COMPUTING RELATIVELY LARGE ALGEBRAIC STRUCTURES BY AUTOMATED THEORY EXPLORATION

by

QURATUL-AIN MAHESAR

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
March 2014

Abstract

Automated reasoning technology provides means for inference in a formal context via a multitude of disparate reasoning techniques. Combining different techniques not only increases the effectiveness of single systems but also provides a more powerful approach to solving hard problems. Consequently combined reasoning systems have been successfully employed to solve non-trivial mathematical problems in combinatorially rich domains that are intractable by traditional mathematical means. Nevertheless, the lack of domain specific knowledge often limits the effectiveness of these systems. In this thesis we investigate how the combination of diverse reasoning techniques can be employed to pre-compute additional knowledge to enable mathematical discovery in finite and potentially infinite domains that is otherwise not feasible.

In particular, we demonstrate how we can exploit bespoke symbolic computations and automated theorem proving to automatically compute and evolve the structural knowledge of small size finite structures in the algebraic theory of quasigroups. This allows us to increase the solvability horizon of model generation systems to find solution models for large size finite algebraic structures previously unattainable.

We also present an approach to exploring infinite models using a mixture of automated tools and user interaction to iteratively inspect the structure of solutions and refine search. A practical implementation combines a specialist term rewriting system with bespoke graph algorithms and visualization tools and has been applied to solve the generalized version of Kuratowski's classical closure-complement problem from point-set topology that had remained open for several years.

ACKNOWLEDGEMENTS

I owe my deepest gratitude to my supervisor Dr. Volker Sorge for the excellent knowledge, enthusiastic encouragement and constant support over the years. His guidance and advise have had a profound influence on the development of my research. I will always cherish the work I did with him.

I would also like to acknowledge the efforts of my thesis group members Professor Jonathan Rowe and Professor Achim Jung in continually reviewing my research progress and giving productive feedback throughout this work. I also acknowledge the ORS Awards Scheme and School of Computer Science, University of Birmingham for supporting me financially.

Thanks are also due to my examiners Professor Simon Colton and Dr. Manfred Kerber, as well as chair Dr. Rami Bahsoon, for taking time out of their busy schedule to examine my thesis. Finally, I would like to express my deepest appreciation to my family, friends and departmental colleagues for their moral support and encouragement.

This thesis is dedicated to my father Dr. Muhammad Usman Mahesar who passed away during my Ph.D. and would have been extremely happy and proud of my achievement.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

Introduction, Related Work and Reasoning Background

CHAPTER 1

INTRODUCTION

Chapter Overview: *This chapter presents the scope of this thesis. The chapter motivates the reader by introducing the role played by automated reasoning systems to solve open problems in mathematics. The major contributions made by this thesis are summarized followed by the list of publications. Finally, an overview of the structure and organization of this thesis is given.*

1.1 Motivation

Automated reasoning systems can be very useful in solving complex problems in mathematical domains in many cases where it is infeasible to compute solutions manually. They can be used in a variety of ways to accomplish particular goals. One of the primary goals of such systems is to prove conjectures i.e. claims for which a proof is not yet known. The existence of certain algebraic structures can also be proved by finding solution models that satisfy the axiomatic definition of the algebraic

structures. The complement of this activity is disproving a conjectured theorem by finding a model or counter example. The problems involving the classification and enumeration of algebra can also be solved by using these systems. However, there are certain limitations due to combinatorial complexity, where the search space can be out of the scope of the current automated reasoning systems; for example when generating algebraic structures with non-trivial properties of larger sizes. These limitations can be overcome by exploring mathematical techniques that pre-compute additional knowledge to restrict the search space in sufficiently diverse domains.

The process of using automated reasoning and other mathematical tools for exploring mathematical theories as defined in [Buc06], consists of the invention of notions, the invention and proof of propositions (lemmas, theorems), the invention of problems, and the invention and verification of methods (algorithms) that solve problems. Mathematical reasoning as described in [Bun85], consists of simultaneous automation of various reasoning processes e.g. learning, theorem proving, model search etc. Each reasoning process requires an input and outputs knowledge. The input knowledge of one technique is the output knowledge of another, where the techniques form an intercommunicating network or a combined reasoning system. The power of such a system in which various techniques interact in well-crafted ways is greater compared to just the sum of the power of the parts.

Within this thesis, we demonstrate the use of diverse automated reasoning tools in solving complex problems in mathematics in particular to prove the existence of certain algebraic structures. The main aim is to compute knowledge by automated means that aids in the generation of solutions in finite domains; and to use human inspection to discover

knowledge that helps to push the boundaries of mathematical discovery in infinite domains. Our first approach automatically explores structural domain knowledge of algebra via symbolic computations and automated theorem proving to increase the solving horizon of various automated reasoning systems to find solution models of large size finite structures. Our second approach uses active involvement of the user in the system combination, where the user can aid in the discovery of knowledge by careful inspection of the structure of solutions which is given as feedback to the system combination. This has allowed us to generate approximate solutions to a class of problems in topological domains that are of infinite nature. Furthermore, we have implemented a specialist term rewriting system that makes use of regular expressions that are based on the discovered knowledge by the human inspection of the solutions. Moreover, we have performed experiments to evaluate the effectiveness of our approaches in solving real mathematical problems.

1.2 Hypotheses

The aim of this thesis is to address the following hypotheses:

- The combination of diverse automated reasoning techniques and other mathematical software tools can push the boundaries of mathematical discovery by generating and structuring additional knowledge.

- Discovery of solutions for large size examples in a finite discrete domain can be aided by automated theory exploration via symbolic computations and automated theorem proving that pre-compute additional knowledge.

- Solutions in potentially infinite domains can be approximated by

exploiting the structural knowledge using diverse systems such as a specialist term rewriting system and visualization tools where active involvement of the user is a necessity.

1.3 Contributions

A summary of the contributions of this thesis is as follows;

1. We describe two novel approaches that exploit the structural knowledge of finite algebra to assist model generation systems in finding solution models for large size algebraic structures.

2. We perform a comparative experimental analysis of diverse automated reasoning techniques to generate quasigroup structures that have certain non-trivial properties.

3. We describe a novel rule based term rewriting system to find approximate solutions to the infinite cases of the generalization of the Kuratowski problem in point free topology. Our term rewriting system exploits the regular expressions that were identified after careful inspection of the intermediate graphs produced by our system, that has helped us to close a problem that remained open for several years.

4. We describe the formation of combined reasoning systems in solving complex mathematical problems where each system performs a distinct task and the combination of these systems allows a powerful approach to computing the solutions. Furthermore, we define a system combination where the user acts as a component within the system to inspect the solutions for the discovery of useful knowledge.

1.4 Publications

This thesis is based partly upon the following conference and workshop publications.

- **Quratul-ain Mahesar** and Volker Sorge "Algebraic Theory Exploration: A Comparison of Technologies", Proceedings of the 14^{th} International Symposium on Symbolic and Numeric Algorithms for Scientific Computing 2012 [MS12a]

- Osama Al-Hassani, **Quratul-ain Mahesar**, Claudio Sacerdoti Coen and Volker Sorge "A Term Rewriting System for Kuratowski's Closure-Complement Problem", Proceedings of the 23^{rd} International Conference on Rewriting Techniques and Applications 2012 [AHMCS12b]

- **Quratul-ain Mahesar** and Volker Sorge "Generation of Large Size Quasigroup Structures Using Algebraic Constraints", Proceedings of the 19^{th} Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice, 2012 [MS12b]

- O. Al-Hassani, **Q. Mahesar**, C. Sacerdoti Coen and V. Sorge "Solving Kuratowski Problems by Term Rewriting", Proceedings of the 19^{th} Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice, 2012 [AHMCS12a]

- **Quratul-ain Mahesar** and Volker Sorge "Property Preserving Generation of Large Size Quasigroup-structures", Proceedings of the 17^{th} Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice, 2010 [MS10]

- **Quratul-ain Mahesar** and Volker Sorge "Classification of

Quasigroup-structures with respect to their Cryptographic Properties", Proceedings of the 16^{th} Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice, 2009 [MS09]

1.5 Overview and Structure

This thesis is based on four parts which are described as follows:
Part I consists of chapters 1, 2 and 3 that provide the overview of the background and related work relevant to the topic of this thesis.

- Chapter 1 gives an overview of the main objectives of the thesis, motivates the reader for the importance of the work done, and provides a summary of the contributions and a list of publications.
- Chapter 2 gives a discussion about the related work that has been previously done using automated reasoning techniques in mathematics.
- Chapter 3 consists of two sections. The first section provides a description on the major logical systems such as propositional logic, first order logic and equational logic. The second section describes the various automated reasoning techniques we have used in our research study, such as automated theorem proving, constraint solving, satisfiability solving, model finding, and term rewriting.

Part II consists of chapters 4, 5, 6 and 7 that provide the relevant background for quasigroups, quasigroup model generation problems and their encodings, the approaches we have proposed and finally the experiments and results.

- Chapter 4 provides the background on the domain of quasigroups

such as a formal definition for quasigroups along with their operations and properties.

- Chapter 5 describes the three main quasigroup model generation problems i.e. constraint satisfaction problems, satisfiability problems and model generation problems. For each of these types of problems, the formulation of encodings for the systems used are given and the corresponding output solutions are shown.

- Chapter 6 describes the two proposed approaches for quasigroup element pre-computations that use symbolic computations and automated theorem proving.

- Chapter 7 describes the experimental set-up that combines symbolic computations with automated reasoning systems to compute quasigroups with interesting non-trivial properties. We then present a discussion on the results obtained.

Part III consists of chapters 8, 9, 10 and 11 that provide background on point-set topology, the Kuratowski closure-complement problem and its generalization, and the term rewriting system that we have proposed and implemented for solving the problem.

- Chapter 8 provides the basic notions of point-set topology and describes the Kuratowski closure-complement problem.

- Chapter 9 discusses the generalization of the Kuratowski closure-complement problem to point free topology using the inference rules of intuitionistic logic and describes the n^{th} approximation to the problem.

- Chapter 10 describes the term rewriting system we have proposed to solve the generalized Kuratowski problem and its variations.

- Chapter 11 describes the methodology, implementation details, results and verification of results.

Part IV is the conclusion that consists of chapters 12 and 13 that present the scientific contributions of the research and directions for future work.

CHAPTER 2

RELATED WORK - AUTOMATED REASONING IN MATHEMATICS

Chapter Overview: *This chapter gives an overview of how reasoning systems such as model generation, automated theorem proving, constraint satisfaction, SAT solving techniques, computer algebra techniques and machine learning have been used previously to produce results in the field of mathematics. In particular the focus is on: existence and combinatorial completion problems, quantitative enumeration and qualitative classification of finite algebras.*

In this chapter, we first describe how different automated reasoning systems have been used previously to solve some open existence problems in finite algebra such as quasigroups, loops, groups and rings. Furthermore, we present an analysis of different approaches that have been proposed to solve the combinatorial problem of completion where one needs to construct the full solution for the problem where there exist

partial element assignments beforehand. It is also useful to find out the number of solution instances that exist for a certain class of algebra of a particular size. We present the different techniques that have been used for the quantitative enumeration of finite algebras such as monoids, semigroups and ag-groups. Moreover, we also present the approaches used for qualitative classification of algebra which not only describes the number of equivalence classes but also specifies the discriminating properties that tell us how the classes differ from each other.
The reader should refer to Chapter 4 for background on finite algebraic structures in particular the quasigroup problems that we mention in the text of this chapter. Chapter 3 should be referred to for the background on different automated reasoning techniques.

2.1 Existence Problems

In mathematics, there are many open problems that are concerned with the existence of an algebraic structure having a particular size exhibiting particular properties. These existence problems have been successfully solved previously by using different automated reasoning tools.
[FSB93, Mcc94, SFS95, ZS94, ZH94] show how advanced automated reasoning techniques can be used to solve the existence of combinatorial problems of quasigroups. In [FSB93, SFS95] Fujitsa uses MGTP, a model-generation based first-order theorem prover and Slaney uses FINDER, a program based on constraint solving to solve several open problems in quasigroup theory producing new results such as:

- Two non-isomorphic Idempotent Qg-1 (Schröder) quasigroups of order 12.

- Qg-2 (Stein's third law) quasigroup of order 12.

- Qg-5 quasigroup for order 12 (idempotent) and for order 10 (without assumption of idempotence) and non-existence results for order 14 and 15 for idempotent models.

[ZS94] uses propositional provers based on the Davis-Putnam algorithm [DP60], and they present new results for quasigroup existence problems that were previously not solved, some of which are given as follows where x and y are elements of the quasigroup:

- Quasigroup with property $((x * y) * x) * x = y$ for order 13 and 14; and non-existence for order 15.
- Quasigroup with property $(x * y) * (y * x) = y$ for order 12.
- Quasigroup with property $(x * y) * y = x * (x * y)$ for order 15.

Furthermore in [ZH94], Zhang and Hsiang show how propositional reasoning can be used to solve open problems in quasigroups. They employ the cyclic group construction technique previously used by [BZ92, Hor74, HS74]. The main idea is to generate an incomplete quasigroup using an Abelian group of order $v - n$ from its first row and from the last n elements of the first column. While the technique is incomplete, it reduces the search space significantly which makes the job of the SAT solver easier, compared to working from scratch.

[SZ95] shows how quasigroup identities can be studied by rewriting techniques. Quasigroups satisfying some constraints that take the form of equations are known as quasigroup identities. Their study identifies two classes of problems for which rewrite techniques can help. The first is concerned with finding the identities of certain types of quasigroups which are conjugate-equivalent to some given identities, and the second decides which identities imply one of the constraints conjugate-equivalent

to some given identities. For example, they show that the quasigroup identity $(x * (x * y)) * y = x$ is a conjugate-implicant of the quasigroup identity $x * (x * y) = y * x$.

[CM01] introduces an approach for finding a single solution to quasigroup constraint satisfaction problems (CSPs) that uses a combination of different techniques, namely constraint solving, machine learning and automated theorem proving. The core of the approach is to automatically generate implied, symmetry breaking and specialization constraints via machine learning and automated theorem proving. These constraints are then used along with the constraints for the basic model of the quasigroup to find solutions for larger instances using a constraint solver. Constraint Solver Choco [Lab00] is used for finding small size examples of quasigroups which are given to the HR [Col02a] theory formation system that invents new concepts, finds conjectures and proves them using the automated theorem prover Otter [McC03b] in order to find implied constraints (implication theorems) and induced theorems that are based on the theory around the examples supplied by Choco. The resulting constraints are interpreted to reformulate the basic CSP model to look for solutions to the specialised CSP. However, the approach is semi-automated and requires expertise in constraint modelling and pure mathematics to interpret the output from HR as constraints and make translations to the input understandable by the constraint solver. The method is further fully automated in [CCM06] where a system is demonstrated for automatically reformulating CSP solver models by combining the capabilities of machine learning and automated theorem proving with CSP systems. Furthermore, the procedure is applied to new finite algebras namely groups, Moufang loops and rings. The system is given a basic CSP formulation and

outputs a set of reformulations, each of which includes additional constraints. Here, one issue comes to mind regarding the time taken in the reformulation of problems that is recovered by reducing the search time for solutions to larger problem instances. The approach can benefit further by translating the constraint satisfaction problem to model generators and SAT solvers for performing comparisons. The results of this approach make it evident that the combination of different automated reasoning systems with machine learning is indeed more powerful and beneficial than using one system on its own.

2.2 Combinatorial Completion Problems

Following [GS02b], an incomplete or partial Latin square is defined as a partially filled table with n rows and n columns such that no symbol occurs twice in a row or a column. The quasigroup or Latin square completion problem (QCP) is the problem of determining whether the remaining entries of the table can be filled in such a way that we obtain a complete Latin square.

[GS02b] describes a structured graph colouring benchmark test suite based on the completion of Latin squares and proposes three complete methods for solving the benchmark, a CSP (constraint satisfaction problem) approach, a hybrid LP (linear programming)/CSP strategy and a SAT-based approach and concludes that none of the methods dominates the other on the benchmark. The SAT-based approach, while being effective on critically constrained instances, suffers from having large problem encodings due to the limited expressiveness of the SAT formulation. The CSP-based approach has compact problem encodings and is effective on under-constrained instances. In particular, the alldiff constraint helps in reducing the search space in under-constrained

instances but has a negative effect when the instances are critically constrained. LP rounding technique that computes variable ranks to assign and propagate variable values, boosts the CSP-based approach by providing a powerful search heuristic.

[DdVC03a] uses a pure constraint programming approach for solving quasigroup completion problems of significantly large sizes than was previously thought possible by [GS02a]. They use a number of previously known ideas such as redundant modelling proposed by [CLW96] where two models primal and dual are connected by channelling constraints that are introduced in [Wal01]. However, the novelty of their approach, that is the key to their success, is the value ordering heuristic also known as min-domain value selection heuristic. According to the heuristic the variable with the minimum domain is given priority of selection for assigning a value.

[SSW98] proposes a method to quasigroup completion problems by maintaining general arc consistency on the n-ary all different constraints using the algorithm of [Rég94]. They show that enforcing general arc consistency on the n-ary constraints is strictly stronger than enforcing arc consistency on the binary constraints [GS97], which is strictly stronger than forward checking [MW98]. The aim of arc consistency algorithms is to effectively remove as many inconsistent values from the domain of variables before the search or at an early stage of the search. A constraint is arc consistent (AC) if for any value of the variable in the constraint there exists a value for the other variable in such a way that the constraint is satisfied. CSP is arc consistent if all the constraints are arc consistent. The constraint is generalized arc consistent (GAC) if for any value of the variable in the constraint there exist values for the other variables in the constraint such that the tuple satisfies the constraint.

Consider the following example of quasigroup completion problem where Q is a quasigroup of size 3 with elements $0, 1, 2$. The multiplication table of quasigroup Q is presented below where the values given in each cell represent the domain of each cell. The example gives an insight on the efficiency of generalized arc consistency in comparison to arc consistency on binary constraints. Both techniques are used for pruning the domain of each cell of the multiplication table of the quasigroup Q in order to reduce the search space for the quasigroup completion problem. The resulting pruned multiplication tables after application of each technique on the example quasigroup multiplication table are given respectively.

Q	0	1	2
0	$\{0\}$	$\{0,1,2\}$	$\{0,1,2\}$
1	$\{0,1,2\}$	$\{0\}$	$\{0,1,2\}$
2	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$

Enforcing arc consistency on the binary constraints in the above example results in:

Q	0	1	2
0	$\{0\}$	$\{1,2\}$	$\{1,2\}$
1	$\{1,2\}$	$\{0\}$	$\{1,2\}$
2	$\{1,2\}$	$\{1,2\}$	$\{0,1,2\}$

Enforcing general arc consistency on all different constraints filters out two more values in the bottom right cell resulting in the following:

Q	0	1	2
0	$\{0\}$	$\{1,2\}$	$\{1,2\}$
1	$\{1,2\}$	$\{0\}$	$\{1,2\}$
2	$\{1,2\}$	$\{1,2\}$	$\{0\}$

Identifying and breaking symmetries is important in reducing the search space in combinatorial problems where we are interested in finding all

solutions for a given problem or want to claim the non-existence of a solution. It has also been shown experimentally in [RM05] that breaking partial symmetries is also beneficial when there is a need for finding one solution. Local symmetries [BS07] also known as conditional symmetries [GKL+05] can be applied to a combinatorial problem instance where there is a partial assignment. These local symmetries are broken by using additional clauses to the original encoding of the problem. [ML07] performs an experimental study using a SAT solver on breaking local symmetries in quasigroup completion problems by computing additional clauses that break the symmetry of the partial element assignments of the problem. Although this helps in reducing the number of solutions, the additional clauses for breaking symmetries deteriorate the performance of the SAT solver, which is due to not only the overhead of dealing with additional clauses but mainly because of the heuristics used by the SAT solvers that do not benefit from these clauses.

2.3 Enumeration of Algebraic Structures

In terms of combinatorics, it is a well known problem to find out how many solution instances exist for a certain class of algebras of a particular size. [DK09, DSS11, DJKK12] show how computer algebra can be used to break symmetries in constraint satisfaction search to find solutions for the enumeration of algebras to a class of problems presenting new results in algebraic combinatorics. They not only provide enumeration results but also store each solution for the algebraic structure to be analysed by algebraists. Their second aim is to generate and store a canonical example from each equivalence class of solutions. This involves breaking the symmetries that allow objects from the same class to be interchanged.

[DK09] combines group-theoretic GAP [GAP08] calculations with the speed and efficiency of the Minion [GJM06] constraint solver to obtain the numbers of monoids up to size 10. They present new results up to isomorphism and anti-isomorphism by showing that there are 858,977 monoids of size eight; 1,844,075,697 monoids of size nine and 52,991,253,973,742 monoids of size ten.

[DJKK12] describes the use of mathematical results combined with distributed constraint satisfaction to obtain and show that the number of non-equivalent semigroups of size 10 is 12,418,001,077,381,302,684 which was a previously open problem in mathematics. They partition and distribute the constraint satisfaction problem specification such that the different partitions of the search space are solved independently where the computing nodes do not require to communicate with each other. They have made advances in both constraint satisfaction and abstract algebra to compute semigroups of size 10, moreover the combination of both the constraint satisfaction technology and mathematics played a vital role in the computations.

Furthermore, [DSS11] presents the enumeration and partial classification of AG-groupoids. Their approach builds on the work done in [DK09, DJKK12] for generating monoids and semigroups respectively, and they introduce a novel adaptation to deal with a different domain i.e. AG-groupoids. Furthermore, they go beyond simple enumeration of the structures by the constraint solver and obtain further division of the domain into interesting subclasses of AG-groupoids. They use GAP for two purposes: firstly to perform symmetry breaking during the constraint solving process and secondly to perform the subsequent subclassification. They also produce multiplication tables of the structures under consideration which can be further used to produce more fine-grained

subclassification as demonstrated in the case of AG-groupoids via a two step approach where they first separate the structures with respect to associativity and commutativity properties and then as a second step perform refinement using more specialised properties.

[SA08b, SA08a] present the generation and enumeration of loops with the inverse property (IP). The enumeration of non-isomorphic IP loops with order up to 13 and commutative IP loops of order 14 is performed by using a finite domain constraint solver Finder [Sla94] to generate representatives of all isomorphism classes. Finder works by expressing each equation or a defining condition as the set of its ground instances on the domain of N elements. It then compiles them into constraints and conducts a backtracking search for solutions to the constraint satisfaction problem using standard techniques such as forward checking and no-good learning that are described in [Dec03].

[CP05] describe the Theorem Modifier (TM) system which is an automated theorem modification system based on an implementation of the methods prescribed in Lakatos's philosophy of mathematics, and relies on the interaction of HR [Col02a], Otter [McC03b] and Mace [McC03a] programs. The effectiveness of TM system is demonstrated in tests, where TM was able to modify 7 out of 9 non-theorems from the TPTP library [SS] into interesting, proved alternatives. Furthermore, on an artificial set of 98 non-theorems, it produced meaningful modifications 80% of the times.

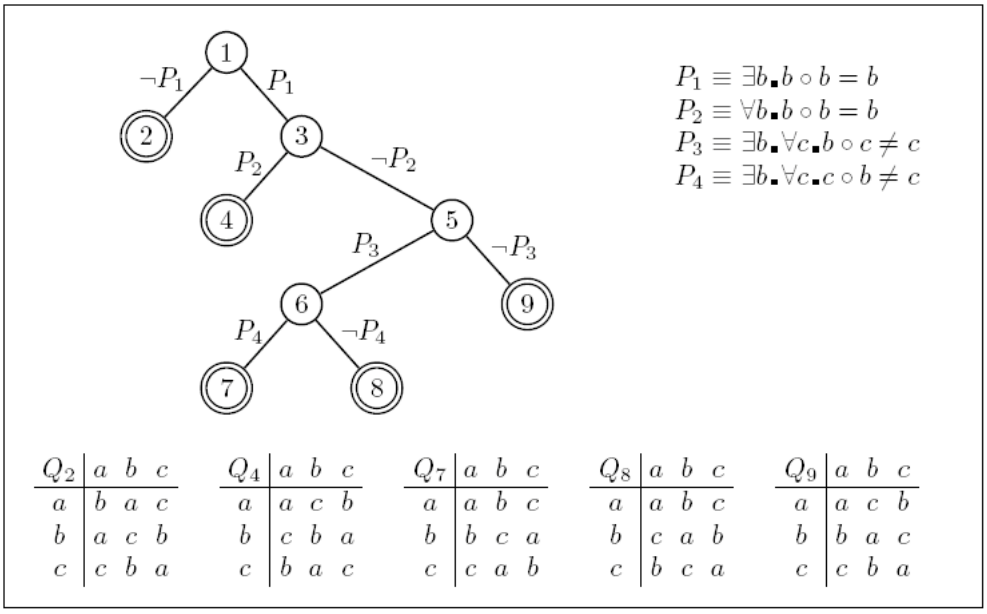
2.4 Qualitative Classification of Algebraic Structures

The qualitative classification of finite algebraic structures not only describes the number of equivalence classes but also specifies the discriminating properties that tell us how the classes differ from each other. This is particularly useful in allowing one to use properties of relatively small structures to help in the classification of larger structures. [CMSM04] presents a semi-automated as well as a fully automated bootstrapping approach to building and verifying classification theorems that classify algebras of a particular type and size into isomorphism classes. The Mace [McC03a] model generator is used to generate representatives of each isomorphism class for the algebra of a particular size, which is then followed by using HR [Col02b] and C4.5 [Qui93] machine learning systems to induce a set of classifying properties. Furthermore, the classification is verified by constructing appropriate verification problems that are simplified using GAP [GAP08] and then proved with the Spass [WBH+99] theorem prover. Moreover, the approach is fully automated using a bootstrapping procedure to build a decision tree that decides the isomorphism class of a given algebra. Each step of the decision tree is verified by first using GAP to simplify the verification problems and then Spass for proving them. Both semi and fully automated approaches successfully generate a number of classification theorems for groups, monoids, quasigroups and loops up to size 6. The approaches present a novel method of using the combination of multiple reasoning systems to tackle difficult classification problems and provide a general method that can be applied to any algebraic domain and equivalence relation.

[SMMC08] extends the study of [CMSM04] to the production of classification theorems up to isotopism. Isotopism is an important generalization of isomorphism and is studied in the domain of algebraic loop theory. Finding isotopy invariants is a more complex task and the machine learning approach did not suffice for this application. Three novel techniques for generating isotopy invariants were developed that use the notion of universal identities via an interplay of model generation and theorem proving, and constructions based on sub-blocks that use computer algebra techniques. The proof problems concerning a conjunction of the invariants forming an isotopy class were simplified with computer algebra techniques and the final proof was found using a satisfiability solver. The bootstrapping algorithm was employed to generate new results within an isotopic classification theorem for loops of size 6 providing a full set of classifying properties, and a summary of similar result for loops of size 7 is also presented. While the verifications of some of the classification theorems pose little difficulty for automated theorem provers, it was found that the verifications of the other classification theorems were beyond the capabilities of state of the art provers. Therefore, since the problems are in a finite domain, Boolean satisfiability was employed. [MS05] presents the application of satisfiability solvers to generate fully verified classification theorems in finite algebra exploring diverse methods to efficiently encode the problems both for Boolean SAT solvers as well as for solvers with built-in equational theory. This lead to an improvement of the overall bootstrapping algorithm.

Finally, [SCMM08] presents the bootstrapping approach that incorporates a set of diverse reasoning techniques, including first order resolution theorem proving, model generation, satisfiability solving and

computer algebra methods, and is successfully applied to produce a number of novel classification theorems for loops and quasigroups with respect to isomorphism and isotopism, in particular for quasigroups up to size 5 and loops up to size 7. Figure 2.1 shows the decision tree for the classification problem of order 3 quasigroups alongwith the five isomorphism class represents. The leaf nodes $2, 4, 7, 8$ and 9 of the tree are the isomorphism classes with the respective represents Q_2, Q_4, Q_7, Q_8 and Q_9. The discriminating properties are labelled on the edges of the tree. The conjunction of these discriminating properties given on the path from the leaf to the root uniquely determine the isomorphism class represented by the leaf node. The full classification theorem corresponds to a disjunction of conjunctions of the discriminating properties.



Q_2	a	b	c
a	b	a	c
b	a	c	b
c	c	b	a

Q_4	a	b	c
a	a	c	b
b	c	b	a
c	b	a	c

Q_7	a	b	c
a	a	b	c
b	b	c	a
c	c	a	b

Q_8	a	b	c
a	a	b	c
b	c	a	b
c	b	c	a

Q_9	a	b	c
a	a	c	b
b	b	a	c
c	c	b	a

Figure 2.1: Decision tree for the classification problem of order 3 quasigroups [SCMM08]

2.5 Concluding Remarks

In this chapter we have presented a survey of how automated reasoning systems have been used to solve mathematical problems concerning finite

algebras. It is evident that the combination of different reasoning systems is very beneficial and helps to solve problems that a single system is unable to solve on its own. A point worth noting is that the capability of these reasoning systems can be enhanced by exploring mathematical domain knowledge. We demonstrate this in Chapter 7, by presenting a description of a model that combines various reasoning systems to automatically explore algebraic theory to enable the generation of large size solution models. It integrates a mix of bespoke algorithms we have implemented and off the shelf reasoning tools. Furthermore, in Chapter 11 we present a novel approach of combining systems where automated system components and user interaction are integrated to mutually support each other in developing solutions to the problems. Efficient encoding for the reasoning systems is necessary and there is a need for modelling systems that can generate the inputs for these systems so that a user does not require expertise to use them. Moreover, translations are also necessary so that if one system is unable to solve a problem, the other systems can be used. In Chapter 7, we present a comparative analysis of different model generation systems that are based on diverse reasoning techniques to compute large size models of quasigroups with non-trivial properties.

CHAPTER 3

LOGIC AND AUTOMATED REASONING

Chapter Overview: *In this research work, different reasoning systems have been used which employ logical formalisms for the inputs and outputs i.e. the communication between these systems is in logic. This chapter first describes the logical systems which include propositional logic, first order logic and equational logic. Furthermore, the various automated reasoning techniques that are used in this research work are described, in particular the input formulations and solution models with statistics are shown for the systems that were used in this work.*

This chapter provides background information on logical systems and automated reasoning techniques. We begin this chapter with a brief introduction to logical systems and explain some of the terminology which appears later. We define what we mean by automated reasoning. It is a large area of artificial intelligence, encompassing many disciplines which are suited to solving particular types of problems and we talk

about some relevant techniques that were used in the research. We limit our discussion of the details of how systems operate to only those systems which were used extensively in this work.

3.1 Logical Systems

As defined in [Fit90b], logic is a formal system in which the formulae are interpreted to either false or true. Every logic has the following components:

- Syntax: This specifies the symbols in the language and how they can be combined to form sentences.
- Semantics: This specifies what facts in the world a sentence refers to. A fact is a claim that may be true or false.
- Inference Procedures: Mechanical methods for computing or deriving new sentences which follow from existing sentences.

3.1.1 Propositional Logic

Propositional logic is a simple language that is useful for showing key ideas and definitions. The basic terms that form the main parts of propositional logic are defined as follows:

- A set of propositional symbols such as P and Q and their semantics are defined by the user.
- A sentence (also called a formula or well-formed formula or wff) is defined as:
  1. A symbol.
  2. If S is a sentence, then $\sim S$ is a sentence, where " $\sim$ " is the "not" logical operator.

3. If S and T are sentences, then $(S \vee T), (S \wedge T), (S \rightarrow T)$, and $(S \Leftrightarrow T)$ are sentences, where the four logical connectives correspond to "or," "and," "implies," and "if and only if," respectively.

