is negation.

3.2 Quandles

3.2.1 Definition

Definition. A **quandle** $Q = (S, *, /)$ is a finite algebra, containing two binary operations that satisfies idempotence, right cancellation and right self-distributivity on S. In another word, for any arbitrary $a, b, c \in A$,

- (Idempotence:) $a * a = a$.
- (Right Cancellation:) $(a * b)/b = a$ and $(a/b) * b = a$
- (Right Self-Distributivity:) $(a * b) * c = (a * c) * (b * c)$.

3.2.2 Examples

A simple example of a quandle is the Unary Quandle on two elements $U_2 = (\{0, 1\}, *, /)$, where

$*$	0	1
0	0	0
1	1	1

and

$/$	0	1
0	0	0
1	1	1

According to the two unary operation table, the following equations hold:

$$0 * 0 = 0$$
$$0 * 1 = 0$$
$$1 * 0 = 1$$
$$1 * 1 = 1$$
$$0 \,/\, 0 = 0$$
$$0 \,/\, 1 = 0$$
$$1 \,/\, 0 = 1$$
$$1 \,/\, 1 = 1$$

Another example of a quandle is the Tait Quandle $(\{0, 1, 2\}, *, /)$, where

$*$	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

and

$/$	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

Notice that both example quandles we have shown above have the same $*$ and / binary operations. However, this is not common among most quandles that have been dis-

covered. Most quandles have very different $*$ and / binary operations. An example is $(\{0,1,2,3,4\}, *, /)$, where

$*$	0	1	2	3	4
0	0	2	3	4	1
1	2	1	4	0	3
2	3	4	2	1	0
3	4	0	1	3	2
4	1	3	0	2	4

and

/	0	1	2	3	4
0	0	3	4	1	2
1	4	1	3	2	0
2	1	0	2	4	3
3	2	4	0	3	1
4	3	2	1	0	4

3.2.3 Results from the ASC Lab

The Bard College Laboratory for Algebraic and Symbolic Computation (ASC) has been conducting research concerning the theory quandles over the years. Below is a list of ASC lab research topics:

- Term Rewriting System[6]

  The lab discovered a term rewriting system, which is both confluent and terminating, for the first-order equational theory of quandles. This system allows one to compute unique normal forms for quandle expressions. Moreover, the system can be used to decide whether a given equation is also an identity for the theory. Related problems such as the Towers of Hanoi and related algebras such as racks are also in the mix. Here are some mathematics objects that we are interested in storing:

  - Quandles expressed as tables;
  - Rewriting rules expressed as pairs of expressions; and
  - Normal forms expressed as expressions.

- Word Problem[5]

  The Word Problem project reduced the word problem for recursively presented groups to the word problem for recursively presented quandles. It proves that, in general, the word problem for recursively presented quandles is not effectively computable. The feature of not-effectively-computable also extends to the word problem

for recursively presented racks as this project has also demonstrated.

Here are some mathematics objects that we are interested in storing:

- Finitely presented quandles expressed as tables;
- Groups expressed as expressions;
- Group quandles expressed as expressions; and
- Racks expressed as tables.

- Constraint Satisfaction Problem (CSP)[4]

  The lab has also considered the relation between constrain satisfaction problems and finite quandles. In particular, they proved that the tractability of a quandle is related to the connectivity of its right Cayley graph. It turns out that a strong notion of connectivity in the Cayley graph governs CSP dichotomy in the class of finite quandles.

  Here are some mathematics objects that we are interested in storing:

  - Cayley graphs expressed as graphs;
  - Malcev terms expressed as expressions; and
  - Merling terms expressed as expressions.

- QuandleViewer

  The QuandleViewer project has developed a graphic user interface written in Mathematica. It allows users to view the inputed quandles expressed as operation tables and Cayley graphs. The program takes the Mace4 portable format (.P), which contains information of quandles, as an input, and translates it to a Mace4-style Mathematica expression, which is called QuandleViewer format.

4
Quandles and Triples Transformation

4.1 Breaking Down A Quandle

Now we will show how to present a quandle as a list of triple lists in Mathematica. We will start with representing an operation table in the "list of lists" Mathematica format. Then we will show how to constitute the "list of lists" of a quandle.

4.1.1 From operation table to "list of lists" Mathematica format

An example of an operation table O is as below:

$*$	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

This is a binary operation with $*$ as the operation symbol. An operation table contains a column of size n, a header of size n and n rows of size n. When applying the operation, we choose the ith element from the column as a and the jth element from the header as b. Then $a * b$ equals the jth element from the ith row of the operation table. Using the operation table O, for example, we have the following equations:

$$
\begin{aligned}
0 * 0 &= 0 \\
0 * 1 &= 2 \\
0 * 2 &= 1 \\
1 * 0 &= 2 \\
1 * 1 &= 1 \\
1 * 2 &= 0 \\
2 * 0 &= 1 \\
2 * 1 &= 0 \\
2 * 2 &= 2
\end{aligned}
$$

We can break O into six parts:

- $*$: the operation type at the upper left.
- $\begin{array}{ccc} 0 & 1 & 2 \end{array}$: the row header RH at the upper right.
- $\begin{array}{c} 0 \\ 1 \\ 2 \end{array}$: the column header CH at the lower left.
- $\begin{array}{ccc} 0 & 2 & 1 \end{array}$: the first row $R1$ from the lower right table.
- $\begin{array}{ccc} 2 & 1 & 0 \end{array}$: the second row $R2$ from the lower right table.
- $\begin{array}{ccc} 1 & 0 & 2 \end{array}$: the third row $R3$ from the lower right table.