4. A finite number of applications of $1 - 3$.

- Given the truth values of all of the constituent symbols in a sentence, that sentence can be "evaluated" to determine its truth value (True or False). This is called an interpretation of the sentence.

- A model is an interpretation (i.e., an assignment of truth values to symbols) of a set of sentences such that each sentence is True.

- A valid sentence (also known as a tautology) is a sentence that is True under all interpretations.

- An inconsistent sentence (also called unsatisfiable or a contradiction) is a sentence that is False under all interpretations.

- Sentence P entails sentence Q, written $P \models Q$, means that whenever P is True, so is Q. In other words, all models of P are also models of Q.

  The inference rules of propositional logic allow us to derive new logical formulae from formulae that are taken to be true. Some inference rules are shown in Table 3.1.

3.1.2 First Order Logic

Following [Fit90b] the basic terminology used in first-order logic (also known as predicate logic) is defined as follows:

Inference Rule	Given	Result
Modus Ponens	$A, A \Rightarrow B$	B
And Introduction	A, B	$A \wedge B$
And Elimination	$A \wedge B$	A
Double Negation	$\sim\sim A$	A
Unit Resolution	$A \vee B, \sim B$	A
Resolution	$A \vee B, \sim B \vee C$	$A \vee C$

Table 3.1: Selection of propositional logic inference rules

The following primitives are defined by the user:

- Constant symbols (i.e., the "individuals" in the world) e.g., Mary, 3.
- Function symbols (mapping individuals to individuals) e.g., father-of(Mary) = John, color-of(Sky) = Blue.
- Predicate symbols (mapping individuals to truth values) e.g., greater(5,3), green(Grass), color(Grass, Green).

The following symbols are supplied by first-order logic:

- Variable symbols. e.g., x, y.
- Connectives. They are the same as used in propositional logic : not ($\sim$), and ($\wedge$), or ($\vee$), implies ($\rightarrow$), if and only if ($\Leftrightarrow$).
- Quantifiers: Universal ($\forall$) and Existential ($\exists$)
  - Universal quantification corresponds to conjunction ("and") in that $\forall\, x\, P(x)$ is equivalent to the conjunction $P(x_1) \wedge P(x_2) \wedge P(x_3) \wedge ... \wedge P(x_n)$ which means that P holds for all values of x in the domain associated with that variable. E.g., $\forall\, x\, dog(x) \rightarrow mammal(x)$.
  - Existential quantification corresponds to disjunction ("or") in that $\exists\, x\, P(x)$ is equivalent to the disjunction

$P(x_1) \vee P(x_2) \vee P(x_3) \vee ... \vee P(x_n)$ which means that P holds for some value of x in the domain associated with that variable. E.g., $\exists\, x\, mammal(x) \wedge lays\text{-}eggs(x)$.

- Universal quantifiers are usually used with "implies" to form "if-then rules." E.g., $\forall\, x\, (phdstudent(x) \rightarrow smart(x))$ means "All phd students are smart".
- Existential quantifiers are usually used with "and" to specify a list of properties or facts about an individual. E.g., $\exists\, x\, (phdstudent(x) \wedge smart(x))$ means "there is a phd student who is smart".
- Switching the order of universal quantifiers does not change the meaning: $\forall\, x\, \forall\, y\, P(x, y)$ is logically equivalent to $\forall\, y\, \forall\, x\, P(x, y)$. Similarly, you can switch the order of existential quantifiers.
- Switching the order of universals and existentials does change meaning:
  - Everyone likes someone: $\forall\, x\, \exists\, y\, likes(x, y)$.
  - Someone is liked by everyone: $\exists\, y\, \forall\, x\, likes(x, y)$.

Sentences are built up from terms and atoms:

- A term (denoting a real-world individual) is a constant symbol, a variable symbol, or an n-place function symbol applied to n terms. For example, x and $f(x_1, ..., x_n)$ are terms, where each x_i is a term.
- An atom (which has value true or false) is either an n-place predicate symbol applied to n terms. Formulae are atoms, or if P and Q are formulae, then $\sim P$, $P \vee Q$, $P \wedge Q$, $P \rightarrow Q, P \Leftrightarrow Q$ are formulae. If P is a formula and x is a variable, then $\forall\, x\, P$ and $\exists\, x\, P$ are formulae.

Inference Rule	Given	Result
Forall introduction	$P(c)$ true for all possible c	$\forall\, x\, P(x)$
Forall elimination	$\forall\, x\, P(x)$	$P(c)$
Exists introduction	$P(c)$	$\exists\, x\, P(x)$
Exists elimination	$\exists\, x\, P(x)$	$P(c)$ for some arbitrary c

Table 3.2: First-order logic inference rules for quantifiers

- A sentence is a well-formed formula (wff) containing no "free" variables. i.e., all variables are "bound" by universal or existential quantifiers. E.g., $\forall\, x\, P(x, y)$ has x bound as a universally quantified variable, but y is free, hence this is not a sentence.

A statement in first order logic can be represented as *clauses*. A *clause* is a disjunction of literals e.g. $A_1 \vee A_2 \vee ... \vee A_n$. A literal is a predicate or its negation. Furthermore, *Conjunctive Normal Form* describes a propositional formula which is a conjunction of clauses. For example the following statement is in conjunctive normal form: $(A \vee \sim B) \wedge (B \vee C)$. The inference rules for first order logic are similar to those of propositional logic. Apart from them, there are some additional inference rules affecting quantifiers which are shown in Table 3.2.

3.1.3 Equational Logic

Equational logic as defined in [Pig75] is a fragment of first-order predicate logic with equality in which universally quantified equations are the only formulas. In other words, equational logic consists of a set of function symbols of fixed arity, variables and constant symbols. Formulas are in the form of equations where all variables are universally quantified. Equational logic plays an important role in defining some classes of algebras. Most algebraic structures that are of interest to algebraists can be axiomatically defined by identities written in equational logic. As an

Inference Rule	$\frac{PREMISE}{CONLUSION}$
[1] Reflexivity	$\frac{}{t=t}$
[2] Symmetry	$\frac{t=t'}{t'=t}$
[3] Transitivity	$\frac{t=t', t'=t''}{t=t''}$
[4] Instantiation	$\frac{t=t'}{t_p=t'_p}$ for every substitution p
[5] Substitution	$\frac{t_1=t'_1,...,t_k=t'_k}{f(t_1,...,t_k)=f(t'_1,...,t'_k)}$ for all n-ary function symbols f of arity k

Table 3.3: Equational logic inference rules

example, the class of semigroups can be defined by the associative identity $x * (y * z) = (x * y) * z$.

Equational logic provides a deductive proof system that enables the generation of more equations from a set of original equations E. Equations are written in the infix form '=' or sometimes expressed as a pair of terms $\langle t, t' \rangle$ where the terms are equal. By applying a set of inference rules on the original equations E we can generate new equations that are known as theorems of the logic. The inference rules for equational logic are given in Table 3.3. The first three rules $1, 2, 3$ capture the properties of an equivalence relation (reflexivity, symmetry, and transitivity). Rule 4 states that equational logic is closed under substitutions as defined in Table 3.3, i.e., if we take an equation from E and apply the same substitution on both sides the new equation is also a consequence of E. Finally, rule 5 means that equational logic is also closed under all n-ary function symbols f , i.e., if $t_1 = t'_1, ..., t_k = t'_k$ are provable from E then $f(t_1, ..., t_k) = f(t'_1, ..., t'_k)$ is also provable from E.

3.2 Automated Reasoning

Automated reasoning refers to reasoning done by a computer using logic. A system that performs automated reasoning uses some form of logical representation and can provide some new information given some background information based on logical reasoning. Logical reasoning depicts the methods of using logical formalizations in order to derive conclusions from the preconditions according to the inference rules given in the formalization.

Some fields of automated reasoning which we use in our research work are described as follows.

3.2.1 Automated Theorem Proving

Automated Theorem Proving (ATP), see for instance [Sut], deals with the development of computer programs that show that some statement (conjecture) is a logical consequence of a set of statements (axioms and hypotheses). The input information is a set of axioms together with the theorem to be proved specified in a particular formal logic, and the output is a formal proof that the theorem follows from the axioms via the inference rules of the formal logic.

ATP has many applications and it can be used in a variety of domains such as mathematics, program analysis and system verification. The language in which the conjecture, hypotheses and axioms are written is a formal logic. This means that a precise, clear and accurate formal statement of the problem is given to the ATP system and there is no form of ambiguity, in contrast to natural languages such as English. The proofs produced by ATP describe how and why the conjectures follow from the axioms and hypotheses. The proofs are in a form that can then

be understood by an expert or a computer program.

There are many powerful ATP systems available to use. Examples of first-order ATP systems include Otter [McC03b], Prover9 [McC], E [Sch02], SPASS [WBH+99], Vampire [RV01] and Waldmeister [BHF96].

We have used Prover9 in our research experiments. Prover9 is a successor of the Otter prover. Prover9 accepts input in the form of first-order and equational logic. Figure 3.1 shows an example input file for Prover9 using quantifiers.

```
formulas(assumptions).
all x all y (subset(x,y) <-> (all z (member(z,x) -> member(z,y)))).
end_of_list.
formulas(goals).
all x all y all z (subset(x,y) & subset(y,z) -> subset(x,z)).
end_of_list.
```

Figure 3.1: Prover9 input file example

The primary mode of inference used by Prover9 is resolution.

Gallier [Gal85] describes the idea of resolution as: "The essence of the method is to prove the validity of a proposition by establishing that the negation of this proposition is unsatisfiable", which means to prove P, the method attempts to disprove 'not P' ($\neg P$). Resolution provides a complete proof procedure for detecting inconsistency of formulae that are expressed in first order logic. Resolution procedure uses a single rule of inference: the Resolution Rule (RR), which is a generalization of the same rule used in propositional logic defined in Section 3.1.1 in Table 3.1.

To prove that a sentence p can be derived from a set of sentences KB, resolution procedure uses the following steps:

(i) Convert $\neg p$ and the sentences in KB to conjunctive normal form.

(ii) Repeat the following steps until either a contradiction is found, no progress can be made or a pre-determined amount of effort has been expended

- Find two clauses that contain a literal in one clause and the negation of the literal in the other clause, for example of the form $u \vee v_1 \vee v_2 ... \vee v_k$ and $\neg u \vee w_1 \vee w_2 ... \vee w_l$
- Combine the two clauses using the resolution rule of inference, adding the resolvent(s) to the set of sentences KB. For example resolving $u \vee v_1 \vee v_2 ... \vee v_k$ and $\neg u \vee w_1 \vee w_2 ... \vee w_l$ gives the resolvent clause: $v_1 \vee v_2 ... \vee v_k \vee w_1 \vee w_2 ... \vee w_l$.
- If one of the resolvents is the empty clause, then a contradiction has been found. Return "p has been proven".

Conjunctive normal form (CNF) is also called the clausal form. Every sentence in CNF is a conjunction of disjunctions of literals. To convert a first order logic sentence to CNF following steps should be followed:

(i) Remove implications

- Replace $P \rightarrow Q$ by $\neg P \vee Q$
- Replace $P \leftrightarrow Q$ by $(\neg P \vee Q) \wedge (P \vee \neg Q)$

(ii) Move negation inwards

- $\neg \forall\, x\ P$ becomes $\exists\, x\ \neg P$
- $\neg \exists\, x\ P$ becomes $\forall\, x\ \neg P$
- $\neg \neg P$ becomes P
- $\neg (P \wedge Q)$ is replaced by $\neg P \vee \neg Q$
- $\neg (P \vee Q)$ is replaced by $\neg P \wedge \neg Q$

(iii) Standardize variables

- each quantifier gets unique variables, for example
  $\exists\, x\, P(x)\ \wedge\ \exists\, x\, Q(x)$ becomes $\exists\, x\, P(x)\ \wedge\ \exists\, y\, Q(y)$

(iv) Move quantifiers to the left

- $\forall\, x\, P\ \vee\ \exists\, y\, Q$ becomes $\forall\, x\, \exists\, y\, (P\ \vee\ Q)$

(v) Eliminate $\exists$ by Skolemization

- $\exists\, x\, P(x)$ becomes $P(A)$
- $\forall\, x\, \forall\, y\, \exists\, z\, P(x, y, z)$ becomes $\forall\, x\, \forall\, y\, P(x, y, F(x, y))$
- $\forall\, x\, \exists\, y\, Pred(x, y)$ becomes $\forall\, x\, Pred(x, Succ(x))$

(vi) Drop universal quantifiers

(vii) Distribute And over Or

- $(P \wedge Q) \vee R$ becomes $(P \vee R) \wedge (Q \vee R)$

In propositional logic, it is easy to determine that two literals contradict each other by simply looking for p and $\neg p$. However, in first order logic this matching process is more complicated because arguments of predicates must be considered. For example, $man(John)$ and $\neg man(John)$ is a contradiction, while $man(John)$ and $\neg man(Tom)$ is not. To detect contradictions in first order logic, a matching procedure is required that compares two literals and discovers whether there exists a set of substitutions *theta* that makes them identical. This procedure is called unification which works by taking two atomic sentences (literals), such as $Knows(John, x)$ and $Knows(John, Paul)$, and return a substitution *theta* that makes them look the same, such as $\{x/Paul\}$. Algorithm 1 defines the unify procedure that takes two literals p, q and

empty substitution list *theta* as input and returns "failure" if the two input literals do not match and a substitution list, *theta*, if they do match.

Algorithm 1 Unification

procedure unify(p**,** q**,** *theta***)**
Scan p and q left-to-right and find the first corresponding terms where p and q "disagree" ; where p and q not equal
if there is no disagreement **then**
  return *theta*
end if
Let r and s be the terms in p and q, respectively, where disagreement first occurs
if variable(r) **then**
  theta = union(theta, $\{r/s\}$)
  unify(subst(*theta*, p), subst(*theta*, q), *theta*)
else if variable(s) **then**
  theta = union(*theta*, $\{s/r\}$)
  unify(subst(*theta*, p), subst(*theta*, q), *theta*)
else
  return "failure"
end if
end

The aim of Prover9 is to detect inconsistency by deriving a contradiction, and for that it makes use of repeated resolution inferences. Prover9 uses the following procedure:

i Preprocess the input file to convert it into the form appropriate for inferencing.

ii Negate the formula given as a goal.

iii Translate all formulae into clausal form.

iv Compute inferences and by default write these in standard output.

v If an inconsistency is detected then stop and print out a proof consisting of the sequence of resolution rules that generated the inconsistency. Print out various statistics associated with the proof.

The proof generated by Prover9 for the example input file given in Figure 3.1 is shown in Figure 3.2

```
============================== PROOF =================================

% Proof 1 at 0.00 (+ 0.00) seconds.
% Length of proof is 14.
% Level of proof is 4.
% Maximum clause weight is 6.
% Given clauses 6.

1 (all x all y (subset(x,y) <-> (all z (member(z,x) -> member(z,y))))) # label(non_clause).
[assumption].
2 (all x all y all z (subset(x,y) & subset(y,z) -> subset(x,z))) # label(non_clause) # label(goal).
[goal].
3 subset(x,y) | member(f1(x,y),x).  [clausify(1)].
4 -subset(x,y) | -member(z,x) | member(z,y).  [clausify(1)].
5 subset(x,y) | -member(f1(x,y),y).  [clausify(1)].
6 subset(c1,c2).  [deny(2)].
7 subset(c2,c3).  [deny(2)].
8 -subset(c1,c3).  [deny(2)].
11 -member(x,c1) | member(x,c2).  [resolve(6,a,4,a)].
12 -member(x,c2) | member(x,c3).  [resolve(7,a,4,a)].
13 member(f1(c1,c3),c1).  [resolve(8,a,3,a)].
14 -member(f1(c1,c3),c3).  [resolve(8,a,5,a)].
15 member(f1(c1,c3),c2).  [resolve(13,a,11,a)].
18 $F.  [ur(12,b,14,a),unit_del(a,15)].

============================== end of proof ==========================
```

Figure 3.2: Prover9 proof example

3.2.2 Term Rewriting Systems

Term rewriting is based on equational logic employing the repeated application of directed equations also known as rewrite rules or substitution rules, unlike equational logic where the equations have no direction. This makes term rewriting well suited for symbolic computations, program analysis and program transformation.
Following [Klo87, HO80, Ter03], a term rewriting system is defined as follows:

Definition 3.2.1 *A Term Rewriting System (TRS) is defined as a pair* (Σ, R)*, where* Σ *denotes the alphabet or signature and* R *is a set of reduction rules (directed equations) also known as rewrite rules. The alphabet* Σ *consists of:*

- V*, a countably infinite set of variables* $x, y, z, ...$

- *F, a non-empty set of function symbols or operator symbols $f, g, ...$ applied to zero or more arguments that define the 'arity' of the function symbols. A 0-ary function symbol is called a constant.*

Definition 3.2.2 *The set of terms $T(\Sigma)$ over the alphabet Σ is inductively defined as follows:*

- *$x \in T(\Sigma)$ where $x \in V$.*
- *$f(M_1, ..., M_n) \in T(\Sigma)$ where $f \in F$ is an n-ary function symbol and $M_1, ...M_n \in T(\Sigma)$ $(n \geq 0)$.*

If no variables occur in a term, then it is called a *ground* term and $T_0(\Sigma)$ denotes the set of ground terms. Terms in which every variable occurs only once are called *linear*. s is a *subterm* of term t if s is a term that occurs somewhere in t.

Definition 3.2.3 *A substitution Θ is a mapping from $T(\Sigma)$ to $T(\Sigma)$ such that:*

- *$\Theta(f(M_1, ..., M_n)) \equiv f(\Theta(M_1), ..., \Theta(M_n)))$ where $f(M_1, ..., M_n) \in T(\Sigma)$ and $n \geq 0$.*

So, Θ is determined by its restriction to the variables.

Definition 3.2.4 *A reduction rule (or rewrite rule) is a pair (t, s) of terms $\in T(\Sigma)$, written as $r : t \rhd s$ where r is the name given to the reduction rule, having two conditions:*

1. *The left hand side (LHS) t is not a variable.*
2. *The variables in the right hand side (RHS) s are already contained in t.*

A reduction rule $r : t \rhd s$ determines a set of rewrites $\Theta(t) \rhd_r \Theta(s)$ for all substitutions Θ.

The following are a few examples based on term rewriting.

Example 3.2.1 *Consider the following rewrite rules:*

- $r_1 : A(x, S(y)) \rhd S(A(x, y))$
- $r_2 : A(x, 0) \rhd x$

Now, $A(0, S(A(S(0), 0)))$ can be simplified using the above rewrite rules as follows:

- $A(0, S(A(S(0), 0)))$

$\rhd$ $S(A(0, A(S(0), 0)))$

$\rhd$ $S(A(0, S(0)))$

$\rhd$ $S(S(A(0, 0)))$

$\rhd$ $S(S(0))$

Example 3.2.2 *Consider the following rewrite rule:*

- $f(g(x)) \rhd g(f(x))$

Now, $f(f(g(f(g(x)))))$ can be simplified using the above rewrite rule as follows:

- $f(f(g(f(g(x)))))$

$\rhd$ $f(f(g(g(f(x)))))$

$\rhd$ $f(g(f(g(f(x)))))$

$\rhd$ $g(f(f(g(f(x)))))$

$\triangleright$ $g(f(g(f(f(x)))))$

$\triangleright$ $g(g(f(f(f(x)))))$

As demonstrated in the above examples and described in [HKK91], rewriting a term consists of replacing a subterm, which matches a left hand side of a rewrite rule, by the right hand side, where variables have acquired the value determined by matching. Iterating this process using a rewrite system R is called reducing or rewriting. If two terms can be rewritten to the same one, a special equational proof is obtained, called a *rewrite proof*. A term which cannot be rewritten is said to be in *normal form*. As given in [Gog98], a TRS is said to be *terminating* or *Noetherian*, if each term has a normal form i.e. there are no infinite sequences of rewrites using it.

There are many tools that employ term rewriting. CoLoR [BK11] is a Coq [HKPM04] library of mathematical definitions and theorems on the termination of rewrite relations. Coq [HKPM04] is a proof assistant based on a higher-order logic allowing powerful definitions of functions. RRL (Rewrite Rule Laboratory) [KZ95] is a rewrite-rule based theorem prover for equational and inductive reasoning. Stratego [BKVV08] is a modular language for the specification of fully automatic program transformation systems based on the paradigm of rewriting strategies. Watson [HAF01] is an interactive equational theorem prover, where theorems are expressed as rewrite rules. It has a programming language where programs are systems of recursively chained rewrite rules, proved and stored in the same way as theorems. We take inspirations from these systems but as described in Chapter 10, we build our own term rewriting system.

3.2.2.1 Knuth-Bendix Completion

The Knuth-Bendix completion algorithm attempts to transform a finite set of equations (over terms) into a finitely terminating, confluent term rewriting system. This term rewriting system serves as a decision procedure for validating the word problem i.e. whether two given terms represent the same element? The word problem is undecidable so the algorithm is not guaranteed to terminate. If the algorithm succeeds it has effectively solved the world problem.

Initially, the completion algorithm attempts to orient input equations according to the reduction order (if $s < t$, then $t \rightarrow s$ becomes a rule, where s and t are terms) Then, it completes this initial set of rules with derived ones. The algorithm iteratively detects critical pairs, obtains their normal forms, and adds a new rule for every pair of the normal forms in accordance with the reduction order.

The completion algorithm may:

1. Terminate with success and yield a finitely terminating, confluent set of rules,
2. Terminate with failure, or
3. Loop without terminating.

3.2.3 Constraint Solvers

Constraint solving is used for solving a constraint satisfaction problem (CSP) where the solution is modelled by a set of constraints on a set of decision variables. A constraint solver then assigns values to each of the variables so that all the constraints are satisfied. In addition to that, a user can specify a function which can be used by the solver to favour a particular solution from a set of many possible solutions. Constraint

solving can be used for tackling a wide variety of combinatorial problems in fields such as scheduling, industrial design, and combinatorial mathematics.

In the case of finite algebras, a constraint solver can be used to find examples of an algebra by encoding the operator table as variables and posting constraints to represent axioms. For example, quasigroups can be found by considering a table of n^2 variables with possible values of 0 to $n-1$ and constraining the variables of each row and column to be all different.

Examples of constraint solvers include Minion [GJM06], Choco [Lab00], Mistral [Heb08] and Abscon [MLB01]. They all differ in terms of the implementation and the syntax for declaring constraints.

We have used the Minion constraint solver in our experiments for generating large size examples of algebraic structures such as quasigroups. Minion [GJM06] is a general-purpose constraint solver, with an expressive input language based on the common constraint modelling device of matrix models. Therefore, it is well suited for our domain of experimentation i.e. quasigroups. The constraint satisfaction problem formulations employ one or more matrices of decision variables, with constraints typically imposed on the rows, columns and planes of the matrices. The input language of Minion has four variable types.

1. 0/1 *variables:* which are used very commonly for logical expressions, and for the characteristic functions of sets.

2. *Bounds variables:* where only the upper and lower bounds of the domain are assigned values.

3. *Sparse Bounds variables:* where the domain is composed of discrete values, e.g. $\{1, 5, 36, 92\}$, but only the upper and lower bounds of

the domain are updated during search.

4. *Discrete variables:* where the domain ranges from the lower to upper bounds specified, but the deletion of any domain element in this range is permitted.

The input language of Minion supports the definition of one, two, and three-dimensional matrices of decision variables. Furthermore, it provides direct access to matrix rows and columns since most matrix models impose constraints on them. The following are a few example constraints which are allowed:

(i) *alldiff:* forces the input vector of variables to take distinct values.

(ii) *gacalldiff:* similar to *alldiff* and additionally enforces generalized arc consistency [SSW98].

(iii) *eq:* constrains two variables to take equal values.

(iv) $abs(x, y)$*:* makes sure that $x = |y|$, i.e. x is the absolute value of y.

(v) $weightedsumgeq(constantVec, varVec, total)$*:* ensures that $constantVec \cdot varVec \geq total$, where $constantVec \cdot varVec$ is the scalar dot product of $constantVec$ and $varVec$.

(vi) $weightedsumleq(constantVec, varVec, total)$*:* ensures that $constantVec \cdot varVec \leq total$, where $constantVec \cdot varVec$ is the scalar dot product of $constantVec$ and $varVec$.

(vii) $element(vec, i, e)$*:* specifies that, in any solution, $vec[i] = e$ and i is in the range $[0 \ldots |vec| - 1]$.

(viii) $watchelement(vec, i, e)$*:* similar to $element(vec, i, e)$ and additionally enforces generalized arc consistency [SSW98].

Let us consider the combinatorial problem of N-Queens to demonstrate how Minion works. The N-Queens problem is stated as the problem of putting n chess queens on an $n \times n$ chessboard such that none of them is able to capture any other using the standard chess queen moves. The column model is used, where there is one variable of domain $1, \ldots, n$ for each row having $n = 4$. We use the *essence modelling language* [FGJ+07] to model the problem and use the translation system *tailor* [Ren] that takes the essence specification and generates the problem in the Minion input format. Figure 3.3 shows the N-Queens problem specification modelled in essence.

```
given n: int
find queens: matrix indexed by [int(1..n)] of int(1..n)
such that
forall i : int(1..n). forall j : int(i+1..n).
|queens[i] - queens[j]| != |i - j|,
alldiff(queens),
letting n be 4
```

Figure 3.3: Essence specification for N-Queens problem

The input for Minion for the N-Queens problem model is shown in Figure 3.4. 4 variables are used, each representing a column of the chess board. These 4 variables are stored in a matrix called queens with domain $\{1, ..., 4\}$ representing a 4×4 chessboard. Two auxiliary variables are used for each of the 6 diagonal constraints, one with domain $\{-3, ..., 3\}$ and the other with domain $\{0, ..., 3\}$. The variable order branches on each of the variables of the queen matrix in turn, then on the two auxiliary variables, to print only the matrix of variables.
The *alldiff* constraint is used on the queens variables. This ensures that two queens cannot be put in the same row. The rest of the constraints stop two queens from being placed on a diagonal. These diagonal constraints are all of the form $|queens[i] - queens[j]| \neq |i - j|$ which is

```
MINION 3
**VARIABLES**
DISCRETE queens[4] {1..4}
# auxiliary variables
DISCRETE aux0 {-3..3}
DISCRETE aux1 {0..3}
DISCRETE aux2 {-3..3}
DISCRETE aux3 {0..3}
DISCRETE aux4 {-3..3}
DISCRETE aux5 {0..3}
DISCRETE aux6 {-3..3}
DISCRETE aux7 {0..3}
DISCRETE aux8 {-3..3}
DISCRETE aux9 {0..3}
DISCRETE aux10 {-3..3}
DISCRETE aux11 {0..3}
**SEARCH**
PRINT [queens]
VARORDER [queens,
aux0,aux1,aux2,aux3,aux4,aux5,aux6,aux7,
aux8,aux9,aux10,aux11]
**CONSTRAINTS**
weightedsumgeq([1,-1],[queens[2],queens[3]], aux0)
weightedsumleq([1,-1],[queens[2],queens[3]], aux0)
abs(aux1,aux0)
weightedsumgeq([1,-1],[queens[1],queens[3]], aux2)
weightedsumleq([1,-1],[queens[1],queens[3]], aux2)
abs(aux3,aux2)
weightedsumgeq([1,-1],[queens[1],queens[2]], aux4)
weightedsumleq([1,-1],[queens[1],queens[2]], aux4)
abs(aux5,aux4)
diseq(2, aux3)
weightedsumgeq([1,-1],[queens[0],queens[3]], aux6)
weightedsumleq([1,-1],[queens[0],queens[3]], aux6)
abs(aux7,aux6)
weightedsumgeq([1,-1],[queens[0],queens[2]], aux8)
weightedsumleq([1,-1],[queens[0],queens[2]], aux8)
abs(aux9,aux8)
weightedsumgeq([1,-1],[queens[0],queens[1]], aux10)
weightedsumleq([1,-1],[queens[0],queens[1]], aux10)
abs(aux11,aux10)
diseq(3, aux7)
diseq(2, aux9)
diseq(1, aux1)
diseq(1, aux5)
diseq(1, aux11)
alldiff([queens])
**EOF**
```

Figure 3.4: Minion input for N-Queens problem

decomposed into $queens[i] - queens[j] = auxa$, $|auxa| = auxb$ and $auxb \neq constant$. As minion has no weighted sum equals constraint, therefore two constraints are used that when used together offer the same functionality, namely a weighted sum less than or equals to ($weightedsumleq$) and weighted sum greater than or equals to ($weightedsumgeq$). The full constraint $queens[i] - queens[j] = auxa$ is represented as $queens[i] - queens[j] \leq auxa$ and $queens[i] - queens[j] \geq auxa$.

The output of Minion for the N-Queens problem where $n = 4$ is given in Figure 3.5.

```
# Minion Version 0.10
# Command line: ./minion -timelimit 200 -sollimit 1 input.minion
Parsing Time: 0.000000
Setup Time: 0.004000
First Node Time: -0.000000
Initial Propagate: -0.000000
First node time: -0.000000
Sol: 2 4 1 3

Solution Number: 1
Time:-0.000000
Nodes: 5

Solve Time: 0.204013
Total Time: 0.208013
Total System Time: 0.020001
Total Wall Time: 0.233183
Maximum Memory (kB): 37012
Total Nodes: 5
Problem solvable?: yes
Solutions Found: 1
```

Figure 3.5: Minion output for N-Queens problem

3.2.4 SAT Solvers

Boolean Satisfiability solving (SAT) refers to the assignment of variables in a propositional formula so that the formula evaluates to true. The following is an example of a SAT problem with the idea being to find assignments of true or false to each of the variables A, B, C and D such

that the whole equation is satisfied.

$$(A \vee B) \wedge (C \vee D) \wedge (\sim A \vee \sim C) \wedge (\sim D \vee B) \wedge (\sim B \vee C)$$

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm [DLL62] is the most common method used by SAT solvers which is based upon the earlier Davis-Putnam algorithm [DP60]. It works by considering partial assignments of values to literals in the clauses, propagating the impact of that assignment and back-tracking whenever a contradiction is detected. SAT is used in various applications including theorem proving, bounded model checking, circuit testing, logic synthesis, artificial intelligence planning and software verification. A comparison of SAT techniques for solving satisfiability problems is given in [SML96]. Some highly efficient implementations of SAT solvers include RSAT [PD07],
PICOSAT [Bie08], MiniSat [ES03] and zChaff [FMM].
We have experimented with MiniSat [ES03] and zChaff [FMM] in our research work. Both these SAT solvers accept the DIMACS CNF format as input. DIMACS CNF is the standard input format used by most of the state of the art SAT solvers. This format defines a Boolean expression, written in conjunctive normal form (CNF), that may be used as an example of the satisfiability problem.
Let us consider the following example of a boolean expression in CNF:

$$(x_1 \mid \sim x_5 \mid x_4) \,\&\, (\sim x_1 \mid x_5 \mid x_3 \mid x_4) \,\&\, (\sim x_3 \mid \sim x_4).$$

The above example can be written in DIMACS CNF format as shown in Figure 3.6.
Every line beginning with "c" is a comment. The first non-comment line introduces the SAT problem in CNF format with 5 variables and 3

```
c Example file.
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

Figure 3.6: Example in DIMACS CNF format

clauses. Each non-comment line that follows defines a clause that is a list of variables separated by space. A positive value represents the corresponding variable and a negative value represents the negation of that variable. Each line ends in a space and the number 0. The SAT solver finds the set of boolean variable assignments that make all the clauses true. Figure 3.7 and Figure 3.8 present the solutions found by the zChaff and MinSat SAT solvers for the example given in Figure 3.6.