Thus, O can be described with six triples:

- $\{O$, operation type, $*\ \}$
- $\{O$, row header, $RH\ \}$
- $\{O$, column header, $CH\ \}$
- $\{O$, first row, $R1\ \}$
- $\{O$, second row, $R2\ \}$
- $\{O$, third row, $R3\ \}$

We can put these four triples into a list as below and form a list of list:

$$\begin{array}{lllll} \{ & \{O, & \text{operation type}, & *\}, \\ & \{O, & \text{row header}, & RH\}, \\ & \{O, & \text{column header}, & CH\}, \\ & \{O, & \text{first row}, & R1\}, \\ & \{O, & \text{second row}, & R2\}, \\ & \{O, & \text{third row}, & R3\} \quad \}. \end{array}$$

Figure 4.1.1.

Notice that some of the predicates contains ranking information such as "first row", "second row" and "third row". This is not appropriate. In order to avoid that, we introduce three intermediate identifiers $RT1$, $RT2$ and $RT3$. $RT1$ refers to the first row of O and $R1$ refer to the row $\quad 0 \quad 2 \quad 1$. Then we can turn the triple $\{O$, first row, $R1\}$ into three triples in Figure 4.1.2 in which none of the predicates contains the ranking information.

$$\begin{array}{lllll} \{ & \{O, & \text{row}, & RT1\}, \\ & \{RT1, & \text{place}, & 1\}, \\ & \{RT1, & \text{value}, & R1\} \quad \}, \end{array}$$

Figure 4.1.2.

We can apply the same method to the second and the third row of O and replace with $\{O$, second row, $R2$ $\}$ and $\{O$, third row, $R3$ $\}$ in the list in Figure 4.1.1. Thus, we can turn the list in Figure 4.1.1 to the list in Figure 4.1.3.

At this point, we still need to add more information of RH, CH, $R1$, $R2$ and $R3$ to the triple list. Let us look at RH first. RH can be break into 3 parts:

$$\begin{array}{lllll} \{ & \{RH, & \text{first element}, & 0\}, \\ & \{RH, & \text{second element}, & 1\}, \\ & \{RH, & \text{third element}, & 2\} \quad \}. \end{array}$$

Here the predicates contains ranking information. We can solve this problem by using the same approach we had for the rows. Then we can describe RH as below:

$$
\begin{array}{llll}
\{ & \{O, & \text{operation type}, & *\}, \\
 & \{O, & \text{header}, & RH\}, \\
 & \{O, & \text{column}, & CH\}, \\
 & \{O, & \text{row}, & RT1\}, \\
 & \{RT1, & \text{place}, & 1\}, \\
 & \{RT1, & \text{value}, & R1\}, \\
 & \{O, & \text{row}, & RT2\}, \\
 & \{RT2, & \text{place}, & 2\}, \\
 & \{RT2, & \text{value}, & R2\}, \\
 & \{O, & \text{row}, & RT3\}, \\
 & \{RT3, & \text{place}, & 3\}, \\
 & \{RT3, & \text{value}, & R3\} \quad \}.
\end{array}
$$

Figure 4.1.3.

$$
\begin{array}{llll}
\{ & \{RH, & \text{element}, & ERH1\}, \\
 & \{ERH1, & \text{place}, & 1\}, \\
 & \{ERH1, & \text{value}, & 0\}, \\
 & \{RH, & \text{element}, & ERH2\}, \\
 & \{ERH2, & \text{place}, & 2\}, \\
 & \{ERH2, & \text{value}, & 1\}, \\
 & \{RH, & \text{element}, & ERH3\}, \\
 & \{ERH3, & \text{place}, & 3\}, \\
 & \{ERH3, & \text{value}, & 2\} \quad \}
\end{array}
$$

Figure 4.1.4.

Then we can include the detailed information of H by joining the list in Figure 4.1.4 to the list in Figure 4.1.3. Notice that, similar to H, C, $R1$, $R2$ and $R3$ are all lists with a sequence of numbers. Thus, we can handle them in the same way that we handle H. Therefore, we can derive a list with the complete information in the end. The complete list is shown as in Figure 4.1.6. In Mathematica, the list of lists would have the representation as suggested below:

```
{{O, operation type, *}, {O, header row, RH}, {RH, element, ERH1}, {ERH1, place, 1},
 {ERH1, value, 0}, {RH, element, ERH2}, {ERH2, place, 2}, {ERH2, value, 1}, {RH, element, ERH3},
⋮
 {R3, element, ER31}, {ER31, place, 1}, {ER31, value, 1}, {R3, element, ER32},
 {ER32, place, 2}, {ER32, value, 0}, {R3, element, ER33}, {ER33, place, 3}, {ER33, value, 2}}
```

Figure 4.1.5.