```
c 3 Clauses are true, Verify Solution successful.
Instance Satisfiable
-1 2 -3 4 -5
Random Seed Used                                0
Max Decision Level                              3
Num. of Decisions                               4
( Stack + Vsids + Shrinking Decisions )         0 + 3 + 0
Original Num Variables                          5
Original Num Clauses                            3
Original Num Literals                           9
Added Conflict Clauses                          0
Num of Shrinkings                               0
Deleted Conflict Clauses                        0
Deleted Clauses                                 0
Added Conflict Literals                         0
Deleted (Total) Literals                        0
Number of Implication                           5
Total Run Time                                  0
RESULT: SAT
```

Figure 3.7: Solution given by zChaff

3.2.5 Model Generators

Model generators find assignments to the elements of algebraic formulae (i.e. predicates, variables and functions) in first-order logic such that the

```
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:             5                                         |
|  Number of clauses:               3                                         |
|  Parse time:                   0.00 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
===============================================================================
restarts              : 1
conflicts             : 0              (-nan /sec)
decisions             : 6              (0.00 % random) (inf /sec)
propagations          : 5              (inf /sec)
conflict literals     : 0              (-nan % deleted)
Memory used           : 8.00 MB
CPU time              : 0 s

SATISFIABLE: -1 -2 -3 -4 -5 0
```

Figure 3.8: Solution given by MiniSat

whole set of formulae is satisfied. Model generators were developed for the purpose of finding a concrete example which categorically disproves the theorem in the case when a proof for the theorem cannot be found by a theorem prover. There are some similarities between a model generator and constraint solver, for instance model generators find assignments to variables where the assignments are often constrained by some notion of the domain in which the formulae are set, for example its size and element types. However, there are also some differences such as their input syntax, which is normally sets of logical formulae rather than explicit variable definitions and constraints. Model generators are also related to SAT solvers as finding the satisfiability of a set of propositional formulae, as done by SAT solvers. SAT solvers can be applied to a sub-set of model generation problems. Several examples of model generators exist such as Mace4 [McC03a], Sem [ZZ01] and Finder [Sla94]. We have used Mace4 in our research work. Mace4 is a model generator that searches for finite models. It comes in a package along with the

Prover9 automated theorem prover. Mace4 can also serve as a complement of Prover9, to find counterexamples. The syntax of the input file for Mace4 is the same as Prover9 and the preprocessing is also done in the same manner. Figure 3.9 shows an example input file that contains the axioms for a non-commutative group using quantified variables.

```
formulas(assumptions).

% Axiom for associativity.
(x * y) * z = x * (y * z).

% Axiom for a left identity element and left inverse.
exists e ((all x (e * x = x)) &
(all x exists y (y * x = e))).

% Axiom for non-commuting elements.
exists a exists b (a * b != b * a).

end_of_list.
```

Figure 3.9: Mace4 Input Example

Mace4 uses the following steps to compute finite models:

1. The domain size is fixed to n where the members of the domain are $\{0, \ldots, n-1\}$.

2. Tables for the function and predicate symbols are set up.

3. All ground instances of the input clauses over the domain are generated.

4. A recursive backtracking procedure fills in the cells of the tables and uses the ground clauses to propagate the effects of the assignments.

5. When contradictions are encountered, backtracking occurs, the propagations and assignments are undone, and other assignments are attempted.

6. If all the tables become full, with no contradictions, a model is found.

7. If Mace4 is iterating through domain sizes, this procedure applies, separately, to each domain size.

Figure 3.10 presents the model found for a non-commutative group of size 6.

```
============================== DOMAIN SIZE 6 =========================

============================== MODEL =================================

interpretation( 6, [number=1, seconds=0], [

        function(c1, [ 0 ]),

        function(c2, [ 1 ]),

        function(c3, [ 2 ]),

        function(f1(_), [ 0, 1, 2, 4, 3, 5 ]),

        function(*(_,_), [
                          0, 1, 2, 3, 4, 5,
                          1, 0, 3, 2, 5, 4,
                          2, 4, 0, 5, 1, 3,
                          3, 5, 1, 4, 0, 2,
                          4, 2, 5, 0, 3, 1,
                          5, 3, 4, 1, 2, 0 ])
]).

============================== end of model ==========================

============================== STATISTICS ============================

For domain size 6.

Current CPU time: 0.00 seconds (total CPU time: 0.00 seconds).
Ground clauses: seen=229, kept=229.
Selections=14, assignments=44, propagations=95, current_models=1.
Rewrite_terms=1596, rewrite_bools=347, indexes=379.
Rules_from_neg_clauses=9, cross_offs=141.

============================== end of statistics =====================

User_CPU=0.00, System_CPU=0.01, Wall_clock=0.

Exiting with 1 model.
```

Figure 3.10: Mace4 Output Model

3.3 Other Mathematical Tools Used

We have used Octave [Eat08] to perform computations on matrices for performing the verification of finite solutions in Chapter 11. Octave is an open-source interactive software system for numerical computations and graphics. It is particularly designed for matrix computations and its mostly syntax compatible with MATLAB.

A matrix is a rectangular array of numbers, the size of which is usually defined as $m \times n$, meaning that it has m rows and n columns. An example of a 2×3 matrix is given as follows:

$$A = \begin{bmatrix} 4 & 8 & 3 \\ 1 & 6 & 2 \end{bmatrix}$$

The matrix A can be created in Octave by using the command:

octave:1 > $A = [4\ 8\ 3;\ 1\ 6\ 2]$

There are functions to create frequently used $m \times n$ matrices. If $m = n$, only one argument is necessary.

- $eye(m, n)$ produces a matrix with ones on the main diagonal and zeros elsewhere. When $m = n$, the identity matrix is generated.
- $zeros(m, n)$ generates the zero matrix of dimension $m \times n$.
- $ones(m, n)$ generates an $m \times n$ matrix where all entries are 1 .
- $rand(m, n)$ generates a random matrix whose entries are uniformly distributed in the interval $(0, 1)$.

The basic matrix arithmetic operations on matrix A are defined as follows:

- $+$, $-$, and $*$ denote matrix addition, subtraction, and multiplication.

- A' transposes and conjugates A.
- $A.'$ transposes A.

Element-wise operations on a matrix are defined as follows:

- $.*$ denotes element-wise multiplication.
- $./$ denotes element-wise division.
- $.\hat{}$ denotes element-wise power operators.

Part II

Structural Domain Knowledge Exploration for Large Size Example Generation

CHAPTER 4

BACKGROUND ON QUASIGROUPS

Chapter Overview: *This chapter provides the necessary background on quasigroups, which are defined along with their operations. Furthermore, some interesting properties of quasigroups, including the common properties as well as non-trivial two-variable properties and implied constraints are defined. Finally, a brief description of quasigroup equivalence classes is given.*

Quasigroups are non-associative algebraic structures whose operation has to satisfy only a single axiom, the Latin square property. There exists a very large number of different finite quasigroups even for small orders. This makes them ideal candidates for applications where the generation of a large number of simple structures is necessary, such as in cryptography. However, the lack of structure makes them difficult to handle algebraically, in particular to enumerate or to classify. We have developed methods to automatically generate relatively large size

quasigroups by bootstrapping structural properties of smaller size quasigroups by computing useful additional knowledge. By "relative" we mean compared to those produced in previous approaches. The techniques for discovering the additional knowledge have allowed us to push the boundaries of current automated reasoning systems for model generation in computing large size examples of quasigroups with interesting properties. More details of these techniques are presented in Chapter 6.

In this chapter, we provide the necessary background on quasigroups. We present a formal definition of quasigroups and the three essential binary operations i.e. multiplication, right division / and left division \. We also provide examples of quasigroups in terms of the multiplication tables for all three operations. We define some interesting properties of quasigroups that are taken from the literature have been used in our research. The non-trivial two-variable properties that we present are not defined with uniform names in the literature, therefore we represent them with names that might be different from the ones given in the literature. Finally, we define some quasigroup equivalence relations.

4.1 Quasigroup Definition and Operations

We define the notion of a quasigroup following [Pfl90, MGA97, Smi06].

Definition 4.1.1 *Let Q be a non-empty set along with a multiplication operation '$*$'. Then $(Q, *)$ is a quasigroup if it has the following properties:*

(1) For all $a, b \in Q, a * b \in Q$ (that is, Q is closed under $*$)

(2) For all $a, b \in Q$, there exist unique $x, y \in Q$ s.t., $x * a = b$ and $a * y = b$ (i.e., $(Q, *)$ has unique solubility of equations)

In our research, we are exclusively interested in finite quasigroups and we generally define a quasigroup $(Q, *)$ of size n over a set of elements $Q = \{0, 1, \ldots, n-1\}$. Also if there is no ambiguity, we denote the binary operation $*$ in a quasigroup Q by juxtaposition, e.g. $x * y$ can be written as xy.

The unique solubility of equations in Definition 4.1.1 ensures that each element of Q occurs exactly once in each row and each column of the multiplication table of $(Q, *)$. Each row and each column is a permutation of the elements of Q. If $|Q| = n$, then the Cayley table for $(Q, *)$ forms an n by n Latin Square consisting of n elements each of which appears exactly once in each row and each column. Conditions (1) and (2) essentially postulate the existence of unique left and right divisors for each element in Q. Thus, $(Q, *, /, \backslash)$ can be defined as a quasigroup having three binary operations of multiplication, right division / and left division $\backslash$ such that for every $a, b, c \in Q$, $a * b = c \Leftrightarrow c/b = a \Leftrightarrow a\backslash c = b$ with the following identities being satisfied:

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

x/y is read as "x divided by y" or "x over y" and $x\backslash y$ is read as "x dividing y" or "x into y".

The following is an example of a Quasigroup $(Q, *, \backslash, /)$ of size 4 given in terms of multiplication tables for all three operations:

*	1	2	3	4
1	2	3	1	4
2	4	1	3	2
3	3	4	2	1
4	1	2	4	3

\	1	2	3	4
1	3	1	2	4
2	2	4	3	1
3	4	3	1	2
4	1	2	4	3

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

4.2 Quasigroup Properties

We are interested in the goal directed construction of quasigroups with certain properties. There are a large number of interesting properties one can define on quasigroups and that can be found in the literature (e.g., [NV] and [BL07]). We have focused in our experiments on a number of non-trivial properties given in [BL07].

We first define the common properties and then the non-trivial two-variable properties. The binary operation $*$ in quasigroup Q is denoted by juxtaposition in the following sections.

Idempotent, unipotent and commutative quasigroups are defined as follows:

Definition 4.2.1 *Let Q be a quasigroup and $x, y \in Q$, then Q is:*

(i) ***Idempotent*** *if $x^2 = x$ for every $x \in Q$.*

(ii) ***Unipotent*** *if $x^2 = y^2$ for every $x, y \in Q$.*

(iii) ***Commutative*** *if $xy = yx$ for every $x, y \in Q$.*

We now define the non-trivial two-variable properties for quasigroups that are generalizations of the common properties defined above. These properties were suggested by Frank Bennett and are defined in [BZ92].

Definition 4.2.2 *Let Q be a quasigroup and $x, y \in Q$, then we define the following properties for Q with their descriptive names given in brackets:*

Qg-1 *: $xy * yx = x$ for all $x, y \in Q$; (Schröder quasigroup)*

Qg-2 *: $yx * xy = x$ for all $x, y \in Q$; (Stein's third law)*

Qg-3 *: $(xy * y)y = x$ for all $x, y \in Q$; (C_3-quasigroup)*

Qg-4 *: $x * xy = yx$ for all $x, y \in Q$; (Stein's first law)*

Qg-5 *: $(yx * y)y = x$ for all $x, y \in Q$;*

Qg-6 *: $yx * y = x * yx$ for all $x, y \in Q$; (Stein's second law)*

Qg-7 *: $xy * y = x * xy$ for all $x, y \in Q$; (Schröder's first law)*

Let Q be a quasigroup and $x, y \in Q$, then we define the following additional properties for Q that the quasigroups with the two-variable properties defined above possess. These properties were computed in [CM01, CCM06], as constraints implied from axioms, but we use them as additional constraints.

Definition 4.2.3 *Let Q be a quasigroup and $x, y \in Q$, then we define the following properties for Q:*

C_1 *: $\forall\, x\, \exists\, y\, (y * y = x)$ (all different diagonal)*

C_2 *: $\forall\, x, y\, (x \neq y) \rightarrow (x * y \neq y * x)$ (anti-Abelian)*

C_3 *: $\forall\, x, y\, (x * x = y) \rightarrow (y * y = x)$ (diagonal symmetry)*

C_4 *: $\forall\, x, y\, (x * y = y) \rightarrow (y * x = x)$ (symmetry of left identities)*

C_5 *: $\forall\, x, y\, (x * x = x)$ (idempotent)*

C_6 *: $\forall\, x, y\, (x * y = x) \leftrightarrow (y * x = x)$*

We now describe the quasigroup equivalence relations that were first introduced in Chapter 2, where the related work that involves the classification of quasigroups into isomorphic and isotopic classes was presented. Following [CDS91], the isotopic and isomorphic equivalence relations can be defined as follows:

Definition 4.2.4 *Let $(L, \circ)$ and $(M, *)$ be two quasigroups. An ordered triple (α, β, γ) of one-to-one mappings α, β, γ of the set L onto the set M is called an isotopism of $(L, \circ)$ upon $(M, *)$, if $(\alpha x) * (\beta y) = \gamma(x \circ y)$ for all $x, y \in L$, where αx is the result of applying α to x. If such an isotopism exists, then quasigroups $(L, \circ)$ and $(M, *)$ are said to be isotopic. The equivalence classes of quasigroups under the isotopy relation are called isotopy classes.*

Example 4.2.1 *We can transform the multiplication table of one quasigroup into another quasigroup by performing any or all three of the following operations:*

- *Permute the rows.*
- *Permute the columns.*
- *Permute the symbols (rename the symbols without changing their relative positions).*

Two quasigroups are isotopic if we can change one to another by using only the above operations. The following two quasigroups Q_1 and Q_2 are isotopic:

Q_1	1	2	3	4
1	1	2	3	4
2	3	4	1	2
3	4	3	2	1
4	2	1	4	3

Q_2	1	2	3	4
1	1	2	3	4
2	4	3	2	1
3	2	1	4	3
4	3	4	1	2

This is because, if the second row of the quasigroup Q_1 is moved into the position of the last row, and third and fourth rows are moved up by one position, then the two quasigroups Q_1 and Q_2 become identical.

Definition 4.2.5 *Two quasigroups are said to be isomorphic if the mappings α, β, γ are equal. An isomorphism from $(L, \circ)$ to $(M, *)$ is a bijective function $f : L \longrightarrow M$ if, for all $a, b \in L$, we have:*
$f(a) * f(b) = f(a \circ b)$. *The equivalence classes of quasigroups under the isomorphic relation are called isomorphism classes.*

CHAPTER 5

QUASIGROUP MODEL GENERATION PROBLEMS AND ENCODINGS

Chapter Overview: *This chapter describes the three main quasigroup model generation problems, which are: quasigroup constraint satisfaction problems, quasigroup satisfiability problems and quasigroup model generation problems. More specifically, we define how these problems are encoded and present the solution models given by the systems.*

In this chapter, we describe the diverse encoding techniques used for defining quasigroups for different model generation systems. These encoding techniques have been taken from the literature. In Section 5.1, we present a definition of a quasigroup constraint satisfaction problem (CSP) and describe the three models that can be used for encoding a quasigroup CSP. In Section 5.2, we describe the minimal and extended encoding used for quasigroup satisfiability problem. Finally, in Section 5.3, we present an encoding that uses axioms for defining a

quasigroup in first-order equational logic which can be used to express quasigroup model generation problems.

5.1 Quasigroup Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) is modelled by a set of constraints on a set of decision variables. A constraint solver then assigns values to each of the variables so that all the constraints are satisfied. The basic definition of a quasigroup constraint satisfaction problem is given as follows:

Definition 5.1.1 *A constraint satisfaction problem (CSP) for a quasigroup is a triple (V,D,C), consisting of*

- *a finite set V of variables which represent the Cayley table (i.e. multiplication table) entries of the quasigroup,*
- *a finite set D, called the domain, representing the elements of the quasigroup,*
- *a finite set of constraints C representing the quasigroup axioms, that assign values from D to variables in V*

Quasigroups of size n can be found by considering a table of n^2 variables of the form $x_{i,j}$ where $i, j \in \{0, 1, \ldots n-1\}$ with possible values in the domain $D = \{0, 1, \ldots, n-1\}$ and constraining the variables of each row and column to be all different by using the constraints such as $x_{i,j} \neq x_{k,j}$ and $x_{i,j} \neq x_{i,k}$ where $i, j, k \in \{0, 1, \ldots n-1\}$, and $i \neq k$, $j \neq k$.
As given in [Wal01], each row and column of a quasigroup multiplication table is a permutation problem, i.e. a constraint satisfaction problem with the same number of variables as values, where a solution is a

permutation of the values. The quasigroup CSP is a multiple permutation problem with $2n$ intersecting permutation constraints i.e. n row permutation constraints and n column permutation constraints. In order to represent those permutations, the quasigroup CSP can be encoded using three models which are given in [DdVC03b, DdVC03a] and are described as follows:

Primal Model: uses primal variables that are defined as the set:

$$X = \{x_{ij} \mid 0 \leq i \leq n-1,\ 0 \leq j \leq n-1\},$$

where the variable x_{ij} represents the cell in the i^{th} row and j^{th} column of the multiplication table of the quasigroup, and n is the size of the quasigroup. The domain of the variables can be defined as $D = \{k \mid 0 \leq k \leq n-1\}$, where each k represents an element of the quasigroup. The domain of possible values are the elements of the quasigroup. The primal constraints can be divided into:

(i) n^2 row constraints of the form $x_{ij} \neq x_{il}$ where $x_{ij}, x_{il} \in X$ and $j \neq l$, which means that two cells in the same row must not have the same element.

(ii) n^2 column constraints of the form $x_{ij} \neq x_{lj}$ where $x_{ij}, x_{lj} \in X$ and $i \neq l$, which means that two cells in the same column must not have the same element.

The above constraints can be implemented by using $2n$ alldiff constraints [Rég94], one for each row and column, thereby reducing the number of constraints.

Row Dual Model: uses the row dual variables that are defined as the

set:

$$R = \{r_{ik} \mid 0 \ \leq \ i \ \leq \ n-1, \ 0 \ \leq \ k \ \leq n-1\},$$

where the variable r_{ik} represents the k^{th} element that can be placed in the i^{th} row of the multiplication table of the quasigroup. The domain of each variable is the set $D = \{j \mid 0 \ \leq \ j \ \leq \ n-1 \ \}$, where j represents the columns or the positions in row i where element k can be placed. The row dual constraints are given as follows:

(i) n^2 constraints of the form $r_{ik} \neq r_{il}$, where $r_{ik}, \ r_{il} \in R$ and $l \neq k$, which means that two elements in the same row must not be assigned to the same column.

(ii) n^2 constraints of the form $r_{ik} \neq r_{jk}$ where $r_{ik}, \ r_{jk} \in R$ and $i \neq j$, which means that the same element in different rows must not be assigned to the same column.

The above constraints are implemented by having alldiff$(r_{i0}, ..., r_{i(n-1)})$ for every row i, and alldiff$(r_{0k}, ..., r_{(n-1)k})$ for every element k.

Column Dual Model: uses the column dual variables that are defined as the set:

$$C = \{c_{jk} \mid 0 \ \leq \ j \ \leq \ n-1, \ 0 \ \leq \ k \ \leq \ n-1\},$$

where c_{jk} is the k_{th} element that can be placed in the j_{th} column of the multiplication table of the quasigroup. All variables have domain $D = \{i \mid 0 \ \leq \ k \ \leq \ n-1\}$, where i represents the rows or the positions in column j where element k can be placed. The column dual constraints are given as follows:

(i) n^2 constraints of the form $c_{jk} \neq c_{jl}$ where $c_{jk},\ c_{jl} \in C$ and $k \neq l$, which means that two elements in the same column must not be assigned to the same row.

(ii) n^2 constraints of the form $c_{jk} \neq c_{lk}$ where $c_{jk},\ c_{lk} \in C$ and $j \neq l$, which means that the same element in different columns must not be assigned to the same row.

We have used the essence modelling language [FGJ+07] to specify the quasigroup CSP in primal model. An example essence specification for QG-1 quasigroups of size 4 is given in Figure 5.1.

```
letting nDomain be domain int(0..3)
find quasiGroup : matrix indexed by [nDomain, nDomain] of nDomain
such that
$ All rows have to be different
forall row : nDomain .
allDifferent(quasiGroup[row,..]),
$ All columns have to be different
forall col : nDomain .
allDifferent(quasiGroup[..,col]),
$ (j*i)*(i*j) = i
forall i : nDomain .
forall j : nDomain .
quasiGroup[quasiGroup[i,j],quasiGroup[j,i]] = i
```

Figure 5.1: Essence specification (Primal model) for a Qg-1 quasigroup of size 4

The solution model for a QG-1 quasigroup of size 4 found by Minion is shown in Figure 5.2.

5.2 Quasigroup Satisfiability Problems (SAT)

The quasigroup satisfiability problem (SAT) consists of a logical propositional formula with n variables $x_1, x_2, ..., x_n$, which can be assigned truth values true of false. A literal l is either a variable x_i (i.e., a positive literal) or its complement $\neg x_i$ (i.e., a negative literal). A

```
# Minion Version 0.10
# Command line: ./minion -timelimit 7200 -sollimit 1 one4-input.minion
Parsing Time: 0.001999
Setup Time: 0.005999
First Node Time: 0.000000
Initial Propagate: 0.000000
First node time: 0.000000
Sol: 0 2 3 1
Sol: 3 1 0 2
Sol: 1 3 2 0
Sol: 2 0 1 3

Solution Number: 1
Time:0.000000
Nodes: 7

Solve Time: 0.211968
Total Time: 0.219966
Total System Time: 0.038994
Total Wall Time: 0.458027
Maximum Memory (kB): 0
Total Nodes: 7
Problem solvable?: yes
Solutions Found: 1
```

Figure 5.2: Minion output model for a Qg-1 quasigroup of size 4

clause is a disjunction of literals and a formula is a conjunction of clauses. For the formula to be satisfiable, one needs to find a variable assignment that makes the formula true, which can then be translated into the quasigroup model. Equations of the form $x_i = x_j$ and $x_i = x_j * x_k$ and their negations can be translated into literals directly, however, nested equations have to be transformed into sequences of equations first. There are two main SAT encodings that have been previously studied for quasigroups [GS02b, KRA+01], which are defined as follows:

(i) *Minimal Encoding:* This is the most basic SAT encoding which includes clauses that represent the following constraints (variable q_{xyz} represents that the z^{th} element of the quasigroup is assigned to the cell at x^{th} row and y^{th} column in the multiplication table of the quasigroup Q, where $x, y, z \in Q$):

- Each cell has to have an element assigned to it:

$$\bigwedge_{x=0}^{n-1} \bigwedge_{y=0}^{n-1} \bigvee_{z=0}^{n-1} q_{xyz}$$

- An element cannot be repeated in the same row:

$$\bigwedge_{y=0}^{n-1} \bigwedge_{z=0}^{n-1} \bigwedge_{x=0}^{n-2} \bigwedge_{i=x+1}^{n-1} (\neg q_{xyz} \vee \neg q_{xiz})$$

- An element cannot be repeated in the same column:

$$\bigwedge_{x=0}^{n-1} \bigwedge_{z=0}^{n-1} \bigwedge_{y=0}^{n-2} \bigwedge_{i=y+1}^{n-1} (\neg q_{xyz} \vee \neg q_{iyz})$$

(ii) *Extended Encoding:* This extends the minimal encoding by adding the following constraints :

- Each element must appear at least once in each row:

$$\bigwedge_{x=0}^{n-1} \bigwedge_{z=0}^{n-1} \bigvee_{y=0}^{n-1} q_{xyz}$$

- Each element must appear at least once in each column:

$$\bigwedge_{y=0}^{n-1} \bigwedge_{z=0}^{n-1} \bigvee_{x=0}^{n-1} q_{xyz}$$

- No two elements can be assigned to the same cell:

$$\bigwedge_{x=0}^{n-1} \bigwedge_{y=0}^{n-1} \bigwedge_{z=0}^{n-2} \bigwedge_{i=z+1}^{n-1} (\neg q_{xyz} \vee \neg q_{xyi})$$

We use Stickel's quasigroup generator [Sti] that generates quasigroup SAT problem extended encoding in the DIMACS CNF format which is a standard way to represent conjunctive normal form Boolean formulae.

The solution models found by MiniSat and zChaff SAT solvers for Qg-1 quasigroup of size 4 are shown in Figure 5.3 and Figure 5.4.

```
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:            64                                         |
|  Number of clauses:             519                                         |
|  Parse time:                   0.00 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
==============================================================================
==============================================================================
restarts              : 1
conflicts             : 0              (0 /sec)
decisions             : 3              (0.00 % random) (3003 /sec)
propagations          : 64             (64064 /sec)
conflict literals     : 0              ( nan % deleted)
Memory used           : 8.00 MB
CPU time              : 0.000999 s

SATISFIABLE: -1 -2 -3 -4 -5 -6 7 8 -9 -10 -11 -12 -13 14 -15 -16 17 -18 -19
-20 21 -22 -23 -24 -25 26 -27 28 -29 -30 -31 -32 -33 -34 35 36 -37 -38 -39
-40 -41 42 -43 44 -45 -46 -47 -48 49 -50 -51 -52 53 -54 55 -56 -57 -58 -59
60 61 -62 -63 -64 0
```

Figure 5.3: MiniSat output model for a Qg-1 quasigroup of size 4

5.3 Quasigroup Model Generation Problems

Model generators such as Mace4 [McC03a] can be used to find finite quasigroup models by using a set of axioms that define a quasigroup in first order equational theory by interpreting the primitives (i.e., predicates, variables and functions) over a finite domain D_n in order to satisfy all the axioms, where n is the size of the quasigroup. If a concrete model is found, it proves the existence of a quasigroup having a certain property for a particular size. Consequently for proof problems, a counter model categorically disproves the theorem. While the quasigroup model generation problem can be given in a straight forward manner in

```
c 567 Clauses are true, Verify Solution successful.
Instance Satisfiable
-1 -2 -3 -4 -5 -6 7 8 -9 -10 -11 -12 -13 14 -15 -16 17 -18 -19 -20
21 -22 -23 -24 -25 26 -27 28 -29 -30 -31 -32 -33 -34 35 36 -37 -38
-39 -40 -41 42 -43 44 -45 -46 -47 -48 49 -50 -51 -52 53 -54 55 -56
-57 -58 -59 60 61 -62 -63 -64
Random Seed Used                                    0
Max Decision Level                                  4
Num. of Decisions                                   5
( Stack + Vsids + Shrinking Decisions )             0 + 4 + 0
Original Num Variables                              64
Original Num Clauses                                567
Original Num Literals                               1443
Added Conflict Clauses                              0
Num of Shrinkings                                   0
Deleted Conflict Clauses                            0
Deleted Clauses                                     0
Added Conflict Literals                             0
Deleted (Total) Literals                            0
Number of Implication                               64
Total Run Time                                      0
RESULT: SAT
```

Figure 5.4: zChaff output model for a Qg-1 quasigroup of size 4

terms of the formulas, the domain size is usually specified as an additional parameter to Mace4. However, our initial experiments have shown that the performance of Mace4 can be improved by constraining the domain size explicitly by giving appropriate equality constraints, Figure 5.5 presents the input file encoding for Mace4 to generate a Qg-1 quasigroup of size 4.

Figure 5.6 presents the output solution model found by Mace4 for a Qg-1 quasigroup of size 4.

```
assign(max_seconds,7200).
assign(domain_size, 4).
assign(max_models, 1).
set(print_models).
formulas(assumptions).
% quasigroup definition
x * (x \ y) = y.
x \ (x * y) = y.
(x / y) * y = x.
(x * y) / y = x.
% size4
all x ( x=0 |  x=1 |  x=2 |  x=3).
% quasigroup property
all x all y ((x * y) * (y * x) = x).
end_of_list.
```

Figure 5.5: Mace input file for a Qg-1 quasigroup of size 4

```
============================== DOMAIN SIZE 4 =========================

============================== MODEL =================================

interpretation( 4, [number=1, seconds=0], [

        function(*(_,_), [
                            0, 2, 3, 1,
                            3, 1, 0, 2,
                            1, 3, 2, 0,
                            2, 0, 1, 3 ]),

        function(/(_,_), [
                            0, 3, 1, 2,
                            2, 1, 3, 0,
                            3, 0, 2, 1,
                            1, 2, 0, 3 ]),

        function(\(_,_), [
                            0, 3, 1, 2,
                            2, 1, 3, 0,
                            3, 0, 2, 1,
                            1, 2, 0, 3 ])
]).

============================== end of model ==========================

============================== STATISTICS ============================

For domain size 4.

Current CPU time: 0.00 seconds (total CPU time: 0.01 seconds).
Ground clauses: seen=84, kept=80.
Selections=4, assignments=12, propagations=69, current_models=1.
Rewrite_terms=240, rewrite_bools=102, indexes=52.
Rules_from_neg_clauses=17, cross_offs=124.

============================== end of statistics =====================

User_CPU=0.01, System_CPU=0.00, Wall_clock=0.

Exiting with 1 model.

Process 21806 exit (max_models) Wed Jun 13 16:05:44 2012
The process finished Wed Jun 13 16:05:44 2012
```

Figure 5.6: Mace output model for Qg-1 quasigroup of size 4

CHAPTER 6

ENRICHING QUASIGROUP PROBLEMS WITH PRE-COMPUTED KNOWLEDGE

Chapter Overview: *This chapter describes our proposed approaches that exploit the structural domain knowledge of quasigroups. The first approach is based on randomization, where symbolic computations and automated theorem proving is used to exclude unsuitable instantiations. The second approach employs a concept of generating systems particularly suitable for quasigroups that can be easily computed for small size quasigroups and then evolved to represent quasigroups of larger sizes. The evolution of the generating systems is done by using symbolic computations and automated theorem proving.*

In this chapter, we describe the two novel approaches we have proposed for the automated theory exploration of quasigroup structures to compute additional knowledge that can help in the discovery of large size solutions for quasigroups with interesting properties. We describe our

first approach in Section 6.1 where randomly computed elements pass through two algebraic filters and are only kept if they pass the property test. We then describe our second approach in Section 6.2, which uses a concept of generating systems particularly suitable for quasigroups. We provide a formal definition of generating systems, and describe their computation and evolution.

6.1 Quasigroup Element Filtering

Our first approach to narrow down the search space of our quasigroup problems is an intelligent quasigroup element computation method that uses algebraic filter criteria on two levels when pre-setting elements. A diagrammatic overview of this approach is shown in Figure 6.1.

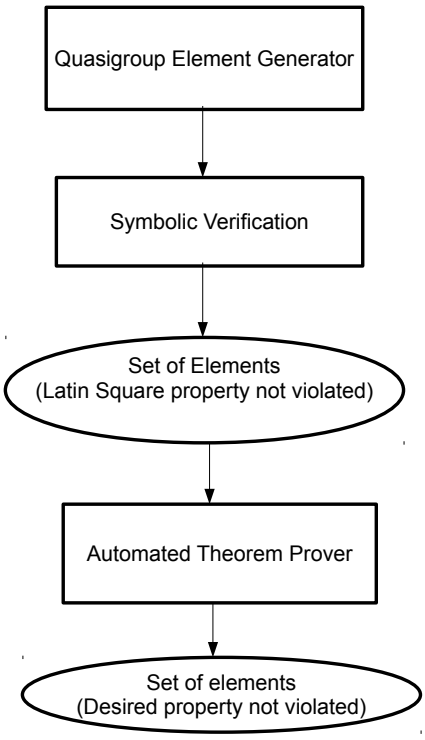


Figure 6.1: Flow diagram of the quasigroup element filtering approach

For a quasigroup Q we randomly generate triples that are added to a set of the form $S = \{(r, c, e)|r, c, e \in Q\}$. Every time an element is added,

we use a symbolic verification function to check that all the elements in the set are unique and that the Latin square property is not violated. We continue this process until we obtain the set S_F of filtered elements that is of a particular pre-defined size. Generally, we specify the size of S_F as a multiple of the size n of the quasigroup Q.

In a second filter step, we then check for the entire set S_F that its elements do not violate the desired quasigroup property P — which can in general be a combination of properties — using Prover9. Figure 6.2 shows an example input encoding of the proof problem given to Prover9.