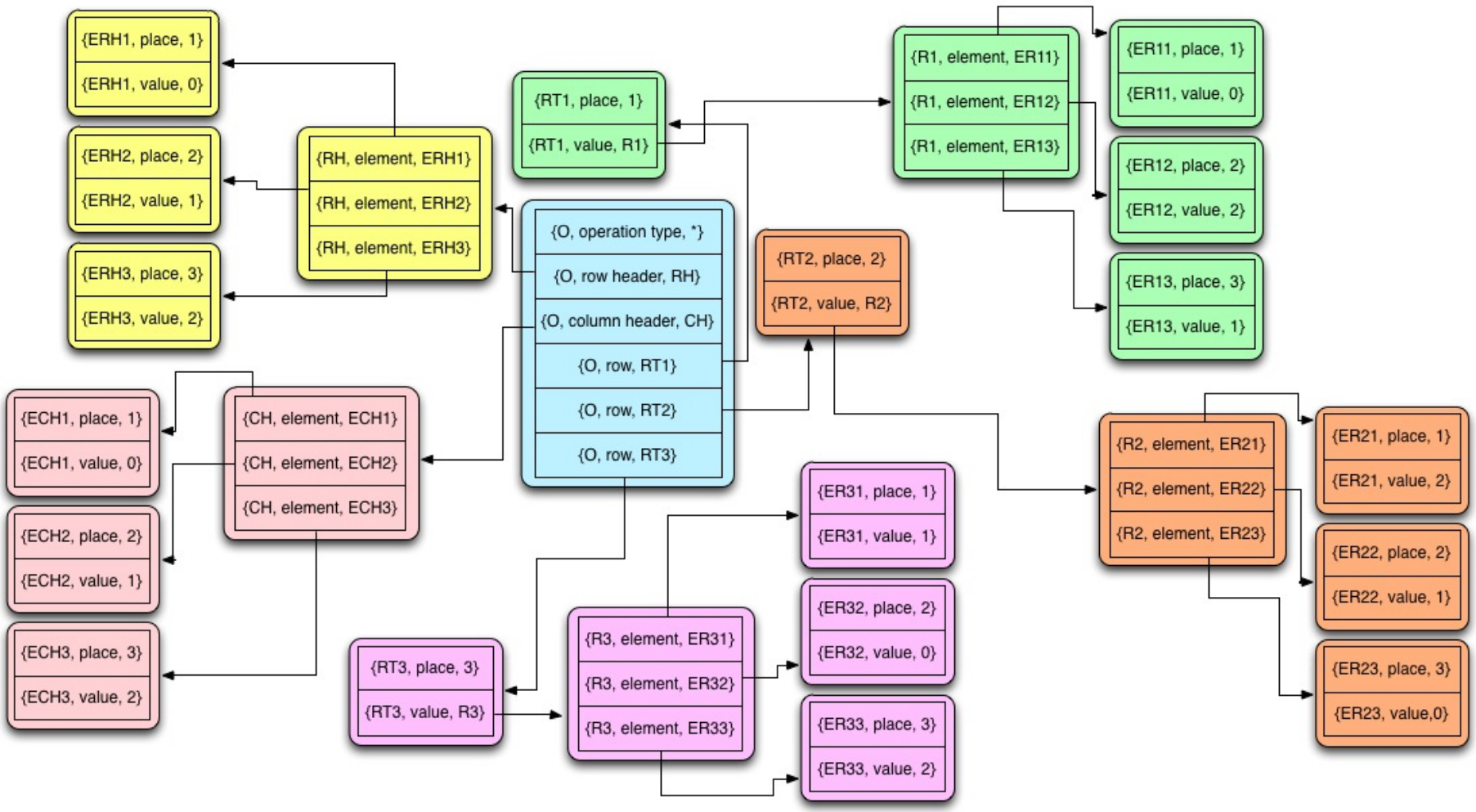
{ERH1, place, 1}
{ERH1, value, 0}
{ERH2, place, 2}
{ERH2, value, 1}
{ERH3, place, 3}
{ERH3, value, 2}
{RH, element, ERH1}
{RH, element, ERH2}
{RH, element, ERH3}
{RT1, place, 1}
{RT1, value, R1}
{R1, element, ER11}
{R1, element, ER12}
{R1, element, ER13}
{ER11, place, 1}
{ER11, value, 0}
{ER12, place, 2}
{ER12, value, 2}
{ER13, place, 3}
{ER13, value, 1}
{O, operation type, *}
{O, row header, RH}
{O, column header, CH}
{O, row, RT1}
{O, row, RT2}
{O, row, RT3}
{RT2, place, 2}
{RT2, value, R2}
{ECH1, place, 1}
{ECH1, value, 0}
{ECH2, place, 2}
{ECH2, value, 1}
{ECH3, place, 3}
{ECH3, value, 2}
{CH, element, ECH1}
{CH, element, ECH2}
{CH, element, ECH3}
{R2, element, ER21}
{R2, element, ER22}
{R2, element, ER23}
{ER21, place, 1}
{ER21, value, 2}
{ER22, place, 2}
{ER22, value, 1}
{ER23, place, 3}
{ER23, value,0}
{ER31, place, 1}
{ER31, value, 1}
{ER32, place, 2}
{ER32, value, 0}
{ER33, place, 3}
{ER33, value, 2}
{RT3, place, 3}
{RT3, value, R3}
{R3, element, ER31}
{R3, element, ER32}
{R3, element, ER33}

Figure 4.1.6.

4.1.2 From quandle to "list of lists" Mathematica format

By definition, A **quandle** $Q = (S, *, /)$ contains a set S and two binary operations $*$, $/$. Each operation can be represented using a operation table. For example, the Tait Quandle $TQ = (\{0, 1, 2\}, *, /)$, where

$*$	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

$/$	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

We can break TQ into 3 parts:

- $\{0, 1, 2\}$: the set S
$*$	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

  : the $*$ operation $OP1$
$/$	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

  : the $/$ operation $OP2$

Thus, the triple representation is:

$$\begin{array}{lllll} \{ & \{TQ, & \text{set}, & S\}, & \\ & \{TQ, & \text{first operation}, & OP1\}, & \\ & \{TQ, & \text{second operation}, & OP2\} & \} \end{array}$$

Figure 4.1.7.

The list in Figure 4.1.7 can then be modified to:

$$\begin{array}{lllll} \{ & \{TQ, & \text{set}, & S\}, & \\ & \{TQ, & \text{operation}, & OPTQ1\}, & \\ & \{OPTQ1, & \text{place}, & 1\}, & \\ & \{OPTQ1, & \text{value}, & OP1\}, & \\ & \{TQ, & \text{operation}, & OPTQ2\}, & \\ & \{OPTQ2, & \text{place}, & 2\}, & \\ & \{OPTQ2, & \text{value}, & OP2\} & \}, \end{array}$$

Figure 4.1.8.

in order to avoid including the ranking information in the predicates. Then we need to fill in the information of S, $OP1$ and $OP2$ by appending more triple lists to the list in Figure 4.1.8.

Notice that $OP1$ is same as the example operation table we showed earlier. Thus, we can describe it as we did in Figure 4.1.6. The only change we are making is replacing O with $OP1$ because we refer to the operation table as $OP1$ now. $OP2$ shares the same header, column and rows as $OP1$ except for the operation type. Thus, we can add the information of $OP2$ by appending these triple lists to the list in Figure 4.1.6:

$$
\begin{array}{llll}
\{ & \{OP2, & \text{operation type}, & /\}, \\
 & \{OP2, & \text{header}, & RH\}, \\
 & \{OP2, & \text{column}, & CH\}, \\
 & \{OP2, & \text{row}, & RTT1\}, \\
 & \{RTT1, & \text{place}, & 1\}, \\
 & \{RTT1, & \text{value}, & R1\}, \\
 & \{OP2, & \text{row}, & RTT2\}, \\
 & \{RTT2, & \text{place}, & 2\}, \\
 & \{RTT2, & \text{value}, & R2\}, \\
 & \{OP2, & \text{row}, & RTT3\}, \\
 & \{RTT3, & \text{place}, & 3\}, \\
 & \{RTT3, & \text{value}, & R3\} \qquad \}.
\end{array}
$$

Figure 4.1.9.

Notice here we do not need to include the information of RH, CH, $R1$, $R2$ and $R3$ any more because this information is already existed in the list in Figure 4.1.6. Joining the lists in Figure 4.1.6 and 4.1.9 would give us the complete information of the two binary operations $OP1$ and $OP2$. The lists are shown in Figure 4.1.10.

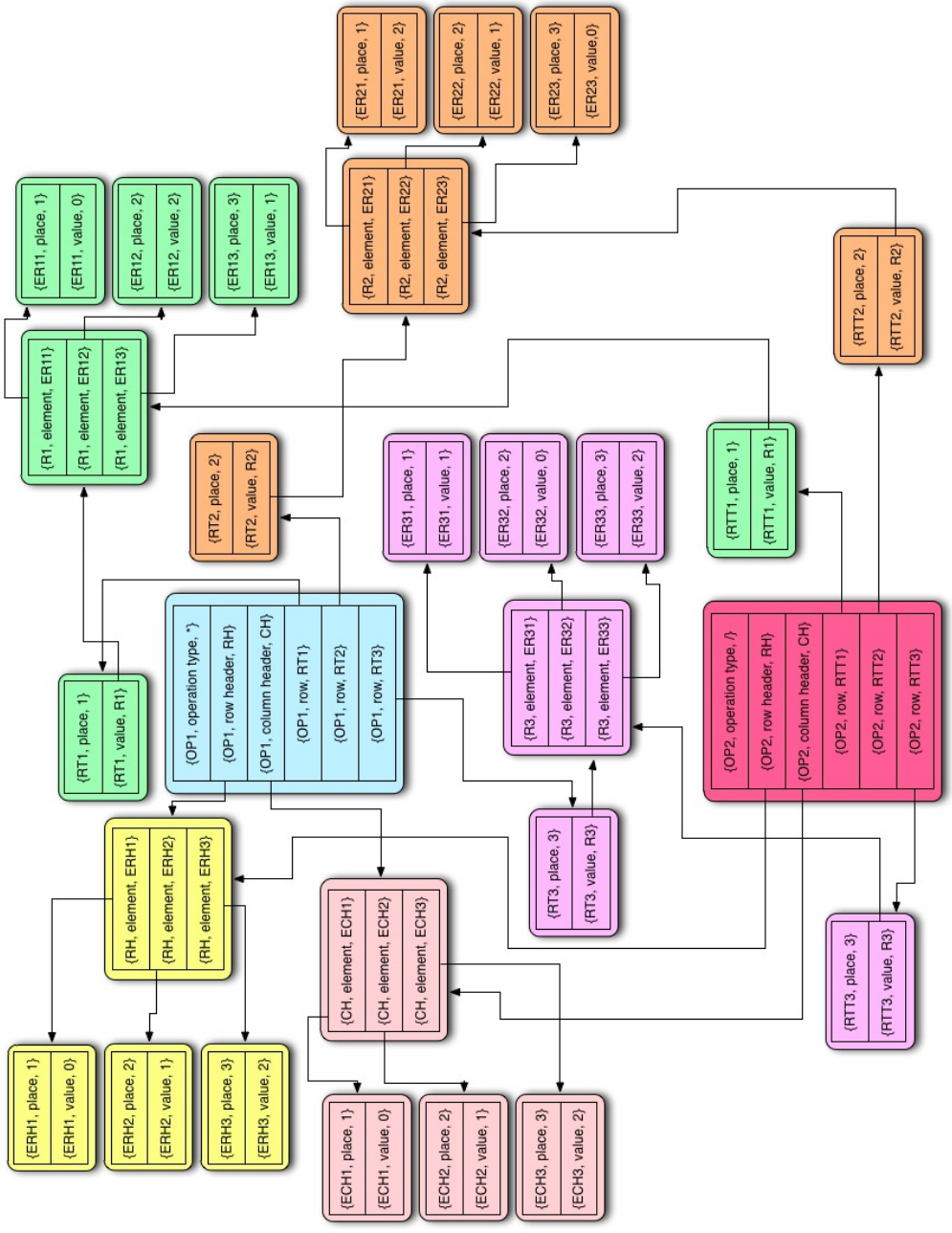
{ER21, place, 1}
{ER21, value, 2}
{ER22, place, 2}
{ER22, value, 1}
{ER23, place, 3}
{ER23, value,0}
{ER11, place, 1}
{ER11, value, 0}
{ER12, place, 2}
{ER12, value, 2}
{ER13, place, 3}
{ER13, value, 1}
{R2, element, ER21}
{R2, element, ER22}
{R2, element, ER23}
{RTT2, place, 2}
{RTT2, value, R2}
{R1, element, ER11}
{R1, element, ER12}
{R1, element, ER13}
{RT2, place, 2}
{RT2, value, R2}
{ER31, place, 1}
{ER31, value, 1}
{ER32, place, 2}
{ER32, value, 0}
{ER33, place, 3}
{ER33, value, 2}
{RTT1, place, 1}
{RTT1, value, R1}
{RT1, place, 1}
{RT1, value, R1}
{OP1, operation type, *}
{OP1, row header, RH}
{OP1, column header, CH}
{OP1, row, RT1}
{OP1, row, RT2}
{OP1, row, RT3}
{R3, element, ER31}
{R3, element, ER32}
{R3, element, ER33}
{OP2, operation type, /}
{OP2, row header, RH}
{OP2, column header, CH}
{OP2, row, RTT1}
{OP2, row, RTT2}
{OP2, row, RTT3}
{RT3, place, 3}
{RT3, value, R3}
{RH, element, ERH1}
{RH, element, ERH2}
{RH, element, ERH3}
{CH, element, ECH1}
{CH, element, ECH2}
{CH, element, ECH3}
{RTT3, place, 3}
{RTT3, value, R3}
{ERH1, place, 1}
{ERH1, value, 0}
{ERH2, place, 2}
{ERH2, value, 1}
{ERH3, place, 3}
{ERH3, value, 2}
{ECH1, place, 1}
{ECH1, value, 0}
{ECH2, place, 2}
{ECH2, value, 1}
{ECH3, place, 3}
{ECH3, value, 2}