```
assign(max_seconds,10).
formulas(sos).
% quasigroup definition
all a all b exists x exists y ( (a * x = b) & (y * a = b) ).
% restricting the domain size to 5
exists a0  exists a1  exists a2  exists a3  exists a4  all x
( (a0 != a1 & a0 != a2 & a0 != a3 & a0 != a4 & a1 != a2 &
a1 != a3 & a1 != a4 & a2 != a3 & a2 != a4 & a3 != a4)
&  (x = a0 | x = a1 | x = a2 | x = a3 | x = a4) ).
% set of computed elements
exists a0  exists a3  exists a4  exists a2  exists a1
( (a0 != a1 & a0 != a2 & a0 != a3 & a0 != a4 & a1 != a2 &
a1 != a3 & a1 != a4 & a2 != a3 & a2 != a4 & a3 != a4) &
(a0 * a3 = a4) &  (a2 * a0 = a4) &  (a0 * a4 = a1) &  (a0 * a2 = a0)
&  (a1 * a1 = a2)  ).
end_of_list.

formulas(goals).
% desired quasigroup property
-(all x all y ((x * y) * (y * x) = x)).
end_of_list.
```

Figure 6.2: Quasigroup proof problem encoding

The encoding for the proof problem is described as follows:

Assumptions: (i) Quasigroup axioms as given in (Def. 4.1.1).

(ii) Quasigroup size axioms; that is, there exists exactly n elements that are all different.

(iii) The equations for the set of pre-set elements S_F.

Goal: $\neg P$, the negation of the conjunction of properties of the

quasigroup.

If Prover9 finds a proof, an example of which is shown in Figure 6.3, then this means that the set of pre-set elements S_F violates the property P and therefore we compute a new S_F. Otherwise, if Prover9 is unable to find the proof within a fixed time period, we assume that the set of pre-set elements S_F does not violate the property P.

```
============================== PROOF =================================

% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 8.
% Level of proof is 2.
% Maximum clause weight is 9.
% Given clauses 29.

3 (exists a0 exists a3 exists a4 exists a2 exists a1 (a0 != a1 &
a0 != a2 & a0 != a3 & a0 != a4 & a1 != a2 & a1 != a3 & a1 != a4 &
a2 != a3 & a2 != a4 & a3 != a4 & a0 * a3 = a4 & a2 * a0 = a4 &
a0 * a4 = a1 & a0 * a2 = a0 &
a1 * a1 = a2)) # label(non_clause).  [assumption].
4 -(all x all y (x * y) * (y * x) = x) # label(non_clause)
# label(goal).  [goal].
18 c10 != c6.  [clausify(3)].
34 c9 * c6 = c8.  [clausify(3)].
35 c6 * c8 = c10.  [clausify(3)].
36 c6 * c9 = c6.  [clausify(3)].
38 (x * y) * (y * x) = x.  [deny(4)].
85 $F.  [para(34(a,1),38(a,1,2)),rewrite([36(3),35(3)]),unit_del(a,18)].

============================== end of proof ==========================
```

Figure 6.3: Proof found by Prover9

We only run Prover9 on a full set of elements (minus the partial quasigroup) in order to more efficiently compute S_F, as generating the input file and calling Prover9 for every single new element slows down the filtering process far more than having to occasionally recompute the entire set, which in practice does not happen too often. The process is more efficient if one uses bespoke algebraic functions to test the single properties in P. However, this would mean we would have to write new code every time a new property is tested. More details on the experiments and results are given in Chapter 7.

6.2 Generating System Representation for Quasigroups

Our second approach uses knowledge based refinement of our problem domain. We define a concept of generating systems for quasigroups which is very similar to the presentations for groups, i.e., it consists of sets of generators and relations. However, contrary to the group presentations in our generating systems each element of the generated quasigroup has to be explicitly represented in terms of a relation in the generators. As a consequence, when increasing the number of relations, we increase the number of elements in the quasigroup that are generated, which is counter-intuitive to the group theoretical notion.
The concept of generating system that we use was first defined in [CMSM04], where it is used to determine a quasigroup structure of size n using n complex equations rather than n^2 simple equations of its Cayley table. The concept is closely related to the one defined in [Eva51].

Definition 6.2.1 *Let $S = \{a_0, \ldots, a_{n-1}\}$ be a finite set together with a binary operation $*$. We call the elements in S* generators *and define a* word *inductively as:*

- *$a_0, \ldots, a_{n-1}$ are words.*
- *if u, v are words, then so is $(u * v)$.*

Thus a word can be built from other words, which are called its *components*. The only component of a generator is the generator itself. The components of a word $w = u * v$ are the word itself and the components of u and v. We say the *generators of a word w* is the union of all generators contained in the components of w. We sometimes write

$w = w(a_1, ..., a_n)$ if $a_1, ..., a_n$ are the generators of w. Examples of words consisting of a single generator can be seen in Example 6.2.2, and furthermore examples of words with a combination of generators can be seen in Example 6.2.3.

We now define the concept of generating systems for quasigroups as follows:

Definition 6.2.2 *Let Q be a finite quasigroup of size n with binary operation $*$, and let $q_0, ..., q_{n-1} \in Q$ be the elements of Q. Let $\{a_0, ..., a_{m-1}\} \subseteq Q$ where $n, m \in \mathbb{N}$ and $0 \leqslant m - 1 \leqslant n - 1$. Then, we define the generating system G for Q as follows:*

$$G = \langle \{a_0, ..., a_{m-1}\} | \{q_1 = w(a_0, ..., a_{m-1}), ..., q_{n-1} = w(a_0, ..., a_{m-1})\} \rangle$$

where:

(i) The set $\{a_0, ..., a_{m-1}\} \subseteq Q$ is called the generators.

(ii) $\{w(a_0, ..., a_{m-1}), ..., w(a_0, ..., a_{m-1})\}$ represents a set of words. Every element $q \in Q$ can be expressed as a word which is called a relation or factorization.

The generating system for quasigroups is different from the one for groups in that every single element is explicitly defined. Moreover, generating systems or number of generators are not uniquely determined. In fact, it is not always desirable to have a minimal number of generators, as instead it can be useful to explicitly build in redundancy into generating systems for instance in the case of idempotent quasigroups.

Example 6.2.1 *Consider the following example of a Quasigroup $(Q, *)$*

of size 4 *given in terms of its multiplication table:*

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

Amongst others the quasigroup Q from Example 6.2.1 can be represented with the following generating systems (observe that we have omitted trivial factorizations like $2 = 2$):

$$G_1 = \langle \{2\} | \{0 = 2 * (2 * (2 * 2)),\ 1 = 2 * 2,\ 3 = 2 * (2 * 2)\} \rangle$$
$$G_2 = \langle \{1\} | \{0 = 1 * 1,\ 2 = (1 * 1) * 1,\ 3 = ((1 * 1) * 1) * 1\} \rangle$$
$$G_3 = \langle \{1, 2\} | \{0 = 2 * (2 * 1),\ 3 = 2 * 1\} \rangle$$

6.2.1 Computing Generating Systems for Quasigroups

We now present a method to compute generating systems from the Cayley table of a given quasigroup which is related to the method in [CMSM04]. Our approach aims to construct generating systems systematically by iteratively increasing the number of generators in a computationally efficient manner. It uses a concept of *traces* that corresponds to the set of all elements in a quasigroup reachable from combinations of a single element alone.

Definition 6.2.3 *Let Q be a quasigroup and let $q \in Q$. We define the* trace *of q in Q as the set $t(q) = \{q_0, \ldots, q_{n-1}\} \in Q$ of all elements in Q such that $q_i = w(q)$ for $0 \leq i \leq n-1$. Thus the q_i are all elements in Q that have q as a generator only.*

Observe that due to quasigroups in general being non-associative we can not simply express the generated elements in terms of powers of the generator q, as one can for example for the sub-group generated by one element. Consequently, when computing the traces, one has to consider all possible combinations of the generators in $w(q)$.

Example 6.2.2 *Consider the following quasigroup* $(Q', *)$ *of size* 4*:*

Q'	0	1	2	3
0	0	3	1	2
1	1	2	0	3
2	3	1	2	0
3	2	0	3	1

The traces of elements are as follows:

$$t(0) = \{0 = 0\}$$

$$t(1) = \{1 = 1,\ 2 = 1 * 1,\ 0 = 1 * (1 * 1),\ 3 = (1 * (1 * 1)) * 1\}$$

$$t(2) = \{2 = 2\}$$

$$t(3) = \{3 = 3,\ 1 = 3 * 3,\ 0 = 3 * (3 * 3),\ 2 = 3 * (3 * (3 * 3))\}$$

We can see that for example in trace $t(3)$ *it is important to actually consider both possible combinations of* 3 *and* 1 *as for example* $1 * 3 = 3$ *would have not yielded another element.*

We now define Algorithm 2 that uses traces to compute a generating system. We abuse the notation slightly, by using $t(q)$ to refer both to the elements of a trace as well as the relations that generate these elements. Let $(Q, *)$ be a quasigroup and $q, q_0, \ldots, q_{n-1} \in Q$. Also,

Algorithm 2 Compute generating systems G for quasigroup Q

Require: Quasigroup $(Q, *)$ of size n where $q, q_0, \ldots, q_{n-1} \in Q$
Require: G an empty list of generating systems
  if $t(q) = Q$ **then**
    $G_1 = \langle \{q\} | t(q) \rangle$
    Add G_1 to G
  end if
  if $t(q_0) \cup \ldots \cup t(q_{n-1}) = Q$ **then**
    $G_2 = \langle \{q_0, \ldots, q_{n-1}\} | t(q_0) \cup \ldots \cup t(q_{n-1}) \rangle$
    Add G_2 to G
  end if
  if $t(q_0) \cup \ldots \cup t(q_{n-1}) \subset Q$ and $Q \setminus t(q_0) \cup \ldots \cup t(q_{n-1}) = \{p_0, \ldots, p_{m-1}\}$ with $p_i = w(q_0, \ldots, q_{n-1}), 0 \leq i \leq m-1$ **then**
    $G_3 = \langle \{q_0, \ldots, q_{n-1}\} | t(q_0) \cup \ldots \cup t(q_{n-1}) \cup \{p_0 = w(q_0, \ldots, q_{n-1}), \ldots, p_m = w(q_0, \ldots, q_{n-1})\} \rangle$
    Add G_3 to G
  end if
  return G

$Q \setminus t(q_0) \cup \ldots \cup t(q_{n-1})$ denotes the elements of quasigroup Q that are not equal to $t(q_0) \cup \ldots \cup t(q_{n-1})$.

Observe that the three if conditions in Algorithm 2 are not mutually exclusive. That is, even if the first condition already yields a generating system, we can employ the other two conditions to construct generating systems with a larger number of generators. Similarly, although the second condition will always yield a generating system, this might contain too many generators and too much redundancy (e.g., consider an idempotent quasigroup; a generating system can comprise all of its elements with exclusively trivial relations). Consequently, the third condition generally gives a smaller set of generators and less trivial relations.

Concretely we have implemented second and third conditions iteratively, step-wise combining traces to obtain generating systems with the smallest possible number of generators. The following example illustrates this technique.

Example 6.2.3 *Consider the following quasigroup* $(Q, *)$ *with* 6 *elements:*

$*$	0	1	2	3	4	5
0	1	0	4	5	2	3
1	0	1	5	4	3	2
2	4	5	3	2	0	1
3	5	4	2	3	1	0
4	3	2	0	1	5	4
5	2	3	1	0	4	5

When computing traces for all elements we get the following results:

$$t(0) = \{0 = 0,\ 1 = 0 * 0\}; \qquad t(1) = \{1 = 1\};$$

$$t(2) = \{2 = 2,\ 3 = 2 * 2\}; \qquad t(3) = \{3 = 3\};$$

$$t(4) = \{4 = 4,\ 5 = 4 * 4\}; \qquad t(5) = \{5 = 5\}.$$

Clearly none of the traces alone yields all elements of the quasigroup and therefore the first condition in Algorithm 2 is not applicable. Using the second condition we can simply combine traces for elements $0, 2, 4$ *thereby obtaining the generating system:*

$$G_1 = \langle \{0, 2, 4\} | \{1 = 0 * 0,\ 3 = 2 * 2,\ 5 = 4 * 4\} \rangle.$$

While this is sufficient, we can do better in terms of the number of generators used, by using the third condition. Suppose we combine traces $t(0)$ *and* $t(2)$*, and compute possible combinations for generators* $0, 2$*, then we get the generating system:*

$$G_1 = \langle \{0, 2\} | \{1 = 0 * 0,\ 3 = 2 * 2,\ 4 = 0 * 2,\ 5 = 0 * (2 * 2)\} \rangle.$$

6.2.2 Expanding Generating Systems

Generating systems can be computed in a number of ways as described in the previous section. However, the prerequisite is generally to have the Cayley table of the quasigroup available already. Since this would be contrary to our goal of exploiting them for model generation we evolve generating systems computed from small size quasigroups to generating systems sufficient for larger size quasigroups. This can essentially be achieved in two different ways:

(a) adding a new element as a generator, or

(b) expressing the new element as a relation in the existing generators.

Let $(Q, *)$ be a quasigroup of size n, i.e., $Q = \{0, \ldots, n-1\}$, with generating system $G = \langle S|R\rangle$. Then we can obtain a generating system G' by either one of the two steps:

(i) $G = \langle S \cup \{n\}|R \cup \{n = n\}\rangle$

(ii) $G = \langle S|R \cup \{n = w(s_1, \ldots, s_k)\}\rangle$, where $s_1, \ldots, s_k \in S$.

Since step (i) does not add any structural knowledge we concentrate on step (ii) to extend generating systems of a quasigroup of size n to one for a size $n+1$ quasigroup. Here we have a choice of generators to use as well as structure of the added relation, which can give us a list of generating systems formed each using a different generator. In addition we require that the newly created relations must be distinct and not already present in the original generating system.
For example, the input generating system
$G_3 = \langle\{1, 3\}|\{2 = 3 * 1,\ 0 = 3 * (3 * 1)\}\rangle$ for a quasigroup of size 4 can be expanded in a number of ways to represent a quasigroup of size 5,

resulting in different generating systems, a few of them are given as follows:

$$G_4 = \langle\{1,3\}|\{2 = 3{*}1,\ 0 = 3{*}(3{*}1),\ 4 = 1{*}(3{*}(3{*}1))\}\rangle$$
$$G_5 = \langle\{1,3\}|\{2 = 3{*}1,\ 0 = 3{*}(3{*}1),\ 4 = 3{*}(3{*}(3{*}1))\}\rangle$$
$$G_6 = \langle\{1,3\}|\{2 = 3{*}1,\ 0 = 3{*}(3{*}1),\ 4 = (3{*}1){*}(3{*}1)\}\rangle$$

6.2.2.1 Applying Quasigroup Element Filter to Generating System Expansion

For expanding generating systems to represent quasigroups of larger sizes we employ the quasigroup element filter described in Section 6.1 for verifying the computed relations. Figure 6.4 shows how we employ this filtering to verify the expanded generating systems. Every time a relation is added, we use a symbolic verification function to check that all the relations in the set are unique and that the Latin square property is not violated. We continue this process until we obtain the set R of filtered relations that is of a particular pre-defined size. Generally, we specify the size of R as a multiple of the size n of the quasigroup Q. In the second filter step we then check for the entire set R that its elements i.e. the relations do not violate the desired quasigroup property P — which can in general be a combination of properties — using an automated theorem prover Prover9.

Let us look at an example of how the quasigroup element filtering is used to expand the input generating system

$G_3 = \langle\{1,3\}|\{2 = 3*1,\ 0 = 3*(3*1)\}\rangle$ for a quasigroup of size 4 to represent a quasigroup of size 5. *Generating system relation generator* is given G_3 as input and computes a new relation $4 = 1{*}(3{*}(3{*}1))$. The newly computed relation is added to the set of relations of G_3 giving us

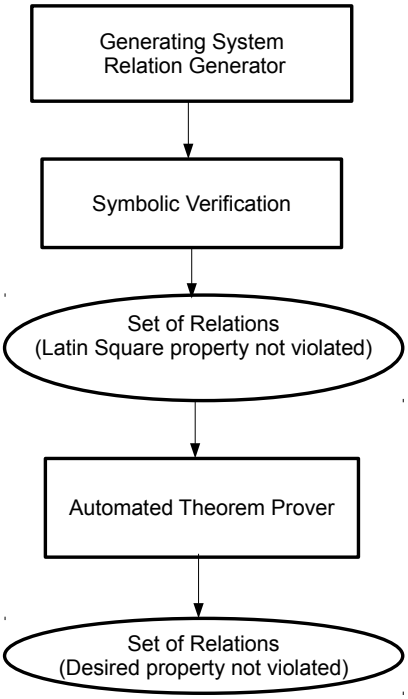


Figure 6.4: Flowchart of the filtering approach applied to generating system evolution

R=$\{2 = 3{*}1,\ 0 = 3{*}(3{*}1),\ 4 = 1{*}(3{*}(3{*}1))\}$. *Symbolic verification* procedure is then used to verify that the set of relations R that represent the quasigroup elements do not violate the Latin Square property, in which case an *Automated theorem prover* is used to verify that R does not violate the desired quasigroup property P. After successfully passing both filtering applications R is selected as the new set of relations. This gives us the generating system
$G_4 = \langle\{1, 3\}|\{2 = 3{*}1,\ 0 = 3{*}(3{*}1),\ 4 = 1{*}(3{*}(3{*}1))\}\rangle$ that represents a quasigroup of size 5.

CHAPTER 7

EXPERIMENTS AND RESULTS

Chapter Overview: *In this chapter we present the experimental set-up of our combined reasoning system for computing large solution models of quasigroups with some interesting properties. We use different model generation systems and knowledge pre-computation techniques to perform experiments for an effective evaluation of the systems and knowledge based techniques.*

In this chapter, we present the experimental set-up of our combined reasoning system to compute large size solution models of quasigroups with some interesting non-trivial two-variable properties that are defined in Chapter 4. We provide a standard comparison of systems for generating solution models and also use additional knowledge that is generated using our pre-computation techniques defined in Chapter 6 and implied constraints that are defined in Chapter 4. We perform experiments to demonstrate the benefits of using this additional

knowledge to push the boundaries of model generation systems to compute large size solution models that they are unable to find without this knowledge. We present an analysis of results by comparing the techniques that were useful for particular properties of quasigroups.

7.1 Experimental Set-up

We now present a description of the system combination that serves as a preprocessor to compute the additional constraints for the approaches presented in Chapter 6. The primary systems we employ and the particular settings we have used with respect to the problem encodings are given as follows:

Constraint Solver: We use Minion [GJM06], together with the primal model that we have described in Chapter 5 for specifying the quasigroup CSP. We use the essence modelling language [FGJ+07] to model the problem and tailor [Ren] to translate it into an efficient problem encoding for Minion.

Finite Model Generator: We use Mace4 [McC03a] as model generator. In addition to the equational encoding of the quasigroup and its property, we explicitly specify the size of the quasigroup by adding appropriate equalities as described in Chapter 5. This choice has proved superior to parametrically specifying the domain size in preliminary experiments.

SAT solver: We use MiniSat [ES03] and zChaff [FMM] SAT solvers with the extended encoding that is described in Chapter 5. We have extended Stickel's quasigroup representation generator [Sti] to be used with our approaches, that generates quasigroup SAT problem encodings in the DIMACS CNF format which is a

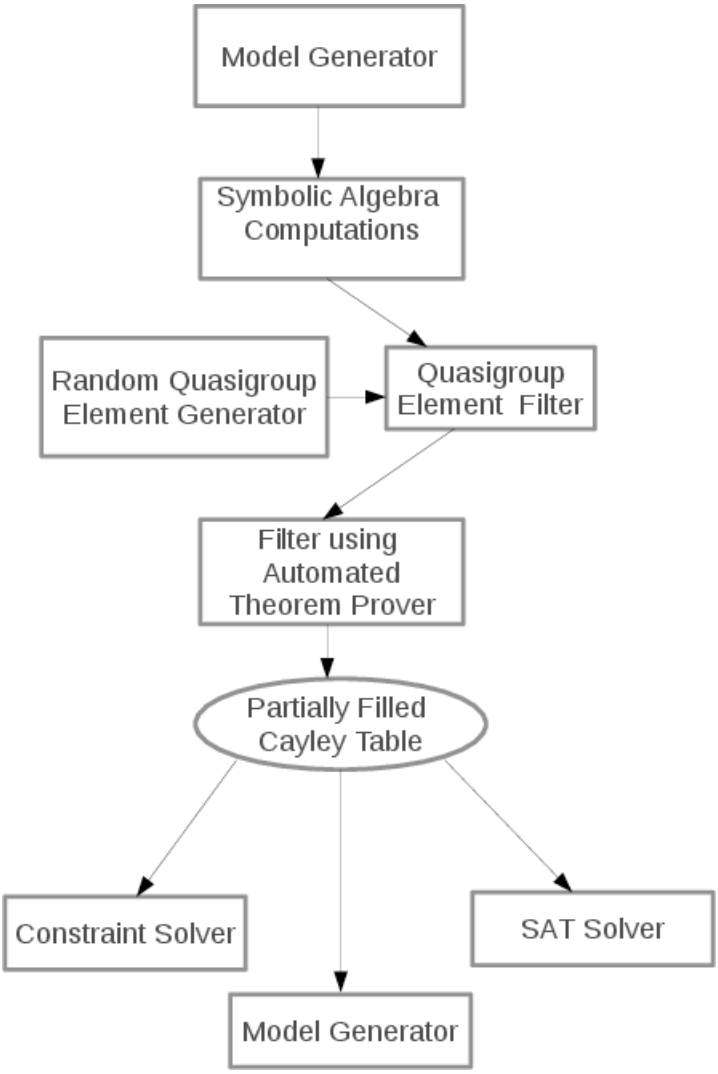


Figure 7.1: Model for the Combined Approach

standard way to represent conjunctive normal form Boolean formulae.

For pre-computing additional constraints according to the techniques presented in Chapter 5, we combine a number of reasoning engines in a preprocessing unit. It integrates a mix of bespoke algorithms we have implemented and off the shelf tools. Fig. 7.1 presents a structural diagram of the basic work flow we have assembled with the following components:

Model generator Mace4 [McC03a] together with Isofilter is employed

to compute small size non-isomorphic quasigroups.

Symbolic Algebra Computations is an Ocaml procedure written by us that implements the techniques of computing and evolving generating systems that are described in Chapter 6.

Random Quasigroup Element Generator is an Ocaml procedure written by us that computes random triples of numbers representing elements of a quasigroup.

Quasigroup Element Filter is the validation function written by us which is also implemented in Ocaml, that checks for a single generated element that they do not violate the Latin square property.

Automated theorem prover Prover9 [McC] is used for verifying that a set of pre-assigned elements in the quasigroup Cayley table does not violate any of the properties that a quasigroup should exhibit. A constraint solver can also be used for the same purpose.

Model Generation Systems To finally generate the quasigroup model we use various systems that are listed in the beginning of this chapter.

These systems are used in slightly varying work flows to construct three different algebraically restricted encodings that are defined as follows.

7.1.1 Quasigroup Element Filtering Procedure

This procedure corresponds to the quasigroup element filtering approach discussed in Chapter 6. Let P be the desired property of our quasigroup of size n to be generated. In addition, let k be the number of random elements we want to preset. In general we let k be defined in terms of n

(i.e., $n, n + \lceil \frac{n}{2} \rceil, 2n, 2n + \lceil \frac{n}{2} \rceil$) to identify the maximum number of pre-computed elements after which pre-computing makes the problem unsatisfiable. The procedure is defined as follows:

1. Use our quasigroup element generator to compute random quasigroup elements for the set S of triples.

2. Filter each element with the simple filter to verify that they do not violate the Latin square property.

3. Once k elements are computed, verify with Prover9 that the set S of elements does not violate property P. Prover9 is assigned 10 seconds to find the proof.

4. If no proof can be found, fix the final set S_F of verified elements. Otherwise generate and filter more elements.

Encoding 1 consists of:

- the Latin Square property which is given in Definition 4.1.1 in Chapter 4,

- the specific quasigroup property P i.e., one of the two-variable properties given in Definition 4.2.2 in Chapter 4,

- a skeleton Cayley table given by S_F (set of pre-computed elements).

7.1.2 Generating System Procedure

This procedure corresponds to the generating system approach discussed in Chapter 6. In our experiments, we have concentrated on evolving generating systems by adding novel relations only. Assume that we want to construct a quasigroup of size n that exhibits a particular property P. Then the procedure works as follows:

1. Generate up to 10 (some have only 1 or 2) non-isomorphic quasigroups of size 5 with property P using Mace4. Note, if Mace4 is unable to find quasigroups of size 5 then we take quasigroups of size 4.

2. Compute the generating systems G for the small size quasigroups using the symbolic algebra computations.

3. Stepwise evolve each generating system to a generating system G' that represents a quasigroup of size n. At each step use the quasigroup element filter to verify that the elements generated by the evolved generating system do not violate the Latin square property and indeed represent a new unique element.

4. Once generating system G' has size n, verify with Prover9 that it does not violate property P. Prover9 is assigned 10 seconds to find the proof.

5. If no proof can be found, G' is the final generating system. Otherwise loop through the other evolved generating systems.

Encoding 2 consists of:

- the Latin Square property which is given in Definition 4.1.1 in Chapter 4,

- the specific quasigroup property P i.e., one of the two-variable properties given in Definition 4.2.2 in Chapter 4,

- a skeleton Cayley table that can be computed using G', by using the relations that represent the pre-computed elements.

7.1.3 Combination of Both Procedures

Finally we define a procedure as a mix of goal directed computation with generating systems and random generation of elements by combining both procedures. The procedure is defined as follows:

1. Use the generating system approach to get the evolved generating systems. This yields a set S of n quasigroup elements.

2. Compute a set S' of k elements using the quasigroup element filtering approach, verifying that none of the elements in S' coincides with elements in S.

3. As k is usually defined in terms of n, the union $S \cup S'$ has now $n + \left\lceil \frac{n}{2} \right\rceil, 2n, 2n + \left\lceil \frac{n}{2} \right\rceil, \ldots$ elements.

Encoding 3 consists of:

- the Latin Square property which is given in Definition 4.1.1 in Chapter 4,

- the specific quasigroup property P i.e., one of the two-variable properties given in Definition 4.2.2 in Chapter 4,

- a skeleton Cayley table given by $S \cup S'$ (set of pre-computed elements).

7.1.4 Employing Implied Constraints

In addition to our element pre-computation approaches, we have also performed experiments using implied constraints that are defined in Chapter 4. We have extended the work done by [CM01, CCM06] where the final model generation is done by Choco [Lab00] constraint solver and quasigroups having only two properties Q-1 and Q-2 with different

implied constraints were generated. We have extended the quasigroup problem encodings to include the additional axioms or constraints: C_1, C_2, C_3, C_4, C_5 with the properties Qg-1, Qg-2, Qg-3, Qg-4, Qg-5, Qg-6, Qg-7, and also C_6 with Q-5. Moreover, we have used three different types of automated reasoning systems i.e. constraint solver (Minion), model generator (Mace4) and SAT solvers (MiniSat and zChaff) for the final model generation.

7.2 Discussion of Results

We have run our experimental set-up to generate quasigroups with properties: Qg-1, Qg-2, Qg-3, Qg-4, Qg-5, Qg-6 and Qg-7 that are defined in Chapter 4. In our experiments, the systems were restricted to construct a single solution for size 3 to at most 25 each (there are possible restrictions on the lowest size as discussed below) and to a time limit of 2 hours for the generation of each quasigroup. Furthermore, the experiments with pre-computation techniques are repeated multiple times (at most 10 times) for each quasigroup property of a particular size.
The results for the direct system comparison without preset elements are given in Table A.1 of Appendix A. The results are presented giving one major row per property, further broken down into the different sizes where quasigroups could be generated. The columns display the CPU time in seconds for the quasigroups that were found, using the four different systems that we have compared. Dashes as entries indicate that a particular system was not able to find a quasigroup of that property and size. If a row is missing for a particular size then none of the systems could find a corresponding quasigroup.
The analysis of the results obtained by each system, show that MiniSat generally outperformed all other systems on nearly all properties, but in

particular for properties Qg-4 and Qg-5. Only zChaff is slightly better for property Qg-6 where it finds a model of size 17. While in general the SAT solvers seem to be performing better than the other two systems, there are some peculiar phenomena to observe: For Qg-5 zChaff finds models for nearly all the same sizes as MiniSat, however, it appears to be less reliable, not finding a model for size 13, while nevertheless finding some of higher sizes. Minion finds a Qg-1 model of size 13 where none of the other systems finds one, while missing the one of size 12.

Table A.2 in Appendix A gives the results for our element pre-computation approaches with the three different encodings described in Section 7.1. Here we have a major column for each encoding that are further parametrized with respect to the number of pre-computed elements, where n is the size of the quasigroups. For Encoding 1 we have experimented with n and $n + \lceil \frac{n}{2} \rceil$ pre-computed elements respectively and for Encoding 3 we have experimented with $n + \lceil \frac{n}{2} \rceil$ elements. For both encodings with $n + \lceil \frac{n}{2} \rceil$ precomputed elements we have only experimented with sizes 10 to 25 and for Encoding 2 with sizes 6 to 25, as the smaller sizes would have been meaningless. The former because the precomputed elements nearly filled the entire table and for the latter as we already used models of size 4 or 5, respectively, to evolve generating systems. This is denoted by n/a in the table. Again, dashes indicate that a system could not find a quasigroup and rows are omitted where no system could find an instance. Furthermore, it should be noted that for Encoding 3 neither SAT solver produced any results and therefore these columns were omitted. Also, the element pre-computation times for all three encodings are negligible.

A comparison of Table A.1 and Table A.2 in Appendix A, shows that our techniques have increased the solvability horizon of Mace4, Minion,

MiniSat and zChaff for some properties, notably Qg-1, Qg-2, Qg-3 and Qg-4. In particular, for the Qg-2 property Mace4, found a size 16 quasigroup with encoding 3 that all other systems were unable to find. Similarly, a Qg-3 quasigroup of size 13 was found by Mace4 with all three encodings, by MiniSat with encoding 2 and by zChaff with encodings 1 and 2. A Qg-4 quasigroup of size 19 was found by zChaff with encoding 2, that all other systems were unable to find. However, there is no clear winning strategy for our encodings, in fact all have their strengths and their weaknesses.

Since the pre-computation techniques are incomplete, it is probably not surprising that for some quasigroup properties and sizes no solution models were found. Moreover, for some properties there exist only 1 or 2 non-isomorphic quasigroups for small sizes, and that gives us less structural information to work with and evolve further.

The results with implied constraints are also presented in Appendix A in Tables A.3, A.4, A.5, A.6, A.7, A.8, A.9. The results are presented in a different table for each property, having a row for different sizes where quasigroups could be generated. The columns display the size of the quasigroup, systems used and the different constraints used. Each cell displays the CPU time in seconds for the quasigroups that were found, using the four different systems that we have compared. Dashes as entries indicate that a particular system was not able to find a quasigroup of that property and size. If a row is missing for a particular size then none of the systems could find a corresponding quasigroup.

A comparison of the results with implied constraints and pre-computation techniques shows that implied constraints narrowed down the search for Qg-4 quasigroups enabling the construction of an example of size 19, as was previously found by the pre-computation

techniques. Qg-5 quasigroup of size 20 was found by MiniSat using constraints C_6 and Qg-6 of size 21 was found by Minion using constraint C_4 and C_5 each, and by MiniSat using constraint C_5 that we were unable to find previously. Although implied constraints helped to find larger sizes for Qg-5 and Qg-6 quasigroups, they did not help in computing larger sizes for Qg-2 and Qg-3 quasigroups as was the case with pre-computation techniques. A summary of results using all approaches is presented in Table 7.1, where $\surd$ denotes that a model was found and $\times$ denotes that a model was not found for the quasigroup property of the given size.

Property	Size	Standard	Pre-Computations	Implied Constraints
Qg-1	4	√	√	√
	8	√	√	√
	9	√	√	√
	12	√	√	√
	13	√	√	√
Qg-2	4	√	n/a	√
	5	√	√	√
	8	√	√	√
	9	√	√	√
	12	√	√	√
	13	√	√	√
	16	×	√	×
Qg-3	3	√	√	√
	4	√	√	√
	7	√	√	√
	9	√	√	√
	10	√	√	√
	12	√	√	√
	13	×	√	×
Qg-4	4	√	√	√
	5	√	√	√
	9	√	√	√
	11	√	√	√
	13	√	×	√
	16	√	×	√
	17	√	×	√
	19	×	√	√
	20	√	×	√
	21	√	×	√
Qg-5	3	√	√	√
	4	√	√	√
	5	√	√	√
	7	√	×	√
	8	√	√	√
	9	√	×	√
	11	√	×	√
	12	√	×	√
	13	√	×	√
	15	√	×	√
	16	√	×	√
	17	√	×	×
	20	√	×	√
Qg-6	5	√	√	√
	9	√	√	√
	13	√	×	√
	17	√	×	×
	21	×	×	√
Qg-7	4	√	√	√
	7	×	√	×
	8	√	√	√
	9	√	×	√
	13	√	×	√
	16	√	√	√

Table 7.1: Summary table for quasigroup results.

Part III

Approximating Solutions in Infinite Domains

CHAPTER 8

BACKGROUND ON POINT-SET TOPOLOGY AND KURATOWSKI CLOSURE-COMPLEMENT PROBLEM

Chapter Overview: *This chapter describes the basic notions of point-set topology and introduces the Kuratowski's closure-complement problem. To provide the necessary background for the following chapters we first define and give examples of the basic concepts used in point-set topology such as: topology, topological space, closed set, open set, closure, interior, exterior and boundary. We then present the theorem for the Kuratowski closure-complement problem along with a proof, which is followed by the formal definition of the problem.*

In the previous part we pushed the boundaries of model finding in finite algebra by computing additional knowledge via automated theory exploration using symbolic computations and automated theorem proving. In this part, we aim to find approximate solutions to problems

that can be of infinite nature by computing and presenting results in a fashion that allows the user to explore the knowledge to obtain valuable insights into the structure of solutions and to guide computations via providing feedback. The motivation for the development of this approach is to solve the generalization of Kuratowski's closure-complement problem that is described in detail in Chapter 9.

In this chapter, we give the relevant background on the basic terms used in point-set topology and introduce Kuratowski's classical closure-complement problem. The definitions, propositions, theorems, proofs and examples presented in this chapter are taken from the literature and have been appropriately referenced.

8.1 Basic Concepts in Point-Set Topology

Following [Kel75], [Mun00] and [Lip65, p. 66-70] we define the basic concepts in point-set topology in the following section.

The term topology can be defined as follows:

Definition 8.1.1 (Topology) *Let X be a non-empty set. A collection $\mathcal{T}$ of subsets of X is called a topology if it satisfies the following three conditions:*

(i) $\emptyset \in \mathcal{T}$ *and* $X \in \mathcal{T}$.

(ii) The union of an arbitrary collection of sets in $\mathcal{T}$ is also in $\mathcal{T}$, or equivalently , if $\{U_i : i \in I\}$ is a collection such that $U_i \in \mathcal{T}$ for each $i \in I$, then $(\cup_{i \in I} U_i) \in \mathcal{T}$.

(iii) The intersection of a finite collection of sets in $\mathcal{T}$ is also in $\mathcal{T}$, or equivalently if $\{U_i : i \in I\}$ is a collection such that $U_i \in \mathcal{T}$ for each $i \in I$, then $(\cap_{i \in I} U_i) \in \mathcal{T}$.

The definition of topology can be further understood more clearly by the following example.

Example 8.1.1 *Let $X = \{x, y, z\}$. The collection $\mathcal{T} = \{\emptyset, X, \{x\}, \{y\}, \{x, y\}, \{y, z\}\}$ is a topology on X. It can be pictured as shown in Figure 8.1 where the ovals represent the open sets.*

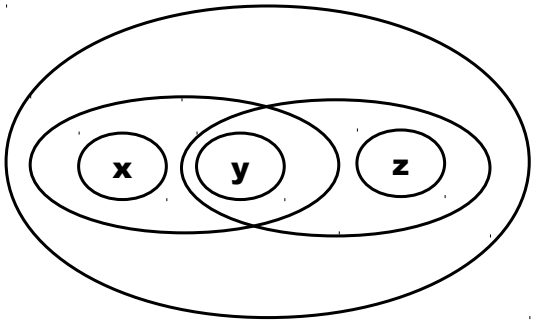


Figure 8.1: Topology Example [Bro10]

We now define a topological space as follows.

Definition 8.1.2 (Topological space) *Let X be a non-empty set, and $\mathcal{T}$ a topology for X. The pair $(X, \mathcal{T})$ is called a topological space.*

An element x of a topological space X is usually referred as a *point* of X. Whenever it makes no confusion, we denote a topological space $(X, \mathcal{T})$ by its underlying set X.
Open and closed sets in a topological space can be defined as follows.

Definition 8.1.3 (Open set) *Let $(X, \mathcal{T})$ be a topological space. A set $U \in \mathcal{T}$ is called an open set.*

Definition 8.1.4 (Closed set) *Let X be a topological space. A set $A \subset X$ whose complement A' is open is called a closed set.*

Closed sets have the following properties:

(i) $\emptyset$ and X are closed sets.

(ii) The intersection of closed sets in X is closed.

(iii) Any finite union of closed sets in X is closed.

Example 8.1.2 *The class* $\mathcal{T} = \{X, \emptyset, \{a\}, \{c, d\}, \{a, c, d\}, \{b, c, d, e\}\}$ *defines a topology on* $X = \{a, b, c, d, e\}$. *The closed subsets of* X *are*

$$\emptyset, X, \{b, c, d, e\}, \{a, b, e\}, \{b, e\}, \{a\}$$

that is, the complements of the open subsets of X. *Note that there are subsets of* X *such as,* $\{b, c, d, e\}$ *which are both open and closed, and there are subsets of* X, *such as* $\{a, b\}$, *which are neither open nor closed.*

$A'' = A$, for any subset A of a topological space X. Hence:

Proposition 8.1.1 *In a topological space* X, *a subset* A *of* X *is open if and only if its complement is closed.*

The terms closure, interior, exterior and boundary in topological spaces are defined below with some relevant examples.

Definition 8.1.5 (Closure in topological spaces) *Let* X *be a topological space, and* $A \subset X$. *The closure of* A, *denoted by* $Cl(A)$ *or* $\bar{A}$, *is defined by* $Cl(A) = \bigcap_i G_i$, *where* $A \subset G_i$ *and* G_i *is a closed set of* X.

Proposition 8.1.2 *Let* $\bar{A}$ *be the closure of* A, *then the following hold:*

(i) $\bar{A}$ *is closed.*

(ii) If F *is a closed superset of* A, *then* $A \subset \bar{A} \subset F$.

(iii) A *is closed if and only if* $A = \bar{A}$.

Example 8.1.3 *Consider the topology $\mathcal{T}$ on $X = \{a, b, c, d, e\}$ of Example 8.1.2 where the closed subsets of X are*

$$\emptyset,\ X,\ \{b, c, d, e\},\ \{a, b, e\},\ \{b, e\},\ \{a\}$$

Accordingly,

$$\overline{\{b\}} = \{b, e\},\ \overline{\{a, c\}} = X,\ \overline{\{b, d\}} = \{b, c, d, e\}$$

Definition 8.1.6 (Interior in topological spaces) *Let X be a topological space, and $A \subset X$. The interior of A, denoted by $Int(A)$, is the open set defined by $Int(A) = \bigcup_i G_i$, where $G_i \subset A$ and G_i is an open set of X. Any point $x \in Int(A)$ is called an interior point of A.*

The interior of A can also be characterized as follows:

Proposition 8.1.3 *The interior of a set A is the union of all open subsets of A. Furthermore:*

(i) Int(A) is open.

(ii) Int(A) is the largest open subset of A, i.e. if G is an open subset of A then $G \subset Int(A) \subset A$.

(iii) A is open if and only if $A = Int(A)$.

Definition 8.1.7 (Exterior in topological spaces) *The exterior of A, written as $Ext(A)$, is the interior of the complement of A, that is,*

$$Ext(A) = Int(A')$$

Definition 8.1.8 (Boundary in topological spaces) *Let X be a topological space, and $A \subset X$. The boundary of A, denoted by $Bd(A)$, is*

the closed set defined by $Bd(A) = Cl(A) \cap Cl(A')$. *Any point* $x \in Bd(A)$ *is called a boundary point of* A.

The following theorem depicts an important relationship between interior, boundary and closure.

Theorem 8.1.1 *Let* A *be any subset of a topological space* X. *Then the closure of* A *is the union of the interior and boundary of* A, *i.e.* $\bar{A} = Int(A) \cup Bd(A)$