Figure 4.1.10.

The last thing we need to include from the Tait Quandle is the set S. We can describe S within 3 more triple lists:

$$\begin{array}{lllll} \{ & \{S, & \text{element}, & 0\}, \\ & \{S, & \text{element}, & 1\}, \\ & \{S, & \text{element}, & 2\} & \} \end{array}$$

Figure 4.1.11.

Hence, by joining the lists in Figure 4.1.8, 4.1.10 and 4.1.11, we would get the list that contains the complete information of the Tait Quandle in Figure 4.1.12

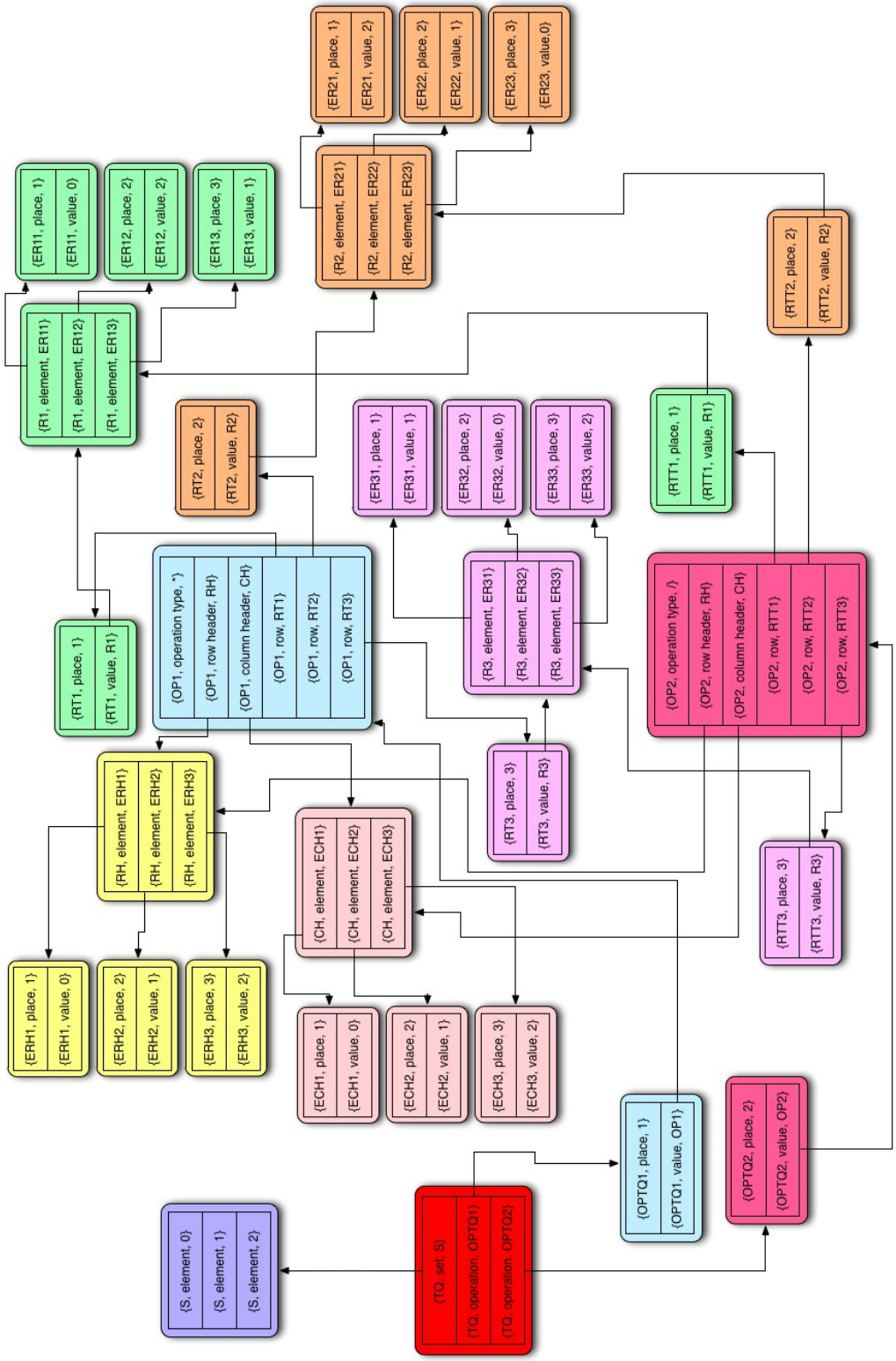
{ER21, place, 1}
{ER21, value, 2}
{ER22, place, 2}
{ER22, value, 1}
{ER23, place, 3}
{ER23, value,0}
{R2, element, ER21}
{R2, element, ER22}
{R2, element, ER23}
{ER11, place, 1}
{ER11, value, 0}
{ER12, place, 2}
{ER12, value, 2}
{ER13, place, 3}
{ER13, value, 1}
{R1, element, ER11}
{R1, element, ER12}
{R1, element, ER13}
{RTT2, place, 2}
{RTT2, value, R2}
{RT2, place, 2}
{RT2, value, R2}
{ER31, place, 1}
{ER31, value, 1}
{ER32, place, 2}
{ER32, value, 0}
{ER33, place, 3}
{ER33, value, 2}
{RTT1, place, 1}
{RTT1, value, R1}
{RT1, place, 1}
{RT1, value, R1}
{OP1, operation type, *}
{OP1, row header, RH}
{OP1, column header, CH}
{OP1, row, RT1}
{OP1, row, RT2}
{OP1, row, RT3}
{R3, element, ER31}
{R3, element, ER32}
{R3, element, ER33}
{OP2, operation type, /}
{OP2, row header, RH}
{OP2, column header, CH}
{OP2, row, RTT1}
{OP2, row, RTT2}
{OP2, row, RTT3}
{RT3, place, 3}
{RT3, value, R3}
{RH, element, ERH1}
{RH, element, ERH2}
{RH, element, ERH3}
{CH, element, ECH1}
{CH, element, ECH2}
{CH, element, ECH3}
{RTT3, place, 3}
{RTT3, value, R3}
{ERH1, place, 1}
{ERH1, value, 0}
{ERH2, place, 2}
{ERH2, value, 1}
{ERH3, place, 3}
{ERH3, value, 2}
{ECH1, place, 1}
{ECH1, value, 0}
{ECH2, place, 2}
{ECH2, value, 1}
{ECH3, place, 3}
{ECH3, value, 2}
{OPTQ1, place, 1}
{OPTQ1, value, OP1}
{OPTQ2, place, 2}
{OPTQ2, value, OP2}
{S, element, 0}
{S, element, 1}
{S, element, 2}
{TQ, set, S}
{TQ, operation, OPTQ1}
{TQ, operation, OPTQ2}

Figure 4.1.12.

4.2 Transformation Implementation

4.2.1 From Mathematica list of triple lists to RDF

In Mathematica[3], quandle information can be stored as a list of triple lists as above.

The input format is the a list of triples in the Mathematica list format:

```
{{http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_1, place, 1},
 {http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_1, value, 1},
 {http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_2, place, 2},
 {http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_2, value, 2},
⋮
{http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1, element,
 http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_1},
{http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1, element,
 http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_2},
{http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1, element,
 http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_3},
{http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1, size, 3}}
```

Figure 4.2.1.

As we can see, this format contains two levels of lists.

- First level: each individual triple list;
- Second level: the collection of all triples.

To translate the triples in Mathematica list format to the RDF format, we need to first change the structure of the list and make it a list of three levels as below:

```
{{{http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_1, place, 1},
   {http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_1, value, 1}},
  {{http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_2, place, 2},
   {http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_2, value, 2}},
⋮
 {{http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1, element,
   http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_1},
  {http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1, element,
   http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_2},
  {http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1, element,
   http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_3},
  {http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1, size, 3}}}
```

Figure 4.2.2.

The new list contains three levels:

- First level: each individual triple list;
- Second level: the collection of all triples that contains the same subject, which is the first element of an individual triple list;
- Third level is the collection of all the second-level lists.

The translation of the two-level list to the three-level list could be done by calling $GatherBy[\_, First]$ in Mathematica. This will also sort each secondary list according to the second element of each individual triple. The new list format would be helpful for the actual translation from the Mathematica triple in list format to the RDF format.