Example 8.1.4 *Consider the four intervals of real numbers* $[a, b]$, (a, b), $(a, b]$ *and* $[a, b)$ *whose endpoints are* a *and* b. *The interior of each is the open interval* (a, b) *and the boundary of each is the set of endpoints, i.e.* $\{a, b\}$

Example 8.1.5 *Consider the topology*

$$\mathcal{T} = \{X, \emptyset, \{a\}, \{c, d\}, \{a, c, d\}, \{b, c, d, e\}\}$$

on $X = \{a, b, c, d, e\}$, *and the subset* $A = \{b, c, d\}$ *of* X. *The points* c *and* d *are each interior points of* A *since*

$$c, d \in \{c, d\} \subset A$$

where $\{c, d\}$ *is an open set. The point* $b \in A$ *is not an interior point of* A; *and since Int(A) is the largest open subset of* A, *therefore,* $Int(A) = \{c, d\}$. *Only the point* $a \in X$ *is exterior to* A, *i.e. interior to the complement* $A' = \{a, e\}$ *of* A, *hence* $Int(A') = \{a\}$. *Accordingly the boundary of* A *consists of the points* b *and* e, *i.e.* $Bd(A) = \{b, e\}$ *which can be computed as follows:*

$$Bd(A) = Cl(A) \cap Cl(A') = \{b, c, d, e\} \cap \{a, b, e\} = \{b, e\}$$

8.2 Kuratowski Closure-Complement Problem

In 1922 Kuratowski proposed the question on an arbitrary topological space, of how many different combinations of the operators of complement and closure exist? This problem was solved by Kuratowski, and the solution consists of 14 combinations. Following [GJ08], the Kuratowski closure-complement theorem and its proof is given as follows.

Theorem 8.2.1 (The Kuratowski Closure-Complement Theorem) *If $(X, \mathcal{T})$ is a topological space and $A \subseteq X$ then at most 14 sets can be obtained from A by taking closures and complements.*

Proof of the Theorem:

Let $(X, \mathcal{T})$ be a topological space and consider the complement operator a acting on the set of subsets of X defined by $a(A) = X \setminus A$ and the closure operator b defined by $b(A) = \bar{A}$. Consider symbol i to denote the interior of a set which is defined by $i(A) = a(b(a(A)))$. Furthermore, the complement operator a satisfies $a^2 = id$, where id is the identity operator, and the closure operator b is idempotent i.e. $b^2 = b$. This immediately shows that every operator in the operator monoid generated by a and b is either the identity operator id or is equal to one of the form $abab...ba, baba...ba, abab...ab$, or $baba...ab$. First we will show that the operators bab and $bababab$ are identical. This gives the upper bound of 14 since the only remaining operators are:

$id,\ a,\ b,\ ab,\ ba,\ aba,\ bab,\ abab,\ baba,\ ababa,\ babab,\ ababab,\ bababa,\ abababa$

Note that $bab \geq ababab$ since $ababab(A)$ is the interior of $bab(A)$. Since b is idempotent, it then follows that $bab = bbab \geq bababab$. On the other

hand, $abab \leq b$, since $abab(A)$ is the interior of $b(A)$ and therefore, $babab \leq bb = b$. Also, since $babab \leq b$ is equivalent to $ababab \geq ab$, therefore $bababab \geq bab$. Combining these two inequalities gives $bab = bababab$ as required.

To complete the proof it suffices to find a topological space with a subset for which each of the 14 possible operators produces a different set.

To complete the proof we present an example to show that the set $A \subseteq \mathbb{R}$ given by:

$$A = (0,1) \cup (1,2) \cup \{3\} \cup ([4,5] \cap \mathbb{Q})$$

attains the bound of 14 i.e. we can produce 14 distinct sets from A by taking complements and closures. These sets are:

$$id(A) = A = (0,1) \cup (1,2) \cup \{3\} \cup ([4,5] \cap \mathbb{Q})$$

$$a(A) = (-\infty, 0] \cup \{1\} \cup [2,3) \cup (3,4) \cup ([4,5] \cap \mathbb{I}) \cup (5, -\infty)$$

$$b(A) = [0,2] \cup \{3\} \cup [4,5]$$

$$ab(A) = (-\infty, 0) \cup (2,3) \cup (3,4) \cup (5, -\infty)$$

$$ba(A) = (-\infty, 0] \cup \{1\} \cup [2, -\infty)$$

$$aba(A) = (0,1) \cup (1,2)$$

$$bab(A) = (-\infty, 0] \cup [2,4] \cup [5, -\infty)$$

$$abab(A) = (0,2) \cup (4,5)$$

$$baba(A) = [0,2]$$

$$ababa(A) = (-\infty, 0) \cup (2, -\infty)$$

$$babab(A) = [0, 2] \cup [4, 5]$$

$$ababab(A) = (-\infty, 0) \cup (2, 4) \cup (5, -\infty)$$

$$bababa(A) = (-\infty, 0] \cup [2, -\infty)$$

$$abababa(A) = (0, 2)$$

Following [GJ08], the solution for the Kuratowski closure-complement problem i.e. how many different combinations of the operators of complement and closure exist is defined as follows:

Definition 8.2.1 *Let $(X, \mathcal{T})$ be a topological space and $A \subseteq X$.*

(i) $k(A)$ *(the k-number of A) denotes the number of distinct sets obtainable from A by taking closures and complementation. A set with k-number n will also be called an n-set.*

(ii) $k((X, \mathcal{T}))$ *(the k-number of* $(X, \mathcal{T})$*) denotes* $max\{k(A) : A \subseteq X\}$.

(iii) $K((X, \mathcal{T}))$ *(the K-number of* $(X, \mathcal{T})$*) denotes the number of distinct Kuratowski closure and complement operators on* (X, T).

The Kuratowski result firstly shows that the k-number of any set in a topological space is at most 14 and secondly the K-number of any topological space is at most 14. The proof of the Kuratowski closure-complement theorem first showed that $K((X, \mathcal{T})) \leq 14$ and then that the k-number of the reals with the usual topology is actually 14. Hence, we can say that $K((X, \mathcal{T})) \geq k((X, \mathcal{T}))$.

CHAPTER 9

GENERALIZATION OF KURATOWSKI PROBLEM TO POINT FREE TOPOLOGY

Chapter Overview: *This chapter describes the generalization of Kuratowski problem (i.e. how many different combinations of the operators of complement and closure exist?) to point free topology using the inference rules of intutionistic logic. We define the inference rules, the operators used, the generalization of the problem and finally the n^{th} approximation to the problem.*

Kuratowski's classical closure-complement problem was proposed and solved by him in 1922. The problem and the solution to the problem is defined in Section 8.2 in Chapter 8. The problem has been generalized in many different ways to consider other operators, such as union or intersection as given in [GJ08], or slightly different settings, such as point free topology (locale theory) as given in [WY00]. The solution to a generalized version could be a significantly larger number of

$$\frac{}{x \leq x}(\texttt{reflexive}) \qquad \frac{x \leq y \quad y \leq z}{x \leq z}(\texttt{transitive})$$

$$\frac{x \leq y \quad y \leq x}{x = y}(\texttt{antisymmetric}) \qquad \frac{x \leq y}{-y \leq -x}(\texttt{antimonotone})$$

$$\frac{}{x \leq - - x}(\texttt{saturates}) \qquad \frac{}{- - -x = -x}(\texttt{quasi-idempotent})$$

$$\frac{}{i(x) \leq x}(\texttt{reduces}) \qquad \frac{x \leq y}{i(x) \leq i(y)}(\texttt{monotone-i})$$

$$\frac{}{i(x) = i(i(x))}(\texttt{idempotent-i}) \qquad \frac{}{x \leq c(x)}(\texttt{saturates})$$

$$\frac{x \leq y}{c(x) \leq c(y)}(\texttt{monotone-c}) \qquad \frac{}{c(x) = c(c(x))}(\texttt{idempotent-c})$$

$$\frac{}{c(-x) \leq -i(x)}(\texttt{compatible-1}) \qquad \frac{}{i(-x) \leq -c(x)}(\texttt{compatible-2})$$

Table 9.1: Axioms for the generalized Kuratowski problem.

combinations, or a proof that infinitely many combinations exist. In this chapter, we first define the axioms or inference rules for the generalized Kuratowski problem. Note that these rules are intutionistic, but this is of no concern in the following. We then define the operators used in the inference rules such as interior, closure and complement; as well as the subset relation $\leq$. This is followed by the definition of the generalization of the Kuratowski closure-complement problem. Furthermore, we define the n^{th} approximation to the problem, approximating graph of order n and the theorems for finding finite and infinite solutions to the problem.

9.1 The Problem

The generalization of the point-free version of the Kuratowski problem is introduced by Sambin in [Sam03]. The problem has been previously studied in [Cor06] but remained open and only minimal progress towards

a solution was achieved. The generalization is obtained by introducing a partial order relation $\leq$ — that captures the inclusion relation for subsets — and relaxing the axioms for the operators as given in Table 9.1 in a rule format, where i denotes the interior operator, c denotes the closure operator and $-$ denotes the complement operator. It must be noted that the relaxed axiomatization, effectively turns i into a reduction operator, c into a saturation operator, and $-$ into a pseudo-complement as defined in Table 9.1. It can be observed that the two compatibility requirements are reminiscent of the classical equation $i(x) = -c(-x)$ (dually $c(x) = -i(-x)$). Indeed, if we define $i(x)$ as $-c(-x)$ and we also assume $--x = x$ for all x, then both compatibility axioms can be derived.
An example model for the axioms can be obtained by combining the definitions of interior, closure and complement with the rules of intuitionistic logic. The interior, closure and complement operators and subset relation $\leq$ are defined as follows.

Definition 9.1.1 (Interior) *Given a topological space $(P, \mathcal{O})$, the interior of a set x is defined as*

$$\{\alpha \mid \exists y \in \mathcal{O},\ \alpha \in y \wedge \forall \beta \in y.\beta \in x\}$$

Definition 9.1.2 (Closure) *Given a topological space $(P, \mathcal{O})$, the closure of x that avoids any reference to negation is defined as*

$$\{\alpha \mid \forall y \in \mathcal{O},\ \alpha \in y \Rightarrow \exists \beta \in y.\beta \in x\}$$

(the set of all accumulation points of x).

Definition 9.1.3 (Complement) *Given a topological space $(P, \mathcal{O})$, the*

complement of x is defined as

$$\{\alpha \in P \mid \neg(\alpha \in x)\}$$

and it hides a negation.

Definition 9.1.4 (Subset relation $\leq$) *Given a topological space $(P, \mathcal{O})$, the subset relation ($\leq$) which satisfies the axioms in Table 9.1 hides implication: $x \leq y$ iff*

$$\forall \alpha,\ \alpha \in x \Rightarrow \alpha \in y$$

The inference rules or axioms presented in Table 9.1 are obtained from the properties of negation and the quantifiers in intuitionistic logic. For instance, from the intuitionistic principle $A \Rightarrow \neg\neg A$ we obtain $x \leq --x$ and from the DeMorgan laws for quantifiers we obtain the two compatibility relations: For example, $\forall\exists\neg \Rightarrow \neg\exists\forall$ becomes $c(-x) \leq -i(x)$.

We are interested in applying a number of different combinations of operators to any subset of a topological space to generate distinct sets. Consequently, we define the generalized Kuratowski problem in terms of equivalent operator combinations.

Definition 9.1.5 (Generalized Kuratowski closure-complement problem) *Let $(P, \leq)$ be any partially ordered set and let $\{i, c, -\}$ be the set of operators on P axiomatized as in Table 9.1. Let $S = \{i, c, -\}^*$ be the set of all words over the operators (i.e., all possible finite combinations). We define the order relation $\leq$ on S for all $w_1, w_2 \in S$ by: $w_1 \leq w_2$ iff $w_1(x) \leq w_2(x)$ for all $x \in P$. Finally, let $\equiv$ over S be the symmetric closure of $\leq$, if $w_1 \leq w_2$ and $w_2 \leq w_1$, then $w_1 \equiv w_2$. The generalized*

Kuratowski closure-complement problem then consists in computing the cardinality of $S_{/\equiv}$, the set of equivalence classes of S modulo $\equiv$.

Furthermore, we define the canonical representative of an equivalence class $[w]_{/\equiv} \in S_{/\equiv}$ as the minimum element of the set according to the shortlex order. Two words are in the shortlex order relation when the first is shorter or, in case they have the same length, when the first comes first in lexicographical order. Moreover, we can naturally extend the relation $\leq$ on S to equivalence classes.
The cardinality of $S_{/\equiv}$ is not necessarily finite. Therefore, for practical purposes it is necessary to define finite approximations to the solution.

Definition 9.1.6 (n^{th} approximation) *Let $S_n = \{i, c, -\}^{\leq n} \subset S$ be the set of all operator combinations up to order n. For $w_1, w_2 \in S_n$ we define $\leq_n$ as $w_1 \leq_n w_2$ iff for all $x \in P$ we can derive $w_1(x) \leq w_2(x)$ by applying the axioms from Table 9.1 to elements $w \in S_n$ only (i.e., we restrict derivations to combinations of maximally n operators). Finally, let $\equiv_n$ be the symmetric closure of $\leq_n$. Then the n^{th} approximation of the generalized Kuratowski closure-complement problem is defined as computing the cardinality of $S_{n/\equiv_n}$.*

As described below, the n^{th} approximation of the problem can be visually represented as a directed graph whose vertices are the equivalence classes of $S_{n/\equiv_n}$ and whose edges represent one step of the $\leq_n$ relation.

Definition 9.1.7 (Approximating graph of order n) *Let $G = (V, A)$ be a directed graph, where we define the set of vertices $V = S_n$ and the set of arcs A by $(v_1, v_2) \in A$ iff $v_1 \leq_n v_2$ for $v_1, v_2 \in V$. Now let V' be the set of all strongly connected components (connected by*

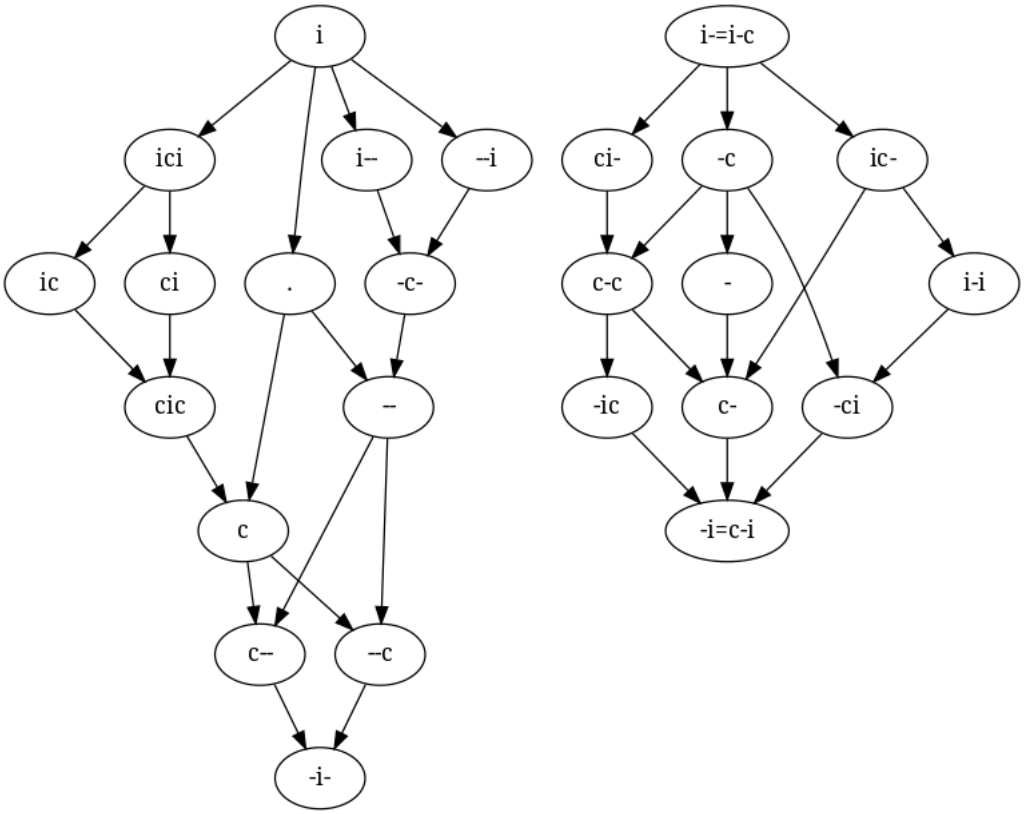


Figure 9.1: Approximating graph of order 3 for the generalized problem.

a directed path) in G. We then define the approximating graph of order n *as* $G' = (V', A')$ *where* $(v'_1, v'_2) \in A'$ *iff* $v'_1 \leq_n v'_2$ *for* $v'_1, v'_2 \in V'$.

The approximating graph can be represented in transitively reduced form, exploiting the transitivity of the $\leq_n$ relation. It can also be observed that every vertex in the approximating graph contains all the elements of the equivalence class it represents. Thus the graph itself provides a solution to the n^{th} approximation problem as the number of vertices in the graph is the cardinality of $S_{n/\equiv_n}$. Consequently, our goal is effectively to construct the graph by partitioning S_n into equivalence classes, which amounts to an inference procedure that determines if $[w_1]_{/\equiv} \leq_n [w_2]_{/\equiv}$ for $[w_1]_{/\equiv_n}, [w_2]_{/\equiv_n} \in S_n/\equiv_n$.

Figure 9.1 shows the approximating graph of order 3 for the generalized Kuratowski closure-complement problem. The vertices are subsets of S_3,

where only three vertices represent equivalence classes with more than one element. Note that '.' corresponds to the empty word ϵ. Also, if there is no symmetric closure between the vertices then they are not connected by arcs, this is a single graph.

We note that the n^{th} approximation is an approximation to the original problem in two ways. First of all it only shows classes whose canonical representative has length at most n. More importantly, however, it does not grant that two distinct classes in the n^{th} approximation will remain distinct for every $(n{+}m)^{\text{th}}$ approximation. Thus the cardinality of the graph may decrease or increase when moving to larger values of n. Nevertheless, the approximation procedure is monotone in the following sense:

- If two words belong to the same class in the n^{th} approximation, they will belong to the same class in any $(n{+}m)^{\text{th}}$ approximation.
- Advancing to the $(n{+}1)^{\text{th}}$ approximation can only collapse more classes or create new ones made only of words of length $n{+}1$.

Graph isomorphism is defined as follows.

Definition 9.1.8 *Let $V(G)$ be the vertex set of a simple graph and $E(G)$ its edge set. Then a graph isomorphism from a simple graph G to a simple graph H is a bijection $f : V(G) \to V(H)$ such that $uv \in E(G)$ iff $f(u)f(v) \in E(H)$.*

The following theorem holds.

Theorem 9.1.1 *If the solution of the generalized problem is finite, then there exists an n such that every $(n{+}m)^{th}$ approximation is isomorphic (as a directed acyclic graph) to the solution.*

The theorem states that approximations *stabilize*, in the sense that larger approximations only augment the cardinality of the equivalence classes, but they do not collapse any existent distinct classes, nor do they add new arcs to the approximating graph.
The theorem does not provide an effective way to decide if an approximation is (isomorphic to) the solution. Consequently, we postulate the following conjecture that provides a simple decision procedure to recognize solutions.

Conjecture 9.1.2 *In the solution of a generalized problem there exists an m such that, if for a given n the $n^{th}$and the $(n+m)^{th}$approximations are isomorphic, then they are isomorphic to the solution.*

We have not tried to prove the conjecture yet and the proof does not seem to be simple. In particular, we do not know what is the m for the set of axioms considered. Nevertheless, we employ an alternative to the conjecture to recognize which approximations are solutions. Let us assume that at a certain point the approximations seem to stabilize, i.e., the $(n+1)^{\text{th}}$ approximation is equal to the n^{th} approximation. We build a *syntactic model* of the solution as follows.

- We take the set P of all strings w made from $\{i, c, -\}$ such that w is a canonical representative of an equivalence class in the n^{th} approximation.
- We define an $\leq$ relation over P by taking the $\leq_n$ relation.
- The i, c and $-$ operators are obtained as finite maps that associate to each $w \in P$ the canonical representative of $i \circ w$ (respective $c \circ w$ and $- \circ w$) in the n^{th} approximation.

In order to verify if $(P, i, c, -)$ is a model for the problem (i.e., if all the axioms hold for $(P, i, c, -)$), we use the scientific data analysis tool

Octave [Eat08] to verify that all axioms hold. If they do, then the n^{th} approximation is isomorphic to the solution of the generalized problem because the model shows that all classes are distinct and moreover the number of classes is maximal because we have only equated combinations that had to be equated because they proved to be equal. Otherwise, we start computing larger approximations and eventually find an $(n{+}m)^{\text{th}}$ approximation that is not isomorphic to the n^{th} approximation that, a posteriori, is not stable.

A priori, if the conjecture is false it may be that all syntactic models built from approximations that are stable (i.e. $(n{+}1)^{\text{th}}$ isomorphic to n^{th}) turn out to be wrong. However, as we will see in Chapter 11, this has not been the case for the different instances of the generalized problem that we considered that are stable.

The following theorem, instead, is obvious:

Theorem 9.1.3 *If the solution of the generalized problem is infinite, then there exists an infinite increasing sequence of approximations with larger and larger cardinalities.*

Our experience shows that in this case a clear pattern emerges, which after some time allows us to predict what new classes will be generated passing from any n^{th} approximation to the $(n{+}1)^{\text{th}}$ approximation. This prediction can then be manually turned into a proof that these new classes will never be collapsed in later approximations and therefore the solution is infinite. Consequently, we focus on finding a solution to the n^{th} approximation of the problem.

CHAPTER 10

THE ADOPTED TERM REWRITING SYSTEM

Chapter Overview: *This chapter describes the adopted term rewriting system to solve the generalized Kuratowski problem. We first present the basic rewriting system, followed by the advanced rewriting system. We also present some variations of the Kuratowski problem that can be solved by our rewriting system.*

10.1 The Basic Rewriting System

We first present the basic rewriting system which uses the axioms given in Table 9.1 and computes approximating graphs for the generalized Kuratowski problem presented in Chapter 9. We use the standard terminology that is used in references such as [BN98]. The system can also be understood as an instance of a generalized equational reasoning

system that is defined in [Coe06], which corresponds more to the actual form in which the system was developed.

A preliminary observation is the fact that in the axiomatization of the problem we can replace the equality with a $\leq$ in all three idempotency inference rules as the (anti)monotonicity of the respective operator yields the equality automatically. For instance, idempotency of the i operator can be replaced by the axiom $i(x) \leq i(i(x))$ since, by monotonicity, we already have $i(i(x)) \leq i(x)$. Therefore, the only rule that employs an equality remains the antisymmetry rule for $\leq$.

The approximating graph of order n can now be computed in two steps:

1. First we compute the initial directed graph G from Definition 9.1.7 whose vertices are all the elements of S_n (words of length at most n) and whose arcs represent all pairs such that $(w_1, w_2) \in \leq_n$. The anti-symmetric rule of the $\leq$ relation and, more generally, equalities are not used in this step.

2. We then apply a standard connected component algorithm (from the OCAML graph library) to this graph. Since a connected component is made of all vertices that are mutually reachable, i.e., mutually less or equal, by antisymmetry of $\leq$ they are all equal. The resulting graph is then the approximating graph of order n that we are looking for.

The second step is completely standard and we can employ implementations directly from the OCAML graph library. Therefore, we only focus on describing the development of the first step.

In order to compute the first directed graph $G = (V, A)$ it is sufficient to find all pairs of vertices $(w_1, w_2) \in A$ in the transitive reduction of the graph of $\leq_n$ (recall that following Definition 9.1.7, w_1, w_2 are words of

length $\leq n$ in S_n), since the connected components algorithm does not distinguish between a transitively reduced and a transitively closed graph. In other words, we look for all pairs $(w_1, w_2) \in A$ such that $w_1 \leq_n w_2$ and there is no $w_3 \in V$ such that $w_1 \leq_n w_3$ and $w_3 \leq_n w_2$. By a close inspection of the rules that have a premise, it is easy to notice that all applications of the transitive rules can be pushed towards the root of the derivation tree. For instance, consider the monotone rule for i and assume (by induction hypothesis) that the derivation of the premise $x \leq y$ is obtained by means of a transitive rule whose premises are $x \leq z$ and $z \leq y$. It is therefore possible to conclude that $i(x) \leq i(z)$ and $i(z) \leq i(y)$ and then, with one final application of transitivity, that $i(x) \leq i(y)$. Since we are interested only in the transitively reduced graph, we avoid the use of the transitive and reflexive properties of $\leq$. The final preliminary observation is that, to compute all pairs (w_1, w_2) in the transitively reduced graph G, it is sufficient for every word $w \in S_n$ to compute the two sets $w{\downarrow} = \{w' \mid w' \leq_n w \wedge |w'| \leq |w|\}$ and $w{\uparrow} = \{w' \mid w \leq_n w' \wedge |w'| \leq |w|\}$ where $|.|$ is the length of the two combinations. The final set is then given by:

$$\bigcup_{w \in S_n} (\{(w, w') \mid w' \in w{\uparrow}\} \cup \{(w', w) \mid w' \in w{\downarrow}\})$$

In order to compute $w{\downarrow}$ and $w{\uparrow}$, we introduce the non confluent, Noetherian term rewriting system presented in Table 10.1. The term rewriting system manipulates both active configurations of the form $\langle w_1, w_2, d\rangle$ (where $d \in \{\leq, \geq\}$) and stuck terms w which cannot be reduced further. The intended big step semantics i.e. the overall result of the execution of the rewriting system is the following: an initial term $\langle \epsilon, w, d\rangle \rhd^* w'$ iff wdw' and $|w'| \leq |w|$. In particular, $w{\downarrow}$ can be

(saturates) $\langle w_1, --w_2, \geq\rangle \rhd w_1 w_2$	(antimonotone) $\langle w_1, -w_2, d\rangle \rhd \langle w_1-, w_2, d^{-1}\rangle$
(quasi-idempotent) $\langle w_1, ---w_2, \geq\rangle \rhd w_1-w_2$	(reduces) $\langle w_1, iw_2, \leq\rangle \rhd w_1 w_2$
(monotone) $\langle w_1, iw_2, d\rangle \rhd \langle w_1 i, w_2, d\rangle$	(idempotent) $\langle w_1, iiw_2, \geq\rangle \rhd w_1 iw_2$
(saturates) $\langle w_1, cw_2, \geq\rangle \rhd w_1 w_2$	(monotone) $\langle w_1, cw_2, d\rangle \rhd \langle w_1 c, w_2, d\rangle$
(idempotent) $\langle w_1, ccw_2, \leq\rangle \rhd w_1 cw_2$	(compatible-1) $\langle w_1, c-w_2, \leq\rangle \rhd w_1-iw_2$
(compatible-2) $\langle w_1, i-w_2, \leq\rangle \rhd w_1-cw_2$	

Table 10.1: The non confluent, Noetherian term rewriting system to compute $w{\downarrow}$ and $w{\uparrow}$.

computed as $\{w' \mid \langle \epsilon, w, \geq\rangle \rhd^* w'\}$ and $w{\uparrow}$ as $\{w' \mid \langle \epsilon, w, \leq\rangle \rhd^* w'\}$. The small step semantics i.e. formal description of the individual steps of the rewriting system is more technical and it involves generic configurations $\langle w_1, w_2, d\rangle$. The idea is that an initial reduction trace $\langle \epsilon, w, d\rangle \rhd^n \langle w_1, w_2, d'\rangle$ represents a partial derivation of wdw' for some yet unknown w'. In the two invariants $\{w' \mid \langle \epsilon, w, \geq\rangle \rhd^* w'\}$ and $\{w' \mid \langle \epsilon, w, \leq\rangle \rhd^* w'\}$, we have $w = w_1 w_2$ and $|w_1| = n$. The partial derivation built in a top-down manner starts with exactly n monotonicity/anti-monotonicity rules: if $w_1 = o_1 \ldots o_n$ where $o_j \in \{-, i, c\}$ then the j-th inference rule in the partial derivation is the monotonicity/anti-monotonicity rule for o_j. Moreover, the hypothesis of the partial derivation is $w_2 d' w_2'$ for some yet unknown w_2' such that $w' = w_1 w_2'$. According to this interpretation, a reduction trace $\langle \epsilon, w, d\rangle \rhd^* \langle w_1, w_2, d'\rangle \rhd w'$ corresponds to a derivation of wdw' where there is a w_2' such that $w' = w_1 w_2'$, the last inference rule in the

top-down construction is an axiom that proves $w_2 d' w_2'$ and $|w_2'| \leq |w_2|$. The proof that reduction traces of length n correspond to partial derivation trees of height n having the property just described is by induction on n. We only sketch here one case of the proof.

Each rule in Table 10.1 corresponds to the rule with the same name in Table 9.1. It means that applying the reduction rule adds the corresponding inference rule to the partial proof tree. The most interesting rule is the rule `antimonotone`: In order to proceed in the derivation we use one more application of antimonotonicity of complement by pushing $-$ on top of the stack w_1 (stack is a tool for systematically tracking locally defined data attached to the open sets of a topological space that comprises of objects that are linked by arrows) and looking for a new derivation for $w_2 d^{-1} w'$. To see that the rule is correct, assume that $\langle \epsilon, w, d \rangle \rhd^n \langle w_1, -w_2, d' \rangle \rhd \langle w_1 -, w_2, d'^{-1} \rangle$. By induction hypothesis, there is a partial proof derivation of wdw' built top-down that starts with monotone/anti-monotone rules for the operators in w_1 and whose hypothesis is $-w_2 d' w''$ for some yet unknown w'' such that $w' = w_1 w''$. By applying anti-monotonicity of $-$ we obtain a new partial proof derivation of wdw' whose new hypothesis is $w_2 d'^{-1} w'''$ and such that $w' = w_1 w'' = w_1 - w'''$. The reduction rule is therefore correct and by applying it we discover that $w'' = -w'''$ or, equivalently, that the next rule in the combination w' after w_1 is $-$.

Strong normalization of the term rewriting system can simply be proved by induction on the length of the second component of active configurations, which always decreases by one in all (anti)monotonicity rules. All remaining rules produce a stuck term.

By inspection of all the rules, it is easy to prove (by induction on the second component of an active configuration) that if $\langle \epsilon, w, d \rangle \rhd^* w'$ then

$|w'| \leq |w|$. Moreover, if $\langle \epsilon, w, d \rangle \rhd^* w'$ and $|w'| = |w|$ then $\langle \epsilon, w', d^{-1} \rangle \not\rhd^* w$. This is important for efficiency reasons, since it means that we are never generating the same arc twice (as $w_1 d w_2$ and $w_2 d^{-1} w_1$). The system clearly has several critical pairs between (anti)monotonicity rules and the remaining rules. Actually, it turns out that every critical pair is not joinable i.e. cannot be made equivalent to another and the system is thus non confluent i.e. diverges. Non-joinability is a feature of our system; because our rewriting rules are never applied under a context, from non-joinability it follows that we never compute the same arc twice in different ways.

Computing all normal forms of a term can be done very efficiently (in terms of actual, non asymptotic computational cost of the program): At every step at most two rules can be applied, one produces a stuck term and the other can be implemented as a tail recursive call. It is thus possible to simplify the code of an implementation for a generic term rewriting system.

10.2 The Advanced Rewriting System

Given a combination $w \in S_n$, the computation of $w{\downarrow}$ and $w{\uparrow}$ by means of the term rewriting system presented in the previous section is very efficient. Nevertheless, the number of combinations to be reduced is exponential in n and the number of reducts for each w is also exponential in n. The limiting factor for the computation of larger and larger approximating graphs is thus the memory required to hold the graph defined by $w{\downarrow}$ and $w{\uparrow}$, which is the initial directed graph G from Definition 9.1.7 before the computation of connected components.

To be able to compute larger approximations, we exploit the following result given in [Cor06]: There exist only 7 distinct equivalence classes of

combinations of closure and interior. While this result is well known in the literature and we can obtain it with our technique for very small values of n, we additionally observed that every class can be associated with a regular expression that generates all elements of the class. Note that this property does not hold any longer when we consider combinations with complement. The seven regular expressions are:

$$\epsilon, i^+, c^+, (ic)^+, (ci)^+, i(ci)^+, c(ic)^+$$

Taking as canonical representatives the shortest expressions in each class, we have the set of representatives as $\{\epsilon, i, c, ic, ci, ici, cic\}$. Let K be any regular expression that generates the set. When we consider combinations that also contain complements, and noting that $---x = -x$, we obtain that all combinations can be partitioned into an infinite number of sets of equivalent combinations whose representatives are all generated by the following regular expression E: $(-|--)?(K--?)^*K?$. The set that corresponds to a representative is the set obtained by replacing any occurrence of $-$ with an odd number of occurrences of $-$ and any occurrence of a term generated by K with an element of its equivalence class. For instance $-----icicicic----$ is a member of the set whose representative is $-ic--$. The sets that correspond to different representatives are not distinct according to the $\equiv$ relation. For instance $c-i-$ and $-i-$ are representatives of different sets, but $c-i- \equiv -i-$. Nevertheless, if two elements belong to the same set, than they are equivalent. Thus the $\equiv$ equivalence relation is more fine grained than the equivalence relation that is induced by partitioning with respect to regular expressions. Therefore, nodes representing sets that correspond to different representatives in the graph will collapse

$(--)$	$(\texttt{cc})$	$(\texttt{ii})$	$(\texttt{cici})$	$(\texttt{icic})$
$--w \rhd w$	$ccw \rhd w$	$iiw \rhd w$	$ciciw \rhd ciw$	$icicw \rhd icw$

$(\texttt{compatible-1 + i-idempotent})$	$(\texttt{compatible-2 + c-idempotent})$
$\langle w_1, c{-}iw_2, \leq \rangle \rhd w_1{-}iw_2$	$\langle w_1, i{-}cw_2, \geq \rangle \rhd w_1 i{-}w_2$

Table 10.2: Additional rewriting rules.

according to the $\equiv$ relation.

The idea to speed up our previous algorithm is to avoid the generation of the vertices (and relative arcs) that correspond to non-canonical representatives of the equivalence classes discussed above. These vertices will all belong to the connected component that will be collapsed to its canonical representative. For instance, for $n = 7$, our previous algorithm handles the vertices $\{-, ---, -----, --------\}$ as potentially distinct.

To implement the idea, we change the already presented algorithms in two ways as follows:

1. We change the definition of S_n with the following one. The changes apply to Definitions 9.1.6 and 9.1.7 and everywhere else in Section 10.1.

   $x \in S_n$ iff x is generated by the regular expression E and $|x| \leq n$

2. We integrate the rewriting system with the rules of Table 10.2 after dropping the rule `quasi-idempotent` and the two `idempotent` rules from the previous rewriting system. The reason why we drop these rules is that their left hand side will never match any active configuration due to restricting the definition of S_n.

Considering the rules in Table 10.2, we observe that all rules of the first line simplify a combination. When the rules are applied repeatedly they

put any combination into their K-normal form. The rules of the second line are obtained by applying Knuth-Bendix completion. Note, however, that our rewriting rules come from a non-symmetric relation ($\leq$) and we have to take care of this during the superposition phase of Knuth-Bendix completion. The names of the new rules are a concatenation of the names of the rules superimposed. The new rules are necessary to keep completeness after having changed the definition of S_n. For instance, because $c{-}ii$ no longer belongs to S_4, we are no longer considering combinations like $(c{-}ii)\uparrow \ni -i$. The new rewriting rule generated by Knuth-Bendix completion takes care of adding $-i$ to $(c{-}i)\uparrow$ by implicitly performing a step of ii-expansion. Note that, in the original rewriting system, monotonicity of i was only used to perform a step of ii-contraction.

In Table 10.2, we only list two rules obtained from the Knuth-Bendix completion because all the others are logically redundant: they enable the derivation of $w_1 \in w_2{\downarrow}$ when there exists a w_3 such that $w_1 \in w_3{\downarrow}$ and $w_3 \in w_2{\downarrow}$. The redundant rules have been pruned by hand, but it is surely possible to automate the procedure.

The new term rewriting system remains Noetherian: all the new rules decrease the length of either the (no longer stuck) combinations or the second component of the active configurations. Of the new rules, only those in the first line need to be applied several times in order to obtain the normal form of a term. However, it is easy to show that all critical pairs are joinable. Newman's lemma states that a terminating rewriting system, that is, one in which there are no infinite reduction sequences, is confluent if it is locally confluent. Therefore, by Newman's lemma, the normalization step implemented by the rules in the first line is confluent, as expected. This completes the proof of Theorem 10.2.1.

Theorem 10.2.1 (Correctness and completeness) *The algorithm based on the advanced rewriting system just described correctly computes the n^{th} approximation of the problem for each n.*

The advanced rewriting system is obtained by rewriting in one step all combinations to their canonical representatives in the equivalence classes identified by the regular expression considered. The same trick can be used more aggressively when we build the n^{th} approximation after the $(n-1)^{\text{th}}$. Indeed, we can add to the n^{th} term rewriting system one rewriting rule per combination of length $(n-1)$ that in one step rewrites the combination to its $(n-1)^{\text{th}}$ canonical representative.
Since the number of these additional rules is exponential in n, we avoid running the Knuth-Bendix completion, by using the new rules only to normalize terms that are not active configurations. The consequence is that we have to normalize exactly the same set of combinations and so we do not save time during the graph generation phase with the rewriting system. The size of the generated graph, however, will be much smaller, since it will no longer contain nodes that are not in $(n-1)^{\text{th}}$ normal form. The benefit is thus a significant reduction of the computational cost (both memory and time) for the computation of the connected components when generating the approximating graph of order n.
The proof of correctness and completeness of the rewriting system obtained with this final improvement is a simple corollary of Theorem 10.2.1. The implementation of the improvement is very cheap: the additional rewriting rules generated at the $(n-1)^{\text{th}}$ step can only be applied to terms that are stuck according to all other rules. Moreover, they only generate stuck terms. Therefore we can implement this final step as a simple look-up in a trie. A trie is a data structure that stores the information about the contents of each node in the path from the

$$\frac{}{-- = \epsilon}(\texttt{axiom 1})$$

$$\frac{}{c- = -i}(\texttt{axiom 2})$$

$$\frac{}{i- = -c}(\texttt{axiom 3})$$

$$\frac{}{c-- = --c}(\texttt{axiom 4})$$

$$\frac{}{c = -i-}(\texttt{axiom 5})$$

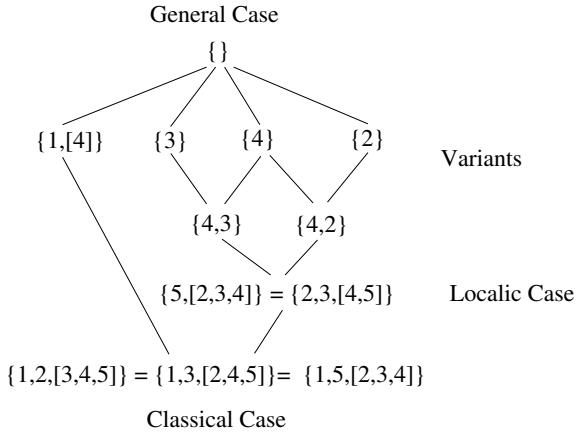


Figure 10.1: Variations of Kuratowski's problem.

root to the node, rather than the node itself.

10.3 More Variations of Kuratowski's Problem

The rewriting system presented so far has been developed as a bespoke approach to solve the generalized Kuratowski problem. However, it turns out that with a parametric implementation, our procedure can be applied to a variety of related problems lying between the classical and general problem. These problems are generated by introducing axioms which restrict the general problem, or generalize the classical one.
Figure 10.1 demonstrates variations of Kuratowski's problem, where the axioms on the left hand side gradually refine the generalized problem to the classical problem according to the graph on the right. The nodes are given as sets of included axioms, with the root as the empty set representing the generalized case. Furthermore, axioms derivable from already included ones are given in square brackets.
The variations are motivated by Sambin's [Sam12] work who proposed the generalized problem in the context of intuitionistic point-free topology. Axioms 1–5 are likewise inspired by axioms commonly found in

topological problems. For example, axiom 1 postulates the complement operator as idempotent, corresponding to its use in classical logic. Axiom 4, $c-- = --c$, is another axiom that is frequently satisfied by concrete basic topologies (see [Sam12]). Adding axiom 5 to the generalized problem, further restricts the saturation operator c. The axiomatization obtained is the one for locale theory, for which it is already known in the literature [WY00] that a maximum of 21 combinations exists. Weaker cases than the localic one can be obtained by effectively splitting axiom 5 into axioms 2 and 3, and considering those either separately or in combination with axiom 4.

All the presented problems in the generalized problem domain can be obtained using our approach, by simply adding the corresponding axioms to our advanced term rewriting systems as pairs of reductions over active configurations. The Knuth-Bendix completion must also be applied to combine the new rules with the ones of the advanced term rewriting system.

10.4 Summary

We have developed a bespoke Term Rewriting System (TRS) implementing the axioms of the generalized Kuratowski problem defined in Chapter 9. In this chapter, we first presented the basic rewriting system that used the axioms given in Table 9.1. The TRS computes the approximating graph of order n for the generalized Kuratowski problem by first computing the initial directed graph G where the vertices represent the elements of S_n (words of length at most n) and arcs represent all pairs such that $(w_1, w_2) \in \leq_n$. In the second step, a standard connected component algorithm is applied to G. The resulting graph is then the approximating graph of order n.

To be able to compute approximating graphs of larger order, we then presented the advanced term rewriting system where we exploited the result from the literature that there exist only 7 distinct equivalence classes of combinations of closure and interior. In addition to that, we observed that every class can be associated with a regular expression that generates all the elements of the class. This allowed us to avoid the generation of the vertices (and relative arcs) that correspond to non-canonical representatives of the equivalence classes. Hence, allowing us to further narrow down the computations and compute larger approximating graphs.

CHAPTER 11

METHODOLOGY, IMPLEMENTATION AND RESULTS

Chapter Overview: *This chapter presents the methodology we have used to solve the generalized Kuratowski problem, implementation details and the results obtained. We first describe the methodology that presents a novel approach of system combination with the active involvement of the user. We then describe the various components used in the implementation of the rewriting engine, and also provide the experimental settings used. This is followed by the discussion of results and their verification. Finally, some concluding remarks are presented.*

The main focus of research into combining mathematical reasoning systems has mainly been on how to solve problems while limiting user involvement. If users were involved, it was either as a last resort when a proof or a computation was stuck, or for driving interactive proofs, for example in proof assistants, where the user makes the main decisions on

how a problem is solved while the system only verifies the validity of steps. In contrast, there has been little study into how automated system components and user interaction can be integrated to mutually support each other in developing problem solutions.

We present a novel system combination that aims at actively working with the user, by computing and presenting results in a fashion that allows the user to obtain valuable insights into the structure of solutions and to guide computations via providing feedback. This human inspection of intermediate results allows the user to explore the knowledge that can help in finding results in infinite domains. The main goal is to give a working mathematician experimental support for the discovery of non-trivial theorems.

We exemplify the methodology with the domain of generalized Kuratowski problems that has originally motivated its development. While these problems are regular classification problems, the main difference to those presented for example in [CMSM04, DSS11] is that the number of classes that have to be considered is not necessarily finite. However, they can be tackled in our framework by a suitable reformulation into a graph rewriting problem and by combining a number of different reasoning techniques. In particular, we have combined a bespoke, parametrisable term rewriting system with off-the-shelf tools for graph reduction and visualization. This resulted in a semi-automatic procedure that enabled incremental graph generation and inspection which enables user adaptation of the search for the solution.

Finite solutions can be automatically verified with the scientific data analysis tool Octave [Eat08], while the existence of infinite ones need to be proved manually, once a significant pattern in the solutions can be discovered. As a result of our experiments, we were able to prove a

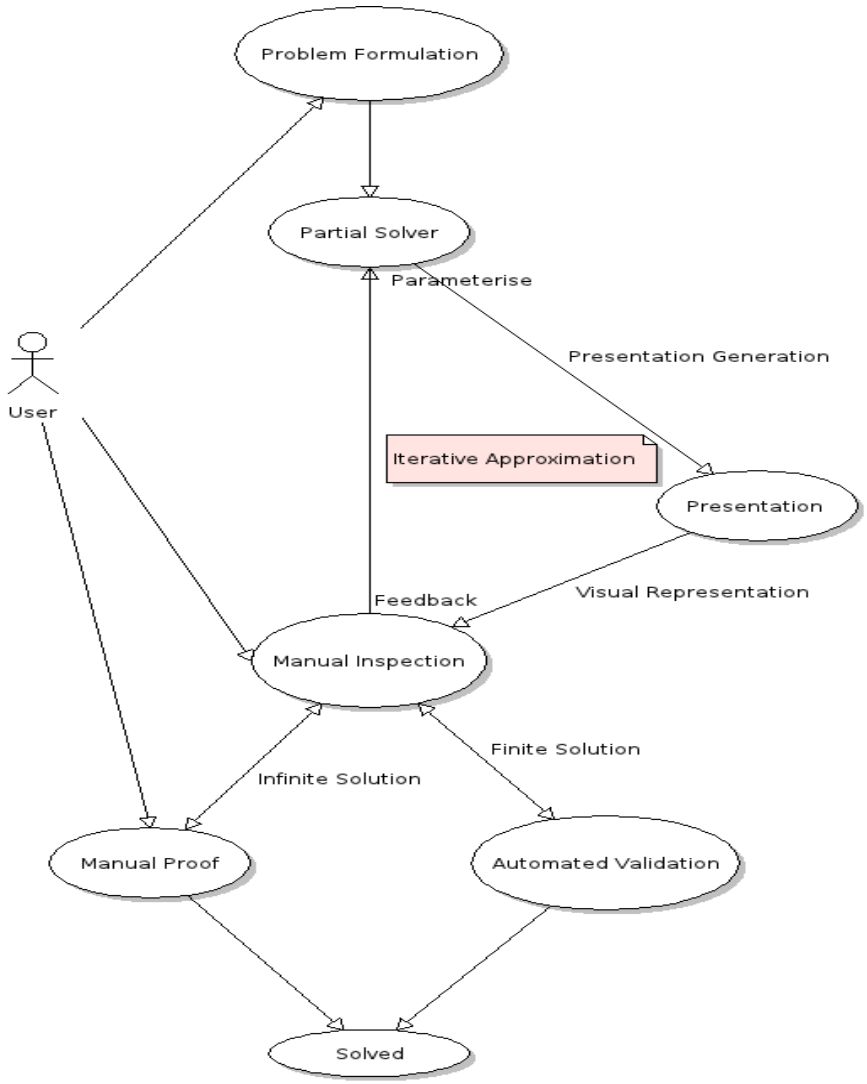


Figure 11.1: Methodology

number of novel classification theorems from the Kuratowski problem domain.

11.1 Methodology

We now present our methodology that aims at integrating meaningful user decisions into iterative automated mathematical problem solving. The main emphasis is that the process should be neither purely automatic nor driven by a user that is just describing a known solution, as in an interactive proof assistant. Instead, we view both automatic component and user as active partners in a symbiotic collaboration. As

one of the results of the methodological view, we obtain a requirement specification for the formulation of problems that can be considered.
Fig. 11.1 presents the methodology abstractly in a use case notation. The user defines the problem in terms of this methodology, and feeds the definition to the partial solver, which in turn feeds the partial solution to a presentation tool in order to transform the partial results into a format that allows the user to inspect the state of the solution and, depending on the outcome, to adjust the systems accordingly. Eventually, the user gets satisfied and conjectures what a solution to the problem is. This could be a partial solution that is thought to be a real solution or it could be an extension of a partial solution suggested by user inspection. The next step is to prove that the conjectured solution is valid. When the conjectured solution is **finite**, the proof can be done automatically by testing if the solution has the expected properties. Otherwise, if the conjectured solution is **infinite**, the user normally has to do the proof by hand. Obviously, if a verification or proof attempt fails, the interaction loop can be resumed to look for the next potential solution.
Note that the components in Fig. 11.1 themselves can hide complex processes. In particular, they can contain anything from single, bespoke algorithms, over simple tools to filter or transform problems, to more complex tools like theorem provers or computer algebra systems, as well as entire system combinations themselves. Furthermore, there is in principle no limitations to additional interactions between integrated systems, for example, results of the presentation could automatically be fed back into the partial solver.
Clearly, the core of the methodology is the Iterative Approximation triangle, that serves as the main feedback loop, where user interactions can be essential in steering the solution process. To facilitate a symbiotic

collaboration between the user and the partial solver, the presentation of partial results plays a crucial role. It helps the user to comprehend the problem and its properties, as well as interim results, to eventually either guide the solver or the user towards a solution. For example, when dealing with a classification problem, that has many thousands of equivalence classes, and it is unknown whether the problem is finite, presenting results as a graph is more useful to the user than having to go through text. The additional possibility to focus on or hide parts of the graph is also crucial to understand the graph structure.
Our methodological view has four distinct requirements on problem formulation. It should be:

(1) **Parameterisable:** The chosen reasoning technique for the partial solver must not only allow for changes in the initial problem definition, but also for parametrization in between producing partial solutions. These parametrizations can require :

   (i) the inclusion of new definitions derived from partial results,

   (ii) the restriction of the generation of partial solutions with respect to a specific feature,

   (iii) the production of partial solutions with varying methods, and

   (iv) the generation of partial solutions for different sub-problems.

(2) **Iterative:** A problem can be decomposed into a sequence of partial solutions. These should be meaningful, in the sense that each iteration represents a coherent and complete step. This is essential to enable user comprehension and thus interaction. Nevertheless, partial solutions can be focused towards certain features of the problem, only expanding further those features that are interesting

and could lead to a solution. This can be achieved by parametrising the partial solver.

(3) **Approximable:** The iteration of partial solutions should always lead towards a final solution of the problem; that is, it never stagnates or even leads away from the solution. This requirement is particularly helpful when a solution of a classification problem is infinite.

(4) **Presentable:** The presentability of partial solutions is a central feature of our methodology, and crucial for successful inclusion into the problem solving process. In particular, dealing with large amounts of data is a tedious task, which can hide the most interesting features that can actually lead to the solution. Consequently, care has to be taken to choose an adequate presentation of partial results and ensure their computation.

11.2 Implementation Details

We have solved the generalized Kuratowski problems by integrating a number of different systems following the methodology from Section 11.1. Figure 11.2 presents the configuration of systems for the interactive loop and the validation of solutions. Our approach uses a combination of a bespoke, parametrisable term rewriting system with off-the-shelf tools for graph reduction and visualization that are defined as follows:

- *Term Rewriting System (TRS):* has been written in pure OCaml using a graph data structure at its core. The connected-components algorithm exploits the `ocamlgraph` library [CFS08] instantiated with an ad-hoc, optimized, hashing function for equivalence classes of combinations. The TRS is fully

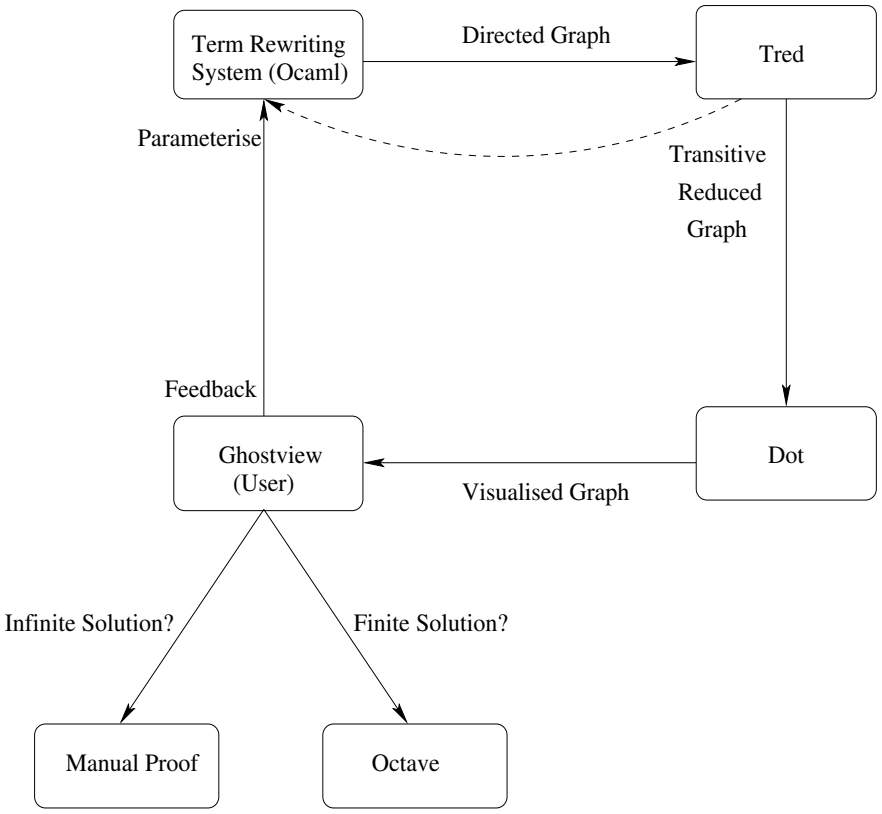


Figure 11.2: System setup for experiments in the Kuratowski problem domain.

parametric not only on the list of reduction rules, but also with respect to the words of S (i.e., the combination of operators) it generates.

- ***tred:*** is a tool in the graphviz library [Gra] to transitively reduce directed graphs. It is applied for graph optimization on the approximation graphs produced by the TRS in each iteration. Its results are not only important for the graph presentation but can also be fed back into the TRS to reduce the size of the inference problem.

- ***dot:*** is another tool in the graphviz library that draws suitably presented graphs in PostScript. This enables the visualization of the output of Tred.

- *User:* The visualization of the approximation graph can be examined by the user, for example directly using a viewer like Ghostview or, given the potential very large size of some of the graphs, further processed to zoom in and examine interesting areas.

- *Octave:* is a scientific data analysis tool [Eat08], very similar to Matlab but freely available. Octave is employed when the approximating solutions seem to have stabilized, i.e. when the $(n{+}1)^{\text{th}}$ approximation is equal to the n^{th} approximation. A detailed verification of the results with Octave is presented in Section 11.3.

The rewrite engine is implemented from scratch, allowing us to take care of the peculiarities of the rewriting system, e.g., by exploiting as much as possible tail recursive calls. This approach allows us to generate graph representations that are manually inspected and help in the generation of elements in S in order to explore particular subgraphs, which, to the best of our knowledge, is difficult to achieve in any existing system. Furthermore, our implementation allows us to be careful with memory consumption. Nevertheless, when supplying the rewrite engine with the rules of the advanced rewriting system, the program runs out of memory after about 12 minutes on one of the cores of a server equipped with a 2.4GHz Intel Xeon processor and 48GB of RAM producing an approximating graph of order 16. That essentially means that it explores all words generated by the regular expression given in Section 10.2 of length at most 16, deriving all equations and inequalities that are provable without using combinations of length ≥ 17. The initial graph generated by the rewriting system contains $1,771,825$ vertices, corresponding to all the combinations of length up to $\equiv_{16}$, and $8,687,605$ arcs, corresponding to steps of the $\leq_n$ relation. The approximating

graph obtained after the computation of the strongly connected components contains $44,138$ vertices, corresponding to distinct equivalence classes. We are not able to compute the number of arcs in the transitively reduced graph as the `tred` tool does not terminate within a 2 hour time-limit and memory limit for approximating graphs of order greater than 12. Note that `tred` has been run in separate threads of the computations of our rewrite system.

11.3 Verification

We use a model verification technique that employs Octave to verify the results. Octave is a scientific data analysis tool [Eat08], very similar to Matlab, but freely available. An automated theorem prover could also be used, but we preferred writing our independent code.

We employ Octave when the approximating solutions seem to have stabilized, i.e. when the $(n{+}1)^{\text{th}}$ approximation is equal to the n^{th} approximation. This indicates to us that a finite solution could have been found, i.e. a conjecture of the fact that the $(n{+}m)^{\text{th}}$ approximation is equal to the n^{th} approximation for every m. Since we cannot test the conjecture on every m, therefore, we employ a model verification technique as an alternative that is described below.

Let us assume that at a certain point the approximations seem to stabilize. We build a *syntactic model* of the solution by first taking the set P of all strings w that is made from $\{i, c, -\}$ such that w is a canonical representative of an equivalence class in the n^{th} approximation. Then we define a $\leq$ relation over P by taking the $\leq_n$ relation. The i, c and $-$ operators are obtained as finite maps that associate to each $w \in P$ the canonical representative of $i \circ w$ (respective $c \circ w$ and $- \circ w$) in the n^{th} approximation. We are now left with the problem of verifying

if $(P, \leq, i, c, -)$ is a model for the problem, i.e. if all the axioms hold for $(P, \leq, i, c, -)$. If they do, the n^{th} approximation is isomorphic to the solution of the generalized problem. Otherwise, we start computing larger approximations and, if the problem has an infinite solution, we will eventually find an $(n+m)^{\text{th}}$ approximation that is not isomorphic to the n^{th} approximation that, a posteriori, was not stable.

This model verification technique has an important characteristic in that it allows a partial *independent verification* of the solution. Indeed, the parametrized partial solver at the base of our methodology is user fed and likely to be a complex piece of software that employs optimizations and heuristics. Complex optimizations are likely to introduce bugs and the additional rules that are introduced by the user to speed up the process and narrow down the problem domain are also prone to errors. A failed independent verification, even if partial, could hint at errors in the partial solver or its instantiation.

For the sake of independent verification, as well as for performance concerns, minimization of the implementation effort and maximization of confidence in the results, we decided to avoid implementing our own verifier. Instead we have described $(P, \leq, i, c, -)$ as a set of matrices and we have used Octave to verify that all axioms hold after rephrasing them as properties on the matrices. We now describe the details of the verification process. Let n be the cardinality of P and let $\{w_1, \ldots, w_n\}$ be a canonical enumeration of P. The relation $\leq$ is represented as the $n \times n$ boolean matrix $\leq$ such that $\leq (i, j) = 1$ iff $w_i \leq w_j$. Similarly, every function $f \in \{c, i, -\}$ is assumed as a relation (a subset of $P \times P$) and represented by the $n \times n$ boolean matrix f such that $f(i, j) = 1$ iff $f(w_i) = w_j$.

Finally, the axioms given in Table 9.1 and Figure 10.1 are re-phrased in

(`reflexive`) $all(diag(leq) == 1)$	(`functional-m`) $all(all(m * ones(size(m)) == 1))$
(`transitive`) $all(all(leq == (leq * leq\ != 0)))$	(`anti-monotone`) $all(all(leq' <= m * leq * m'))$
(`anti-symmetric`) $all(all(leq\ .* leq' == eye(size(leq))))$	(`saturates`) $all(diag(leq * m' * m' == 1))$
	(`quasi-idempotent`) $all(all(m == m * m * m))$
(`functional-i`) $all(all(i * ones(size(i)) == 1))$	(`functional-c`) $all(all(c * ones(size(c)) == 1))$
(`reduces`) $all(diag(i * leq) == 1)$	(`saturates`) $all(diag(leq * c') == 1)$
(`monotone-i`) $all(all(leq <= i * leq * i'))$	(`monotone-c`) $all(all(leq <= c * leq * c'))$
(`idempotent-c`) $all(all(c == c * c))$	(`idempotent-i`) $all(all(i == i * i))$
(`compatible-1`) $all(diag(m * c * leq * m' * i' == 1))$	(`compatible-2`) $all(diag(m * i * leq * m' * c' == 1))$
(`axiom-1`) $all(diag(m * m * leq == 1))$	
(`axiom-2`) $all(diag(i * m * leq * c' * m' == 1))$	(`axiom-3`) $all(diag(c * m * leq * i' * m' == 1))$
(`axiom-4`) $all(all(m * m * c == c * m * m))$	(`axiom-5`) $all(all(c == m * i * m))$

Table 11.1: Axioms for the matrix representation of $(P, \leq, c, i, -)$.

Table 11.1 to apply on the matrix representation. The axioms in Table 11.1 are given in Octave syntax. To avoid clashes with Octave operators, the matrices for $(\leq, c, i, -)$ are called respectively (leq, c, i, m). The other Octave commands used are:

- A' for the transpose of the matrix A;
- $*$ for matrix multiplication;
- $.*$ for element-by-element matrix multiplication;
- $==$, $<=$ and $!=$ for element-by-element equality, inequality and

dis-equality tests;

- $diag(A)$ to return the diagonal of the matrix A;
- $eye(n)$ to return an identity $n \times n$ matrix;
- $size(A)$ to return the size n of the $n \times n$ matrix;
- $all(A)$ where A is a boolean matrix to return a row vector of ones and zeros with each element indicating whether all the elements of the corresponding column of the matrix are ones;
- $all(v)$ where v is a row vector to check if all element are ones.

The only new axioms with respect to Table 9.1 and Figure 10.1 are the `functional-f` axioms that for each $f \in \{c, i, -\}$ check that the f relation is actually a function. The reader should convince himself that all other matrix expressions yield 1 (where 0=(false) and 1=(true)) iff the corresponding old axiom holds.

As an example, we show here how to prove that `compatible-1` for matrices is equivalent to the original formulation. Suppose that $all(diag(m * c * leq * m' * i' == 1))$ yields 1. Then for all j it must be $(m * c * leq * m' * i')(j, j) = 1$. If F and G are boolean matrices that encode functions as relations, matrix multiplication $F * G$ encodes the composed relation $G \circ F$. Indeed $(F * G)(j, l) = \Sigma_h F(j, h) * G(h, l)$ where $F(j, h) = 1$ iff $F(j) = h$ and $G(h, l) = 1$ iff $G(h) = G(F(j)) = l$. Moreover, matrix transposition encodes the inverse function. Thus $m * c$ encodes $c \circ m$ and $m' * i'$ encodes $(m \circ i)^{-1}$. Thus $(m * c * leq * m' * i')(j, j) = 1$ iff there exists two necessarily unique h and l such that $(m * c)(i, h) = 1$ and $leq(h, l) = 1$ and $(m' * i')(l, j) = 1$ i.e. iff there exists two necessarily unique h and l such that $c(m(j)) = h$ and $h \leq l$ and $m(i(j)) = l$, i.e. iff $c(m(j)) \leq m(i(j))$.

Obviously, if the classification is infinite, we have to show that the approximating graphs grow infinitely large. It is actually sufficient to spot in the growing approximations an infinite, regular enough growing sub-graph and show that it grows infinitely. In a sense, this is the most challenging part of the problem. We will discuss how we achieved this using graph inspection in the next section.

11.4 Results

The chaotic nature of the resulting graph does not help very much in finding any simple description of either the set of equivalence classes or the elements of most of the equivalence classes. Nevertheless, the manual inspection of the generated graph has allowed us to spot sufficient regularity to solve the problem by showing that the number of equivalence classes is infinite. In fact, all the equivalence classes whose representatives are generated by the following regular expression are distinct: $c?(--c)^*(--)?$. Moreover, each one is less than or equal to every other class generated by a longer representative (e.g. $--c \leq --c--$) and they are all bounded by $-i-$, which is also distinct from them and is the minimum i.e. least element of the lattice. Figure 11.3 contains a clipping of the approximating graph of order 12 for the generalized problem visualized with the `dot` tool. The clip contains the approximation of the infinite subgraph with elements of the $c?(--c)^*(--)?$, together with some surrounding nodes. The outgoing arc at the bottom leads to the bottom element of the graph, $-i-$, that is not visible. It is obvious to see that the entire subgraph

(i) has only one outgoing arc to the bottom element,

(ii) contains fewer elements than all elements in the remainder of the

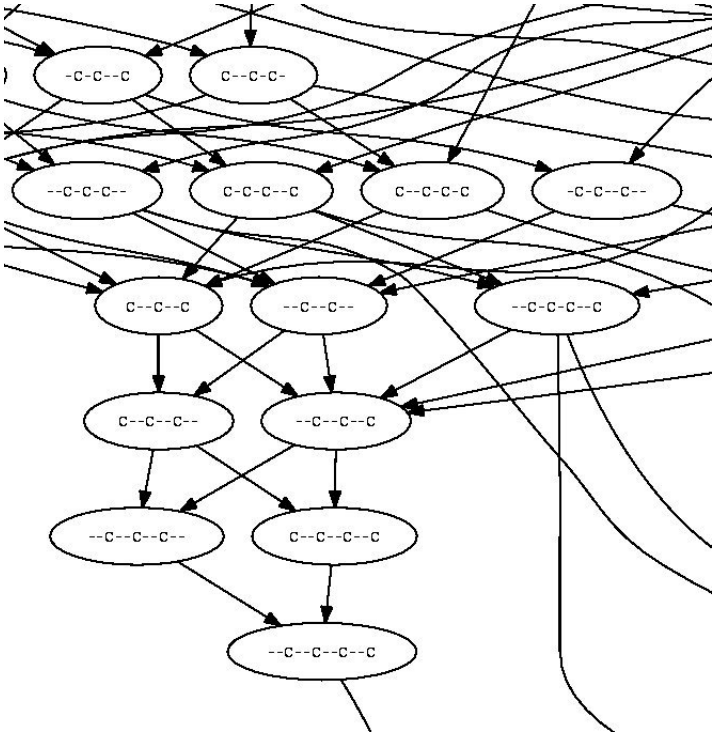


Figure 11.3: Infinite subgraph for the generalized problem.

graph, and

(iii) grows downwards with increasing word length.

We now give the proof of the formal argument that leads to the above result using our rewriting formalism.

Proof:

Firstly, to demonstrate that the equivalence classes generated by the regular expression $r = c?(--c)^*(--)?$ indeed constitute an increasing sequence w.r.t. $\leq$, we let $\langle w_1, w_2, \geq \rangle$ be any configuration such that $w_2 \neq \epsilon$ and w_1 and $w_1 w_2$ are generated by r.

Induction Hypothesis: If $\langle w_1, w_2, \geq \rangle \rhd^* w$ then w is generated by r and is shorter. The argument is by induction over the length $|w_2|$:

1. Suppose w_2 starts with c or with $--$. Then either one of the `saturates` rules is applicable, resulting in a shorter expression.

2. Suppose w_2 starts with c and `monotone` is applicable. Thus w_2 is shorter and we can apply the induction hypothesis.

3. Suppose w_2 starts with $-$ and `antimonotone` is applicable. The new configuration is $\langle w_1-, -w_2', \leq\rangle$. The only applicable rule is now `antimonotone` again and we can conclude using the induction hypothesis.

Similarly, for the base case we can show that $-i-$ is indeed the bottom element i.e. least element: Let $\langle w_1, w_2, \geq\rangle$ be any configuration such that $w_2 \neq \epsilon$ and w_1 and w_1w_2 are generated by r. If $\langle w_1, w_2, \leq\rangle \rhd^* w$ then w is in the same class as $-i-$. Again by induction over $|w_2|$ we can show:

1. Suppose w_2 starts with $c--$ with `compatible-1` we get $w_1-i-w_2 = -i-$.

2. Suppose w_2 starts with c, then `monotone` is applicable and we can apply the induction hypothesis.

3. Suppose w_2 starts with $-$ then `antimonotone` is applicable. The new configuration is $\langle w_1-, -w_2', \leq\rangle$. The only applicable rule is now again `antimonotone` and we can conclude using the induction hypothesis.

We have applied our implementation to other problems in the domain that are introduced in Chapter 10. This has enabled us to verify the results known from the literature of 14 and 21 combinations in the classic and localic case, respectively. For the remaining problems we have obtained a mixed picture of both finite and infinite cases.
Table 11.2 lists the approximating graphs from order 14 to 16 for the infinite cases in terms of vertices and arcs as well as infinite subgraphs

Axiom set	Order 14		Order 15		Order 16		Infinite subgraph
	Classes	Arcs	Classes	Arcs	Classes	Arcs	
$\emptyset$	10439	?	16869	?	27315	?	$(--c)^*$
$\{2\}$	135	269	142	285	149	299	$(--ci)^*$
$\{3\}$	135	269	142	285	149	299	$(--ic)^*$
$\{4\}$	278	640	283	649	288	660	$(--ici)^*$

Table 11.2: Approximating graph for all the variants that do not stabilize.

identified. Again, no arc count could be computed for the general case due to non-termination of `tred`.

Up to this point, we have proved formally only the infinite subgraph of the general case. Whilst adding axiom 2 or 3 or 4 only, the approximating graphs also continue to grow, the infinite subgraph that we spotted in the general case collapses to a finite one as the equation $c-- = --c$ forces all combinations generated by the regular expression $c?(--c)^*(--)?$ into less than four classes. Consequently, the argument we have used to show that the general case is infinite no longer holds. Thus the formal proof for these cases is still outstanding.

For the remaining problem variants, the approximating graphs stabilize. The exact figures for the graphs are given in Table 11.3. Axiom sets $\{1, 2, 3\}$ and $\{2, 3\}$ are the classical and localic cases from the literature and we can observe that in our system their approximating graphs stabilize after only a few iterations. Similarly, for the axiom combinations $\{2, 4\}$ and $\{3, 4\}$, the set of equivalence classes stabilizes quickly at 35 classes after 8 iterations. Finally, when adding axiom 1 alone we get a

Axiom set	Classes	Arcs	Stabilising Iteration
$\{1\}$	126	268	13
$\{1, 2, 3\}$	14	16	4
$\{2, 3\}$	21	31	5
$\{2, 4\}$	35	57	8
$\{3, 4\}$	35	57	8

Table 11.3: Approximating graphs for all variants that stabilize.

stable approximation after 13 iterations with 126 classes. The correctness of these cases, is verified using the Octave tool to check the axioms on the syntactic model generated from the stabilized solutions.

11.5 Concluding Remarks

We have presented a methodology that provides a framework for symbiotic collaboration between a user and a problem solver. We have motivated the need for an adequate problem formulation that enables iterative inspections of partial solutions for the user to steer the overall process. This methodology is particularly effective if solutions are infinite or computationally intractable to solve automatically. Indeed it has helped us to solve a number of open problems in the generalized Kuratowski problem domain, using a specialist term rewriting system together with a number of off-the-shelf tools.

The approach is capable of showing several million lemmas about relations between combinations of operators, which has allowed us to iteratively approximate the solution to the problem. The resulting graph exhibited enough regularity to enable us to show that the solution space of the problem is infinite, thereby successfully closing the problem. A posteriori, the proof that the number of combinations is infinite was relatively easy and the infinite set of distinct combinations is generated by a simple regular expression. Nevertheless, the problem has remained open for more than nine years, and the clutter in the rest of the graph made it difficult to spot the infinite subgraph.

From a mathematical point of view, the result is quite interesting. It shows that the generalization to a saturation operator partially independent from the reduction operator greatly adds to the expressive power of the system. This is one of the main intuitions at the base of the

Basic Picture of Sambin [Sam12], a complete re-formulation of point-wise and point-free topology that is deeply rooted in intuitionistic and predicative logic.

Part IV

Conclusions

CHAPTER 12

CONTRIBUTIONS

We have presented our approaches to solve complex mathematical problems having finite and infinite solutions by exploring the domain knowledge by combining various systems for solution generation. Our first approach combines symbolic computations and automated theorem proving to automatically explore the structural information of small size finite algebraic structures. This automated exploration helps in discovering knowledge that when as given as input pushes the boundaries of model generation systems to compute large size examples of finite algebraic structures. Our second approach of system combination actively involves the user to perform inspection of the graphical results to gain valuable insights into the structure of the solutions and provide useful feedback to guide the computation and produce results in infinite domains.

In this thesis, we have addressed the hypotheses presented in Chapter 1 as follows:

- We have used the combination of diverse automated reasoning techniques and other mathematical software tools to push the boundaries of mathematical discovery by generating and structuring additional knowledge in sufficiently diverse domains.

We have demonstrated this by the formation of a combined reasoning system that is more powerful compared to using a stand-alone system.

- We have used automated theory exploration in the finite discrete domain of quasigroups via symbolic computations and automated theorem proving to pre-compute additional knowledge that has enabled us to generate solutions for large size examples of quasigroups with interesting properties that were previously not possible to generate.

- We have exploited the structural knowledge by using diverse systems such as a specialist term rewriting system and visualization tools where active involvement of the user is a necessity to approximate infinite solutions for the generalization of Kuratowski closure-complement problems to point free topology that had remained open for several years.

The remainder of this chapter reviews, in more detail, the contributions of this thesis.

12.1 Combining Systems to Solve Complex Mathematical Problems

In our work, we have combined disparate reasoning techniques and other mathematical software tools for solving complex mathematical problems. In Chapter 7, we have seen a description of a model that combines various systems for pre-computing additional constraints according to our techniques presented in Chapter 6. It integrates a mix of bespoke algorithms we have implemented and off the shelf tools. Furthermore, in Chapter 11 we have seen a novel approach of combining systems where

automated system components and user interaction are integrated to mutually support each other in developing solutions to the problems. The user is actively involved in the inspection of the structure of solutions to discover knowledge and provide useful feedback into the system combination. In Chapter 11, we have seen the diagrammatic experimental set up of the system combination and present the various components used such as the specialist term rewriting system, graph algorithms, visualization tools and verification tool.

12.2 Automated Theory Exploration for Computing Large Size Examples in Finite Domain

One of the aims of our research was to explore the structural domain knowledge of small size algebraic structures to aid in the computation of large size structures in finite algebras. For this exploration, we have primarily focused on quasigroups. Chapter 4 gives a detailed background on quasigroups. Chapter 6 defines our two novel techniques for pre-computing elements where we evolve a set of elements using two filters, firstly by symbolic verification and secondly by automated theorem proving. Furthermore, in Chapter 6, we also defined some implied constraints which were computed in [CM01, CCM06]. We extended their work by encoding the quasigroup model generation problem with the additional constraints for a number of quasigroup properties and perform the final model generation using different automated reasoning tools.