With the three-level list,

1. We create an empty list called $RDFElementList$.
2. We loop through all the secondary lists. For each secondary list, we create a temporary empty list called $tmpRDFElementList$.
3. Then we loop through all the triple lists in the secondary list. For each individual triple list, we create an XML tag with the predicate, which is the second element of the triple list, as the tag name and the object, which is the third element of the triple list, as the content of the "rdf:resource" attribute of the tag.
4. We append the XML tags we create for each individual triple list to $tmpRDFElementList$. Step 2 through 4 are implemented in the code in Figure 4.2.3.
5. We then return to the secondary level. We obtain the subject, which is the first element of an individual triple list. Since all the triple lists inside a secondary list contains the same subject, it does not matter which individual triple list is chosen.

```
tmpRDFElementList = {};
tmpList = list[[i]];

For[j = 1, j ≤ Length[tmpList], j ++,
 Module[{},
  tmpRDFElementList = Append[tmpRDFElementList,
    XMLElement[tmpList[[j]][[2]], {"rdf:resource" → tmpList[[j]][[3]]}, {}]
   ]
 ]
];
```

Figure 4.2.3.

With the subject value we then create an XML tag with "rdf:Description" as the tag name, the subject value as the content of the "rdf:about" attribute of the tag and the temporary list we have above as the content of the tag.

6. We append the XML tag we create for each secondary list to *RDFElementList*.

Step 5 through 6 are implemented in the code in Figure 4.2.4.

```
RDFElementList = Append[RDFElementList,
  XMLElement["rdf:Description",
   {"rdf:about" → tmpList[[1]][[1]]},
   tmpRDFElementList]
 ]
```

Figure 4.2.4.

The resulting RDF representation from Figure 4.2.1 is shown in Figure 4.2.5.

To complete the RDF file, the last thing we need to do is adding a parent XML tag around the XML tags in Figure 4.2.5.

```
RDFElementList = XMLElement["rdf:RDF", {"xmlns:rdf" → "&rdf;"}, RDFElementList];
```

Figure 4.2.6.

```
– <rdf:Description rdf:about="http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_1">
    <place rdf:resource="1"/>
    <value rdf:resource="1"/>
  </rdf:Description>
– <rdf:Description rdf:about="http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_2">
    <place rdf:resource="2"/>
    <value rdf:resource="2"/>
  </rdf:Description>
⋮
– <rdf:Description rdf:about="http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1">
    <element rdf:resource="http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_1"/>
    <element rdf:resource="http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_2"/>
    <element rdf:resource="http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_3"/>
    <size rdf:resource="3"/>
  </rdf:Description>
```

Figure 4.2.5.

The complete RDF file would have the representation as below:

```
– <rdf:RDF>
  – <rdf:Description rdf:about="http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_1">
      <place rdf:resource="1"/>
      <value rdf:resource="1"/>
    </rdf:Description>
  – <rdf:Description rdf:about="http://www.asclab.org/zheng/quandles/3/operationComponent/column/1/element_2">
      <place rdf:resource="2"/>
      <value rdf:resource="2"/>
    </rdf:Description>
⋮
  – <rdf:Description rdf:about="http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1">
      <element rdf:resource="http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_1"/>
      <element rdf:resource="http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_2"/>
      <element rdf:resource="http://www.asclab.org/zheng/quandles/3/quandleComponent/set/1/element_3"/>
      <size rdf:resource="3"/>
    </rdf:Description>
  </rdf:RDF>
```

Figure 4.2.7.

5
Conclusion

This project looks into the problem of storing and sharing mathematics information on the World Wide Web. In particular, we consider the encoding of finite algebras from universal algebras as triples for the Semantic Web. The project focuses on the collection of finite quandles, which is a class of finite algebras.

Included are various pieces of software that have been developed to translate between different formats for expressing quandles. The target formats involved are listed as below:

- Mace4 Portable Format (.P). See example in Figure 5.0.1.

```
interpretation( 3, [number = 5,seconds = 0], [
      function(*(_,_), [0,2,1,2,1,0,1,0,2]),
      function(/(_,_), [0,2,1,2,1,0,1,0,2])]).
```

Figure 5.0.1.

- QuandleViewer Format (Mace4-style Mathematica Expression). See example in Figure 5.0.2.

```
interpretation[3, 5, 0, {
    function[∗, 2, {{1, 3, 2}, {3, 2, 1}, {2, 1, 3}}],
    function[/, 2, {{1, 3, 2}, {3, 2, 1}, {2, 1, 3}}]}]
```

Figure 5.0.2.

- Prolog-style Triples. See example in Figure 5.0.3.

```
rdf('I', 'own', 'my apartment').
```

Figure 5.0.3.

- RDF-XML

Moreover, we have the following intermediate formats to simplify the translation functions:

- Mathematica List of Mathematica-style Triples
- Mathematica List of List of Triple-link
- Mathematica List of Prolog-style Triple Strings

The translation relations are shown in the Figure 5.0.4.

The work that has been done in this project sets a foundation for representing all levels of properties of quandles in the Semantic Web. To be able to store quandle objects in the Semantic Web allows us to append information that states certain properties and relations about the quandle objects in the linked database.

5.1 Future Work

Natural projects that can continue from this project include the following:

- Storing some representations of expression such as equations, presentations, and rewriting systems;

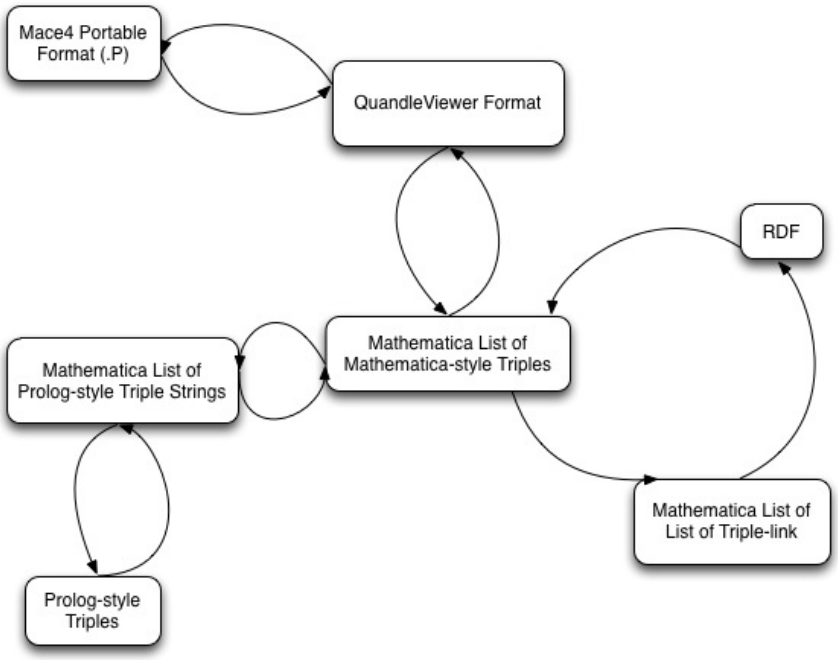

Figure 5.0.4.

- Storing first-order properties of quandles;
- Storing second-order properties of quandles;
- Storing other algebra objects;
- Storing information of relationships between different kinds of algebras;
- Develop software that can search and retrieve quandle information from the Semantic Web; and
- Develop a trust system that store proofs in the Semantic Web style and does verification of the proofs.

Bibliography

[1] Toby Segaran, Colin Evans, and Jamie Taylor, *Programming the Semantic Web*, O'Reilly Media, Sebastopol, CA, 2009.

[2] Tim Berners-Lee and Mark Fischetti, *Weaving the Web*, HarperSanFrancisco, Chapter 12, 1999.

[3] Stephen Wolfram, *The Mathematica Book*, Wolfram Media, Incorporated, 2003.

[4] Brita Brudvig, Benjamin Fish, Solomon Garber, Max Jeter, Liwen Song, and Bob McGrail, *CSPs and Connectedness: P/NP-complete Dichotomy for Idempotent, Right Quasigroups* (2012).

[5] James Belk and Bob McGrail, *The General Word Problem for Quandles is Undecidable* (2012).

[6] Peter Golbus, Claudio Gutierrez, Bob McGrail, and Hannah Vallee, *Quandles, Term Rewriting Systems, and the Towers of Hanoi* (2011).

[7] James Hendler, Tim Berners-Lee, and Eric Miller, *Integrating Applications on the Semantic Web*, Journal of the Institute of Electrical Engineers of Japan **Vol 122(10)** (2002), 676–680.

[8] W3C Team Submission. *Turtle - Terse RDF Triple Language*, `http://www.w3.org/TeamSubmission/turtle/`. Accessed (April 24, 2013).

[9] W3C Recommendation. *RDF/XML Syntax Specification (Revised)*, `http://www.w3.org/TR/rdf-syntax-grammar/`. Accessed (April 24, 2013).

[10] *The 9th International Conference on Mathematical Knowledge Management*, `http://cicm2010.cnam.fr/mkm/`. Accessed (April 25, 2013).

[11] *DML 2010, 3nd workshop/conference: Towards a Digital Mathematics Library*, `http://www.fi.muni.cz/~sojka/dml-2010.html\#topics`. Accessed (April 25, 2013).

[12] *OpenMath Website*, `http://www.openmath.org/index.html`. Accessed (April 25, 2013).

[13] *American Mathematical Society (AMS)*, `http://www.ams.org/home/page`. Accessed (April 28, 2013).

[14] *UACalc, A Universal Algebra Calculator*, `http://uacalc.org/`. Accessed (April 28, 2013).

[15] *The KnotPlot Site*, `http://www.knotplot.com/`. Accessed (April 28, 2013).

[16] *Prover9 and Mace4*, `http://www.cs.unm.edu/~mccune/mace4/`. Accessed (April 28, 2013).

[17] *Wolfram Mathematica: Technical Computing Software*, `http://www.wolfram.com/mathematica/`. Accessed (April 28, 2013).

[18] *Maple 17 - Technical Computing Software for Engineers*, `http://www.maplesoft.com/products/maple/`. Accessed (April 28, 2013).

[19] *MATLAB - The Language of Technical Computing*, `http://www.mathworks.com/products/matlab/`. Accessed (April 28, 2013).

[20] *Wolfram—Alpha: Computational Knowledge Engine*, `http://www.wolframalpha.com/`. Accessed (April 28, 2013).

[21] *Wolfram MathWorld: The Web's Most Extensive Mathematics Resource*, `http://mathworld.wolfram.com/`. Accessed (April 28, 2013).

[22] *Knot Atlas: Main Page*, `http://katlas.math.toronto.edu/wiki/Main_Page`. Accessed (April 28, 2013).

[23] *Wikipedia*, `http://www.wikipedia.org/`. Accessed (April 28, 2013).