In chapter 7, we have performed a comparative analysis of three main types of automated reasoning tools for the final model generation of large

size quasigroups structures, i.e. a constraint solver, a satisfiability solver and a model generator.

12.3 Approximating Solutions in Infinite Domains

We have studied the generalized version of Kuratowski's classical closure-complement problem from a point-set topology perspective. We have defined the problem in detail in Chapter 9. The problem had remained open for several years. To solve the problem, we have used a computational procedure that combines a term rewriting system with bespoke graph algorithms which is described in detail in Chapter 10. The resulting graph exhibited enough regularity to enable us to show that the solution space of the problem is infinite, thereby successfully closing the problem. From the mathematical point of view, the result is quite interesting as it shows that the generalization to a saturation operator partially independent from the reduction operator greatly adds to the expressive power of the system.

We have demonstrated through our work that combination of different reasoning systems where the user acts as a component within the system is very useful in solving complex mathematical problems. This active involvement of the user has helped us to discover new knowledge that enabled us to narrow down the search and compute solutions of greater magnitude. This novel approach of combining the user and reasoning systems could be applied to solve other complex mathematical problems where intermediate solution models can be computed and inspected by the user.

CHAPTER 13

FUTURE WORK

The work in this thesis has allowed us to propose a number of future directions which are discussed in this chapter.

13.1 Framework for Experimental Mathematics

Several environments and formalisms have been proposed previously for the combination and integration of mathematical software systems. Most of these systems follow a traditional automated theorem proving approach, in which a given conjecture is to be proved or refuted by the cooperation of different reasoning engines. However, there is a lack of support for experimental mathematics in which new conjectures can be constructed by an interleaved process of model computation, model inspection, property conjecture and verification using state-of-the-art symbolic reasoning systems.

One interesting future direction would be to create an experimental symbolic mathematics framework to create an environment that will enable a user to:

(i) provide high-level specification of experiments by combining

systems,

(ii) run experiments and inspect (intermediate) results,

(iii) re-use set-ups and results as parts of other systems,

(iv) generate efficient stand-alone systems from a specification.

13.2 Decomposition Techniques

Techniques to compose larger structures from smaller ones exist in most algebraic domains; for example in group theory, direct sums, semi-direct products, wreath products, etc. can be used to construct larger groups from smaller ones and it can be shown that these constructions either preserve or induce certain properties in the larger group. While some composition techniques, like direct sums, also exist for quasigroup and loop theory, their impact on modularising classifications is very limited due to exponential increase of numbers of structures for larger sizes. One interesting future direction will be to develop novel theory formation techniques to use given examples of small algebraic structures to find compositions for larger ones that preserve properties. For example, computing all substructures for large size bilattice models [Fit90a] is a computationally intensive task and novel algorithms that use composition techniques to compute larger substructures from smaller ones can reduce the search space by decreasing the amount of computations needed. Some basic definitions that pertain to bilattices are given as follows:

Definition 13.2.1 *A pre-bilattice is a structure* $\mathcal{B} = (B, \leq_t, \leq_k)$*, such that* B *is a set containing at least two elements, and* $(B, \leq_t)$*,* $(B, \leq_k)$ *are complete lattices.*

A negation of $\mathcal{B}$ *is a unary operation* $\neg$ *on* B *satisfying the following properties: (1)* $\neg\neg x = x$ *(2) if* $x \leq_t y$ *then* $x \geq_t y$*, (3) if* $x \leq_k y$ *then*

$\neg x \leq_k \neg y$.

Definition 13.2.2 *A bilattice is a structure* $\mathcal{B} = (B, \leq_t, \leq_k, \neg)$*, such that* $(B, \leq_t, \leq_k)$ *is a pre-bilattice with a negation* $\neg$*.*

As a result, one would not only gain additional computational methods for composing substructures, but also theoretical insights into additional ways of how to decompose large substructures into smaller ones. Generation of such substructures can help in finding counter models for certain construction theories for bilattices in order to prove that they are not correct. These would on the one hand be valuable results in their own right and on the other hand could also be exploited by automated reasoning techniques for example to further simplify proofs.

13.3 Other Generalizations of Kuratowski Problem

The generalized version of the Kuratowski's classical closure-complement problem from point-set topology is not the only possible generalization. One possible future direction is to investigate other typical examples of generalizations that can be obtained by considering other topological or set theoretical operators like union or intersection. Variations on the original problem and applications of the classification to the characterization of properties of subsets of a topological space can be found in [GJ08] that also contains a large bibliography on variants and applications of the problem. [WY00] shows that in locale theory there are exactly 21 different combinations. In our study, we have mapped out a landscape of generalized problems that lie between the finite classical and localic cases, where results were previously known, and the infinite generalized case, which is a new result. Also, formal proofs for the

generalized case with additional axiom 2 or 3 or 4 only given in Chapter 11 are still outstanding and further investigation is need for these cases. The results from our work demonstrate that our approach scales well to all the generalizations mentioned earlier. Indeed, the system we have implemented is fully parametric on the list of reduction rules of the advanced rewriting system. Furthermore, the computation of the Knuth-Bendix like completion that we perform manually could be easily automated.

APPENDIX A

EXPERIMENTAL RESULTS FOR QUASIGROUPS

This appendix presents the results of the experiments we have performed in the finite domain of quasigroups. The experiments were performed to evaluate the different techniques we have used with model generator (Mace4), constraint solver (Minion) and SAT solvers (MiniSat and zChaff) with a time limit of 2 hours for the generation of each quasigroup. The tables present the following information:

(i) A standard comparison of systems for generating quasigroups with two-variable properties is presented in Table A.1.

(ii) A comparison of systems using our element pre-computation approaches is presented in Table A.2.

(iii) A comparison of systems using implied constraints is presented in Tables A.3, A.4, A.5, A.6, A.7, A.8, and A.9.

In all of the tables presented in this appendix, times are represented in seconds.

Property	**Size**	**mace4** time [s]	**minion** time [s]	**minisat** time [s]	**zchaff** time [s]
Qg-1	4	0.01	0.45	0.001	0.00
	8	2.07	2.47	0.02	0.30
	9	0.64	1.02	0.04	0.034
	12	–	–	265.55	4535.47
	13	–	23.71	–	–
Qg-2	4	0.01	0.24	0.00	0.00
	5	0.01	0.24	0.002	0.00
	8	0.13	134.53	0.01	0.02
	9	0.01	1146.73	0.11	0.37
	12	10.76	–	171.29	–
	13	317.03	–	2229.24	–
Qg-3	3	0.01	0.24	0.00	0.00
	4	0.01	0.25	0.00	0.00
	7	1.73	0.28	0.00	0.03
	9	0.99	0.28	0.00	0.00
	10	–	–	141.77	215.93
	12	–	48.67	2886.82	–
Qg-4	4	0.01	0.55	0.00	0.00
	5	0.01	0.24	0.00	0.00
	9	0.02	0.25	0.02	0.017
	11	6.53	0.67	0.08	0.19
	13	0.01	0.38	2.39	13.01
	16	0.08	814.76	28.91	103.77
	17	0.07	20.51	7.43	132.36
	20	–	–	393.40	–
	21	–	–	248.81	–
Qg-5	3	0.01	0.24	0.00	0.00
	4	0.01	0.25	0.00	0.00
	5	0.01	0.26	0.00	0.00
	7	0.70	0.35	0.00	0.00
	8	92.22	1.83	0.00	0.07
	9	0.31	0.26	0.00	0.00
	11	–	–	0.33	6.68
	12	–	–	0.34	12.03
	13	–	–	1347.78	–
	15	–	–	44.94	594.76
	16	–	–	2912.9	3.39
	17	–	–	6410.52	–
Qg-6	5	0.00	0.25	0.00	0.00
	9	72.41	78.67	0.03	0.08
	13	–	–	0.13	1.44
	17	–	–	–	5994.71
Qg-7	4	0.01	0.25	0.00	0.00
	8	0.02	0.28	0.00	0.00
	9	0.06	0.56	0.00	0.00
	13	0.75	0.29	0.08	0.59
	16	–	–	10.05	171.63

Table A.1: Quasigroups found for particular properties using different systems.

P	S	Encoding 1								Encoding 2				Encoding 3	
		n				$n + \lceil \frac{n}{2} \rceil$								$n + \lceil \frac{n}{2} \rceil$	
		mace4	**minion**	**minisat**	**zchaff**	**mace4**	**minion**	**minisat**	**zchaff**	**mace4**	**minion**	**minisat**	**zchaff**	**mace4**	**minion**
		time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]
Qg-1	4	0.01	0.24	0.002	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	8	–	–	–	–	n/a	n/a	n/a	n/a	0.02	53.80	0.017	0.026	n/a	n/a
	9	0.03	0.47	0.01	0.017	n/a	n/a	n/a	n/a	0.49	33.28	0.01	0.005	n/a	n/a
	12	293.02	–	–	–	–	–	–	–	–	–	–	–	1041.45	–
	13	1133.54	–	865.94	–	–	–	–	–	–	–	–	–	–	–
Qg-2	5	0.01	0.25	0.001	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	8	0.01	0.29	0.006	0.001	n/a	n/a	n/a	n/a	0.08	33.28	–	0.014	n/a	n/a
	9	–	–	–	–	n/a	n/a	n/a	n/a	0.24	25.66	–	0.32	n/a	n/a
	12	387.00	3332.47	38.10	443.29	358.57	3899.29	7.31	270.43	73.51	–	50.83	1.39	15.64	–
	13	369.54	–	752.07	3237.47	4153.02	–	1281.8	–	5987.13	–	222.84	–	125.88	–
	16	–	–	–	–	–	–	–	–	–	–	–	–	3408.92	–
Qg-3	3	0.01	–	0.003	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	4	0.01	0.24	0.004	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	7	0.01	0.25	0.003	0.002	n/a	n/a	n/a	n/a	0.02	0.26	–	0.001	n/a	n/a
	9	0.10	0.26	0.009	0.045	n/a	n/a	n/a	n/a	96.39	4.03	–	0.002	n/a	n/a
	10	1.03	57.69	1.26	2.55	0.67	6.54	1.20	0.57	71.79	361.92	–	14.91	0.22	0.53
	12	509.24	6547.02	71.3	355.18	103.65	1630.76	23.73	–	286.83	49.32	63.88	1575.08	20.10	157.64
	13	2064.90	–	–	–	–	–	–	3233.3	1199.65	–	415.41	1396.2	247.51	–
Qg-4	4	0.01	0.26	0.002	–	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	5	0.00	0.25	0	0.001	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	9	–	–	–	–	n/a	n/a	n/a	n/a	0.01	7.13	0.022	0.005	n/a	n/a
	11	–	–	–	–	–	–	–	–	0.01	7.13	–	0.02	–	–
	19	–	–	–	–	–	–	–	–	–	–	–	1016.36	–	–
Qg-5	3	0.01	0.25	0.002	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	4	0.01	0.26	0	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	5	0.01	0.24	0.001	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	8	–	–	–	–	n/a	n/a	n/a	n/a	0.01	–	0.005	0.002	n/a	n/a
Qg-6	5	0.01	0.24	0.002	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	9	–	–	–	–	n/a	n/a	n/a	n/a	0.02	1.03	–	–	n/a	n/a
Qg-7	4	–	0.25	0.002	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	7	–	–	–	–	n/a	n/a	n/a	n/a	0.01	–	–	–	n/a	n/a
	8	–	–	–	–	n/a	n/a	n/a	n/a	–	3.56	–	0.001	n/a	n/a
	16	–	–	–	–	–	–	–	–	–	–	–	48.16	–	–

Table A.2: Quasigroups found for properties (P) of sizes (S) with algebraic pre-computations.

Qg-1						
Size	**Systems**	C_1 time [s]	C_2 time [s]	C_3 time [s]	C_4 time [s]	C_5 time [s]
4	**mace4**	0.00	0.01	0.00	0.01	0.00
	minion	0.24	0.25	0.25	0.26	0.27
	minisat	0.00	0.00	0.00	0.00	0.00
	zchaff	0.00	0.00	0.00	0.001	0.00
8	**mace4**	2.07	0.34	0.01	2.04	0.01
	minion	1.49	1.49	1.49	1.49	0.41
	minisat	0.01	0.00	0.01	0.00	0.00
	zchaff	0.09	0.11	0.01	0.04	0.02
9	**mace4**	0.64	45.06	0.07	0.60	–
	minion	0.63	0.63	0.64	0.63	–
	minisat	0.01	0.02	0.01	0.03	–
	zchaff	0.04	0.18	0.15	0.01	–
12	**mace4**	–	–	4788.10	–	821.16
	minion	–	–	–	–	–
	minisat	–	899.92	1158.99	2552.18	239.92
	zchaff	–	–	–	6535.26	–
13	**mace4**	–	–	–	–	55.85
	minion	9.51	9.99	9.51	9.51	32.26
	minisat	–	3187.06	6279.92	5203.53	–
	zchaff	–	–	–	–	–

Table A.3: QG-1 quasigroups found using implied constraints.

Qg-2						
Size	**Systems**	C_1 time [s]	C_2 time [s]	C_3 time [s]	C_4 time [s]	C_5 time [s]
4	**mace4**	0.01	0.01	0.01	–	–
	minion	0.25	0.25	0.25	–	–
	minisat	0.00	0.00	0.00	–	–
	zchaff	0.00	0.00	0.00	–	–
5	**mace4**	0.01	0.01	0.01	0.01	0.01
	minion	0.24	0.24	0.25	0.26	0.25
	minisat	0.00	0.00	0.00	0.00	0.00
	zchaff	0.00	0.00	0.00	0.00	0.00
8	**mace4**	0.13	0.13	0.01	–	–
	minion	47.05	65.79	42.85	–	–
	minisat	0.01	0.01	0.01	–	–
	zchaff	0.04	0.00	0.15	–	–
9	**mace4**	0.01	0.01	0.08	30.67	0.01
	minion	347.10	560.84	305.52	190.37	7.41
	minisat	0.05	0.32	0.01	0.01	0.01
	zchaff	0.07	2.37	0.82	0.02	0.09
12	**mace4**	10.76	10.99	158.31	–	–
	minion	–	–	–	–	–
	minisat	13.59	32.91	12.16	2739.27	2814.71
	zchaff	–	–	–	–	–
13	**mace4**	323.35	279.64	61.39	–	–
	minion	–	–	–	–	–
	minisat	1595.98	55.70	215.86	–	4264.09
	zchaff	–	2363.77	–	–	–

Table A.4: QG-2 quasigroups found using implied constraints.

Qg-3						
Size	**Systems**	C_1 time [s]	C_2 time [s]	C_3 time [s]	C_4 time [s]	C_5 time [s]
3	**mace4**	0.01	0.00	0.00	0.01	0.00
	minion	0.16	0.22	0.22	0.23	–
	minisat	0.00	0.00	0.00	0.00	–
	zchaff	0.00	–	–	–	–
4	**mace4**	0.01	0.00	0.00	0.01	0.00
	minion	0.16	0.23	0.22	0.22	0.16
	minisat	0.00	0.00	0.00	0.00	0.00
	zchaff	0.00	0.00	–	–	–
7	**mace4**	1.71	0.13	0.02	0.45	0.01
	minion	0.18	0.23	0.86	0.24	0.20
	minisat	0.00	0.00	0.00	0.00	0.00
	zchaff	0.03	0.00	–	–	–
9	**mace4**	61.85	0.12	0.24	22.31	
	minion	0.19	–	0.25	0.24	–
	minisat	0.03	0.02	0.14	0.00	–
	zchaff	1.02	0.00	–	–	–
10	**mace4**	–	–	87.74	–	4.63
	minion	–	–	–	–	4.24
	minisat	31.38	33.67	134.83	79.10	2.18
	zchaff	956.92	164.43	–	–	–
12	**mace4**	–	–	2048.24	–	–
	minion	35.47	–	–	–	–
	minisat	1407.15	7086.15	3666.31	2949.97	428.07
	zchaff	–	–	–	–	–

Table A.5: QG-3 quasigroups found using implied constraints.

Qg-4						
Size	**Systems**	C_1 time [s]	C_2 time [s]	C_3 time [s]	C_4 time [s]	C_5 time [s]
4	**mace4**	0.00	0.00	0.01	0.01	0.00
	minion	0.25	0.32	0.18	2.47	0.44
	minisat	0.00	0.00	0.00	0.00	0.00
	zchaff	0.00	0.00	0.00	0.00	0.00
5	**mace4**	0.01	0.00	0.01	0.01	0.00
	minion	0.28	0.28	0.25	0.30	0.54
	minisat	0.00	0.00	0.00	0.00	0.00
	zchaff	0.00	0.00	0.00	0.00	0.00
9	**mace4**	0.02	0.01	0.01	0.02	0.01
	minion	0.39	0.33	0.32	0.49	0.30
	minisat	0.01	0.01	0.01	0.01	0.01
	zchaff	0.01	0.00	0.00	0.02	0.00
11	**mace4**	6.41	0.83	0.01	0.83	0.01
	minion	0.49	0.69	0.51	0.63	0.61
	minisat	0.03	0.04	0.03	0.03	0.03
	zchaff	0.10	0.13	0.05	0.03	0.01
13	**mace4**	0.01	0.01	1388.43	0.01	2633.77
	minion	0.39	0.36	0.27	2.48	0.55
	minisat	5.78	6.41	1.38	1.06	7.49
	zchaff	25.86	116.25	123.13	85.33	59.85
16	**mace4**	0.06	0.08	–	0.05	40.33
	minion	792.95	585.70	582.39	571.11	18.93
	minisat	4.71	25.77	8.41	2.28	14.05
	zchaff	1437.89	696.45	4251.49	2613.16	5422.00
17	**mace4**	0.06	0.07	–	0.04	1743.09
	minion	12.96	14.48	17.79	6.76	0.45
	minisat	5.34	63.51	3.74	14.87	0.33
	zchaff	–	1756.19	–	–	4594.64
19	**mace4**	–	–	31.92	–	–
	minion	–	–	–	–	–
	minisat	502.19	1317.75	3414.92	1184.78	10.72
	zchaff	–	–	–	–	–
20	**mace4**	–	–	6801.47	–	–
	minion	–	–	–	–	–
	minisat	1944.21	–	1470.21	–	–
	zchaff	–	–	–	–	0.33
21	**mace4**	–	–	–	–	–
	minion	–	–	–	–	–
	minisat	1.60	1.33	1.80	1.39	1.12
	zchaff	–	–	–	–	0.42

Table A.6: QG-4 quasigroups found using implied constraints.

Qg-5							
Size	**Systems**	C_1 time [s]	C_2 time [s]	C_3 time [s]	C_4 time [s]	C_5 time [s]	C_6 time [s]
3	**mace4**	0.00	–	0.01	–	–	0.01
	minion	0.18	–	0.23	–	–	0.25
	minisat	0.00	–	0.00	–	–	0.00
	zchaff	0.00	–	0.00	–	–	0.00
4	**mace4**	–	–	–	–	–	0.01
	minion	–	–	–	–	–	0.24
	minisat	–	–	–	–	–	0.00
	zchaff	–	–	–	–	–	0.00
5	**mace4**	0.00	0.00	0.00	0.01	0.00	0.01
	minion	0.18	0.27	0.24	0.16	0.23	0.24
	minisat	0.00	0.00	0.00	0.00	0.00	0.00
	zchaff	0.00	0.00	0.00	0.00	0.00	0.00
7	**mace4**	0.47	–	0.03	0.01	0.00	0.09
	minion	0.35	–	0.30	0.16	0.25	0.31
	minisat	0.00	–	0.00	0.00	0.00	0.00
	zchaff	0.08	–	0.00	0.00	0.00	0.00
8	**mace4**	61.59	0.07	0.00	0.04	0.01	3.40
	minion	1.10	0.57	0.82	0.19	0.29	1.03
	minisat	0.01	0.00	0.00	0.00	0.00	0.02
	zchaff	0.16	0.00	0.00	0.00	0.00	0.00
9	**mace4**	0.20	–	29.56	–	–	0.25
	minion	0.20	–	0.25	–	–	0.26
	minisat	0.04	–	0.00	–	–	0.00
	zchaff	0.00	–	0.04	–	–	0.09
11	**mace4**	–	–	20.58	667.29	2993.49	–
	minion	–	6173.77	–	1537.45	282.09	–
	minisat	0.36	0.13	0.07	0.12	0.06	0.09
	zchaff	0.45	0.04	1.63	0.22	0.04	0.06
12	**mace4**	–	–	–	–	–	–
	minion	–	–	–	–	–	–
	minisat	–	–	–	–	–	0.59
	zchaff	–	–	–	–	–	5.91
13	**mace4**	–	–	–	–	–	–
	minion	–	–	–	–	–	–
	minisat	74.62	–	–	–	–	18.27
	zchaff	241.20	–	–	–	–	1233.07
15	**mace4**	–	–	–	–	–	–
	minion	–	–	–	–	–	–
	minisat	0.45	–	14.45	–	–	1.94
	zchaff	775.36	–	21.43	–	–	1804.82
16	**mace4**	–	–	–	–	–	–
	minion	–	–	–	–	–	–
	minisat	–	–	–	–	–	8.24
	zchaff	–	–	–	–	–	–
20	**mace4**	–	–	–	–	–	–
	minion	–	–	–	–	–	–
	minisat	–	–	–	–	–	706.39
	zchaff	–	–	–	–	–	–

Table A.7: QG-5 quasigroups found using implied constraints.

Qg-6						
Size	**Systems**	C_1 time [s]	C_2 time [s]	C_3 time [s]	C_4 time [s]	C_5 time [s]
5	**mace4**	0.00	0.01	0.00	0.00	0.01
	minion	0.17	0.16	0.23	0.17	0.24
	minisat	0.00	0.00	0.00	0.00	0.00
	zchaff	0.00	0.00	0.00	0.00	0.00
9	**mace4**	47.62	7.33	9.27	15.02	0.17
	minion	54.08	54.66	54.76	27.09	6.64
	minisat	0.00	0.02	0.00	0.00	0.01
	zchaff	0.11	0.02	0.03	0.04	0.03
13	**mace4**	–	–	–	–	–
	minion	–	–	–	–	–
	minisat	7.02	1.65	0.52	1.15	1.07
	zchaff	51.51	58.48	13.72	311.60	394.45
21	**mace4**	–	–	–	–	–
	minion	–	–	–	0.40	0.41
	minisat	–	–	–	–	0.26
	zchaff	–	–	–	–	–

Table A.8: QG-6 quasigroups found using implied constraints.

Qg-7						
Size	**Systems**	C_1 time [s]	C_2 time [s]	C_3 time [s]	C_4 time [s]	C_5 time [s]
4	**mace4**	0.01	0.01	0.01	0.00	0.01
	minion	0.15	0.26	0.24	0.56	0.25
	minisat	0.00	0.00	0.00	0.00	0.00
	zchaff	0.00	0.00	0.00	0.00	0.00
8	**mace4**	0.01	0.05	0.01	0.01	0.01
	minion	4.63	0.29	0.37	0.27	0.26
	minisat	0.00	0.00	0.00	0.00	0.00
	zchaff	0.00	0.00	0.00	0.00	0.00
9	**mace4**	0.07	1.07	2.05	0.05	0.01
	minion	0.39	0.52	0.49	0.55	0.31
	minisat	0.00	0.01	0.00	0.00	0.00
	zchaff	0.01	0.00	0.00	0.00	0.00
13	**mace4**	0.49	–	–	0.33	–
	minion	0.24	0.33	0.32	0.23	0.28
	minisat	0.09	2.66	0.38	0.032	0.38
	zchaff	2.64	4.94	68.00	0.84	2.07
16	**mace4**	–	–	–	–	–
	minion	–	–	–	–	1290.93
	minisat	82.50	1383.55	79.68	–	240.70
	zchaff	4039.93	–	–	–	–

Table A.9: QG-7 quasigroups found using implied constraints.

LIST OF REFERENCES

[AHMCS12a] O. Al-Hassani, Q. Mahesar, C.S. Coen, and V. Sorge. Solving Kuratowski problems by term rewriting. In *Proceedings of the 19th Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*, 2012. 1.4

[AHMCS12b] O. Al-Hassani, Q. Mahesar, C.S. Coen, and V. Sorge. A term rewriting system for Kuratowski's closure-complement problem. In *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications*, 2012. 1.4

[BHF96] A. Buch, Th. Hillenbrand, and R. Fettig. WALDMEISTER: High performance equational theorem proving. In *Proceedings of the 4th International Symposium on Design and Implementation of Symbolic Computation Systems*, pages 63–64, 1996. 3.2.1

[Bie08] A. Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):75–97, 2008. 3.2.4

[BK11] F. Blanqui and A. Koprowski. CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 21(4):827–859, 2011. 3.2.2

[BKVV08] M. Bravenboer, K. T. Kalleberg, R. Vermaas, and E. Visser. Stratego/xt 0.17. a language and toolset for program transformation. *Sci. Comput. Program.*, 72(1-2):52–70, June 2008. 3.2.2

[BL07] F. E. Bennett and C. C. Lindner. *Quasigroups, in: The CRC Handbook of Combinatorial Designs*. CRC Press, 2007. 4.2

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, Cambridge, 1998. 10.1

[Bro10] J.L. Brown. Introductory topology, 2010. http://www.ces.clemson.edu/~jimlb/Teaching/2009-10/Math986/Topology.pdf. (document), 8.1

[BS07] B. Benhamou and M. R. Saidi. Eliminating local symmetry in CSP. In *Proceedings of the International Symmetry Conference*, 2007. 2.2

[Buc06] B. Buchberger. Mathematical theory exploration. In *Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2006. 1.1

[Bun85] A. Bundy. Discovery and reasoning in mathematics. In Aravind K. Joshi, editor, *IJCAI*, pages 1221–1230. Morgan Kaufmann, 1985. 1.1

[BZ92] F. E. Bennett and L. Zhu. Conjugate orthogonal latin squares and related structures. In *J. H. Dinitz & D. R. Stinson (eds): Contemporary Design Theory: A Collection of Surveys*. John Willey & Sons, 1992. 2.1, 4.2

[CCM06] J. Charnley, S. Colton, and I. Miguel. Automatic generation of implied constraints. In *Proceedings of the 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva Del Garda, Italy*, pages 73–77, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press. 2.1, 4.2, 7.1.4, 12.2

[CDS91] J. Cooper, D. Donovan, and J. Seberry. Latin squares and critical sets of minimal size. *Australas. J. Combin.*, 4:113–120, 1991. Combinatorial mathematics and combinatorial computing (Palmerston North, 1990). 4.2

[CFS08] S. Conchon, J. Filliâtre, and J. Signoles. Designing a generic graph library using ML functors. In *Proceedings of the Ninth Symposium on Trends in Functional Programming*, volume 8, pages 124–140. Intellect, 2008. 11.2

[CLW96] B. M. W. Cheng, J. H. M. Lee, and J. C. K. Wu. Speeding up constraint propagation by redundant modeling. In *Proceedings of the 2nd Int. Conf. on Principles and Practice of Constraint Programming*, pages 91–103. Springer Verlag, 1996. 2.2

[CM01] S. Colton and I. Miguel. Constraint generation via automated theory formation. In *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 575–579. Springer, 2001. 2.1, 4.2, 7.1.4, 12.2

[CMSM04] S. Colton, A. Meier, V. Sorge, and R. L. McCasland. Automatic generation of classification theorems for finite algebras. In *IJCAR*, volume 3097 of *Lecture Notes in Computer Science*, pages 400–414. Springer, 2004. 2.4, 6.2, 6.2.1, 11

[Coe06] C. S. Coen. A semi-reflexive tactic for (sub-)equational reasoning. In *Proceedings of the 2004 International Conference on Types for Proofs and Programs*, TYPES'04, pages 98–114. Springer-Verlag, 2006. 10.1

[Col02a] S. Colton. *Automated Theory Formation in Pure Mathematics*. Springer, 2002. 2.1, 2.3

[Col02b] S. Colton. The HR program for theorem generation. In *Proceedings of the Eighteenth Conference on Automated Deduction*, pages 285–289. Springer, 2002. 2.4

[Cor06] L. Corsi. Combinazioni di operatori di interno, chiusura e loro complemento in LJ. Tesi di laurea in Matematica, Università di Padova, 2006. 9.1, 10.2

[CP05] S Colton and A Pease. The TM system for repairing non-theorems. In *Selected papers from the IJCAR'04*

disproving workshop, Electronic Notes in Theoretical Computer Science, volume 125(3), pages 13–26. Elsevier, 2005. 2.3

[DdVC03a] I. Dotú, A. del Val, and M. Cebrián. Channeling constraints and value ordering in the quasigroup completion problem. In *Proc. of IJCAI-2003*, pages 1372–1373. Morgan Kaufmann, 2003. 2.2, 5.1

[DdVC03b] I. Dotú, A. del Val, and M. Cebrián. Redundant modeling for the quasigroup completion problem. In *Principles and practice of Constraint Programming (CP03) Lecture Notes in Computer Science*, pages 288–302. Springer, 2003. 5.1

[Dec03] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. 2.3

[DJKK12] A. Distler, C. Jefferson, T. Kelsey, and L. Kotthoff. The semigroups of order 10. In *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 883–899. Springer Berlin Heidelberg, 2012. 2.3

[DK09] A. Distler and T. Kelsey. The monoids of orders eight, nine & ten. *Annals of Mathematics and Artificial Intelligence*, 56(1):3–21, May 2009. 2.3

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962. 3.2.4

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960. 2.1, 3.2.4

[DSS11] A. Distler, M. Shah, and V. Sorge. Enumeration of AG-groupoids. In *Proceedings of the 18th Calculemus and 10th international conference on Intelligent computer mathematics*, MKM'11, pages 1–14, Berlin, Heidelberg, 2011. Springer-Verlag. 2.3, 11

[Eat08] J. W. Eaton. *Gnu Octave Manual*. Network Theory Ltd., 3rd edition, 2008. 3.3, 9.1, 11, 11.2, 11.3

[ES03] N. Eén and N. Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. 3.2.4, 7.1

[Eva51] T. Evans. On multiplicative systems defined by generators and relations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 47:637–649, 10 1951. 6.2

[FGJ+07] A. M. Frisch, M. Grum, C. Jefferson, B. M. Hernández, and I. Miguel. The design of essence: a constraint language for specifying combinatorial problems. In *Proceedings of IJCAI-07*, pages 80–87, 2007. 3.2.3, 5.1, 7.1

[Fit90a] M. Fitting. Bilattices in logic programming. In *Proc. of the 20th Int. Symp. on Multiple-Valued Logic*, pages 63–70. IEEE Press, 1990. 13.2

[Fit90b] M. Fitting. *First-order logic and automated theorem proving*. Springer-Verlag New York, Inc., 1990. 3.1, 3.1.2

[FMM] Z. Fu, Y. Marhajan, and S. Malik. zChaff SAT Solver. http://www.princeton.edu/~chaff/zchaff.html. 3.2.4, 7.1

[FSB93] M. Fujita, J. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. In *IJCAI'93: Proceedings of the 13th international joint conference on Artifical intelligence*, pages 52–57, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. 2.1

[Gal85] Jean H. Gallier. *Logic for computer science: foundations of automatic theorem proving*. Harper & Row Publishers, Inc., 1985. 3.2.1

[GAP08] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4.12*, 2008. 2.3, 2.4

[GJ08] B. J. Gardner and M. Jackson. The Kuratowski closure-complement theorem. *New Zealand Journal of Mathematics*, 38:9–44, 2008. 8.2, 8.2, 9, 13.3

[GJM06] I. P. Gent, C. Jefferson, and I. Miguel. Minion: A fast, scalable, constraint solver. In *Proceeding of the 2006 conference on ECAI 2006*, pages 98–102. IOS Press, 2006. 2.3, 3.2.3, 7.1

[GKL+05] I. P. Gent, T. Kelsey, S. A. Linton, I. McDonald, I. Miguel, and B. M. Smith. Conditional symmetry breaking. In *Proceedings of the Principles and Practice of Constraint Programming - CP 2005, LNCS 3709*, pages 256–270. Springer, 2005. 2.2

[Gog98] J. A. Goguen. Stretching first order equational logic: Proofs with partiality, subtypes and retracts. In *Proceedings of the Workshop on First Order Theorem Proving*, page pages, 1998. 3.2.2

[Gra] Graphviz - graph visualization software. http://www.graphviz.org/. 11.2

[GS97] C. P. Gomes and B. Selman. Problem structure in the presence of perturbations. In *Proceedings of the 14th National Conference on AI*, pages 221–226. AAAI Press, 1997. 2.2

[GS02a] C. P. Gomes and D. B. Shmoys. The promise of LP to boost CSP techniques for combinatorial problems, 2002. 2.2

[GS02b] C. P. Gomez and D. Shmoys. Completing quasigroups or latin squares: a structured graph coloring problem. In *Proceedings of Computational Symposium on Graph Coloring and Generalization*, 2002. 2.2, 5.2

[HAF01] M. R. Holmes and J. Alves-Foss. The Watson theorem prover. *J. Autom. Reason.*, 26(4):357–408, May 2001. 3.2.2

[Heb08] E. Hebrard. Mistral, a constraint satisfaction library. In *Proceedings of the Third International CSP Solver Competition*, 2008. 3.2.3

[HKK91] M. Hermann, C. Kirchner, and H. Kirchner. Implementations of term rewriting systems. *The Computer Journal*, 34(1):20, 1991. 3.2.2

[HKPM04] G. Huet, G. Kahn, and Ch. Paulin-Mohring. *The* Coq *Proof Assistant - A tutorial - Version 8.0*, April 2004. 3.2.2

[HO80] G. Huet and D. C. Oppen. Equations and rewrite rules: a survey. Technical report, Stanford, CA, USA, 1980. 3.2.2

[Hor74] J.D Horton. Sub-latin squares and incomplete orthogonal arrays. *Journal of Combinatorial Theory, Series A*, 16(1):23 – 33, 1974. 2.1

[HS74] A. Hedayat and E. Seiden. On the theory and application of sum composition of latin squares and orthogonal latin squares. 1974. 2.1

[Kel75] J. L. Kelley. *General Topology*. Graduate Texts in Mathematics. Springer, 1975. 8.1

[Klo87] J. W. Klop. Term rewriting systems: a tutorial. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 32:143–182, 1987. 3.2.2

[KRA+01] H. Kautz, Y. Ruan, D. Achlioptas, C. Gomes, B. Selman, and M. Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, pages 351–358. Morgan Kaufmann Publishers Inc., 2001. 5.2

[KZ95] D. Kapur and H. Zhang. An overview of rewrite rule laboratory. *J. of Computer and Mathematics with Applications*, 29:91–114, 1995. 3.2.2

[Lab00] F. Laburthe. Choco: implementing a CP kernel. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems*, page 7185, 2000. 2.1, 3.2.3, 7.1.4

[Lip65] S. Lipschutz. *Schaum's outline of theory and problems of general topology*. Schaum's outline series. Schaum Pub. Co., 1965. 8.1

[McC] W. McCune. http://www.cs.unm.edu/~mccune/prover9/. 3.2.1, 7.1

[Mcc94] W. Mccune. A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical report, Mathematics and Computer Science Division Argonne National Laboratory, 1994. 2.1

[McC03a] W. McCune. Mace4 reference manual and guide. *CoRR*, cs.SC/0310055, 2003. 2.3, 2.4, 3.2.5, 5.3, 7.1, 7.1

[McC03b] W. McCune. Otter 3.3 reference manual. *CoRR*, cs.SC/0310056, 2003. 2.1, 2.3, 3.2.1

[MGA97] S. Markovski, D. Gligoroski, and S. Andova. Using quasigroups for one-one secure encoding. In *Proc. VIII Conf. Logic and Computer Science LIRA 97, Novi Sad*, pages 157–162, 1997. 4.1

[ML07] R. Martins and I. Lynce. Breaking local symmetries in quasigroup completion problems. Technical report, IST/INESC-ID, Technical University of Lisbon, Portugal, 2007. 2.2

[MLB01] S. Merchez, C. Lecoutre, and F. Boussemart. Abscon: A prototype to solve CSPs with abstraction. In *CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 730–744, London, UK, 2001. Springer-Verlag. 3.2.3

[MS05] A. Meier and V. Sorge. Applying SAT Solving in the Classification of Finite Algebras. *JAR*, 35(1–3):201–235, 2005. 2.4

[MS09] Q. Mahesar and V. Sorge. Classification of quasigroup-structures with respect to their cryptographic properties. In *Proceedings of the 16th Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*, 2009. 1.4

[MS10] Q. Mahesar and V. Sorge. Property preserving generation of large size quasigroup-structures. In *Proceedings of the 17th Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*, 2010. 1.4

[MS12a] Q. Mahesar and V. Sorge. Algebraic theory exploration: A comparison of technologies. In *Proceedings of the 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2012. 1.4

[MS12b] Q. Mahesar and V. Sorge. Generation of large size quasigroup structures using algebraic constraints. In *Proceedings of the 19th Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*, 2012. 1.4

[Mun00] J. R. Munkres. *Topology*. Prentice Hall, Incorporated, 2000. 8.1

[MW98] P. Meseguer and T. Walsh. Interleaved and discrepancy based search. In *Proceedings of ECAI-98*, pages 239–243. Wiley, 1998. 2.2

[NV] G. P. Nagy and P. Vojtechovsky. `http://www.math.du.edu/loops/doc/htm/CHAP007.htm`. 4.2

[PD07] K. Pipatsrisawat and A. Darwiche. Rsat 2.0: Sat solver description. Technical report, University of California, Los Angeles, 2007. 3.2.4

[Pfl90] H. O. Pflugfelder. *Quasigroups and Loops: Introduction*, volume 7 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, Germany, 1990. 4.1

[Pig75] D. Pigozzi. Equational logic and equational theories of algebras. Technical report, Purdue University, 1975. 3.1.3

[Qui93] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993. 2.4

[Rég94] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the twelfth national conference on Artificial Intelligence (vol. 1)*, AAAI '94, pages 362–367, 1994. 2.2, 5.1

[Ren] A. Rendl. Tailor - tailoring constraint models to constraint solvers. http://www.cs.st-andrews.ac.uk/~andrea/tailor/. 3.2.3, 7.1

[RM05] A. Ramani and I. L. Markov. Automatically exploiting symmetries in constraint programming. In *Proceedings of the 2004 Joint ERCIM/CoLOGNET International Conference on Recent Advances in Constraints*, CSCLP'04, pages 98–112. Springer-Verlag, 2005. 2.2

[RV01] A. Riazanov and A. Voronkov. Vampire 1.1. In Rejeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, volume 2083 of *LNAI*, pages 376–380, Siena, Italy, 2001. Springer Verlag. 3.2.1

[SA08a] J. Slaney and A. Ali. Counting loops with the inverse property. *Quasigroups and Related Systems*, 16(1):13–16, 2008. 2.3

[SA08b] J. Slaney and A. Ali. Generating loops with the inverse property. In *Proceedings of the Workshop on Empirically Successful Automated Reasoning for Mathematics (ESARM) : CICM 2008, Conferences on Intelligent Computer Mathematics, University of Birmingham, UK.*, 2008. 2.3

[Sam03] G. Sambin. Some points in formal topology. *Theoretical Computer Science*, 305(1-3):347–408, 2003. 9.1

[Sam12] G. Sambin. *The Basic Picture: a structural basis for constructive topology*. Oxford University Press, 2012. 10.3, 11.5

[Sch02] S. Schulz. E: A Brainiac theorem prover. *Journal of AI Communication*, 15(2–3):111–126, 2002. 3.2.1

[SCMM08] V. Sorge, S. Colton, R. McCasland, and A. Meier. Classification results in quasigroup and loop theory via a combination of automated reasoning tools. *Commentationes Mathematicae Universitatis Carolinae*, 49(2):319–339, 2008. (document), 2.4, 2.1

[SFS95] J. Slaney, M. Fujita, and M. Stickel. Automated reasoning and exhaustive search: Quasigroup existence problems. *Computers & Mathematics with Applications*, 29(2):115 – 132, 1995. 2.1

[Sla94] J. K. Slaney. Finder: Finite domain enumerator - system description. In *Proceedings of the 12th International Conference on Automated Deduction*, CADE-12, pages 798–801. Springer-Verlag, 1994. 2.3, 3.2.5

[Smi06] J. D. H. Smith. *An Introduction to Quasigroups and Their Representations (Studies in Advanced Mathematics)*. Chapman and Hall/CRC, 1 edition, November 2006. 4.1

[SML96] Bart Selman, David Mitchell, and Hector Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81:17–29, 1996. 3.2.4

[SMMC08] V. Sorge, A. Meier, R. Mccasland, and S. Colton. Automatic construction and verification of isotopy invariants. *Journal of Automated Reasoning*, 40(2-3):221–243, 2008. 2.4

[SS] G. Sutcliffe and C. Suttner. `http://www.cs.miami.edu/~tptp/`. 2.3

[SSW98] P. Shaw, K. Stergiou, and T. Walsh. Arc consistency and quasigroup completion. In *Proceedings of the ECAI-98 workshop on non-binary constraints*, 1998. 2.2, ii, viii

[Sti] M. E. Stickel. Quasigroup generator. http://www.ai.sri.com/~stickel/qga.lsp. 5.2, 7.1

[Sut] G. Sutcliffe. http://www.cs.miami.edu/~tptp/OverviewOfATP.html. 3.2.1

[SZ95] M. E. Stickel and H. Zhang. Studying quasigroup identities by rewriting techniques: Problems and first results. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, RTA '95, pages 450–456, London, UK, UK, 1995. Springer-Verlag. 2.1

[Ter03] Terese. *Term Rewriting Systems by "Terese"*, volume 5 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003. 3.2.2

[Wal01] T. Walsh. Permutation problems and channelling constraints. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2250 of *Lecture Notes in Computer Science*, pages 377–391. Springer Berlin Heidelberg, 2001. 2.2, 5.1

[WBH+99] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic. Spass version 2.0. In *Proc. CADE-18*, pages 275–279. Springer, 1999. 2.4, 3.2.1

[WY00] H. Wei and Z. Yasoming. Interior and boundary in a locale. *Advances in Mathematics*, 29(5):439–443, 2000. 9, 10.3, 13.3

[ZH94] H. Zhang and J. Hsiang. Solving open quasigroup problems by propositional reasoning. In *Proceedings of the International Computer Symp*, 1994. 2.1

[ZS94] H. Zhang and M.E. Stickel. *Implementing the Davis-Putnam Algorithm by Tries*. Technical report. Iowa State University, Department of Computer Science, 1994. 2.1

[ZZ01] Q. Zhang and Qi Zhang. SEM User's Guide. Technical report, Department of Computer Science, University of Iowa, 2001. 3.2.5