FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Marek Dančo

The Application of SAT Solving to Finite Model Finding

Department of Algebra

Supervisor of the master thesis: Mgr. Mikoláš Janota, Ph.D.
Study programme: Mathematics for Information Technologies

Prague 2024

I want to express my heartfelt gratitude to *Mgr. Mikoláš Janota, Ph.D.* for the extensive time and effort he dedicated to supervising this thesis. His commitment has been truly invaluable, and I am deeply thankful for his support.

Title: The Application of SAT Solving to Finite Model Finding

Author: Marek Dančo

Department: Department of Algebra

Supervisor: Mgr. Mikoláš Janota, Ph.D., Department of Algebra

Abstract: We propose a novel SAT-based approach to calculating the number of non-isomorphic algebraic structures of a given type, a significant challenge for current automated tools. Our program uses canonizing sets to build compact lexleader symmetry breaking constraints, enabling the construction of propositional formulas solved with state-of-the-art SAT solvers. In this thesis, we apply this method to effectively identify all non-isomorphic models across finite algebraic structures with a single binary operation, including structures such as semigroups and loops. We provide an implementation of our program and evaluate it on various such structures. The experimental results demonstrate the efficacy of our approach, as we successfully computed previously unknown counts for certain structures, highlighting its potential to address complex enumeration problems.

Keywords: first order logic SAT solving finite model

Název práce: Aplikace SAT řešičů pro hledání konečných modelů

Autor: Marek Dančo

Katedra: Katedra algebry

Vedoucí diplomové práce: Mgr. Mikoláš Janota, Ph.D., Katedra algebry

Abstrakt: Predstavujeme nový prístup k výpočtu neizomorfných algebraických štruktúr daného typu, založený na probléme splniteľnosti SAT. Enumerácia všetkých neizomorfných modelov je významným problémom pre dnešné automatické nástroje. Náš program využíva kanonizujúce množiny na vytvorenie kompaktných lexleader obmedzujúcich podmienok na odstránenie symetrií. Tým dosiahneme vytvorenie výrokových formulí, ktorých riešenie nám sprostredkúvajú moderné SAT solvery. V tejto diplomovej práci využívame túto metódu na efektívnu identifikáciu neizomorfných modelov naprieč konečnými algebraickými štruktúrami s jednou binárnou operáciou, ako sú napríklad semigrupy a lupy. Ponúkame implementáciu nášho programu, ktorú vyhodnotíme na rôznych týchto štruktúrach. Naše výsledky preukazujú účinnosť tohto prístupu, pomocou ktorého sme vypočítali doteraz neznáme počty určitých štruktúr, čím poukazujeme na potenciál tohto prístupu riešiť komplexné výpočetné problémy.

Klíčová slova: logika prvního řádu problém splnitelnosti konečné modely

Contents

Introduction **6**

1 Preliminaries **7**

1.1 Cayley Tables 9

2 MACE-style Model Finding **11**

2.1 Flattening 12

2.2 Grounding 13

2.3 Clause Splitting 14

2.4 Symmetry Breaking 16

2.4.1 The Least Number Heuristic 17

2.5 Finding All Non-isomorphic Models 18

3 Lexleader Models **19**

3.1 Canonical Forms 19

3.2 Encoding Lexleader to SAT 21

3.2.1 Encoding of the Total Order 21

3.2.2 Encoding of the Symmetry Breaking Constraint 22

3.3 Insight into Complexity 25

3.4 Transpositions 26

4 Canonizing Sets **27**

4.1 Computing Canonizing Sets 27

4.2 SAT Based Implementation 29

4.2.1 Computation Algorithm 30

4.2.2 Reduction Algorithm 32

4.3 Size of Canonizing Sets 34

5 Experiments **35**

5.1 Used Algebraic Structures 35

5.2 Results 37

Conclusion **43**

Bibliography **45**

List of Figures **50**

List of Tables **51**

A Attachments **52**

A.1 Repository 52

Introduction

In this thesis, we focus on the automatic counting and enumeration of all finite algebraic structures with a single binary operation, which are closed under the operation. These structures are commonly referred to as magmas (or, more rarely, groupoids). We utilize the axiomatic definitions of these structures, expressed by a first-order equational theory. The objective is to implement a program that, given the axioms of a magma and a small natural number n, enumerates or counts all magmas of order n, up to isomorphism, that satisfy the given axioms. For the given problem, we aim to provide an encoding in the language of propositional logic in conjunctive normal form. The resulting formula is such that from all of its satisfying assignments, we can construct all non-isomorphic magmas satisfying the input axioms.

The problem of counting all satisfying assignments to a SAT formula (known as #SAT) is a well-known #P-complete problem. As a consequence of Toda's theorem, if we could solve #SAT in polynomial time, we would also be able to solve in polynomial time every problem in the polynomial hierarchy. Therefore, we can deduce that the enumeration of all satisfying assignments in the general case is intractable. However, with our program, we were able to enumerate the correct numbers of several well known magmas for small orders. Additionally, we present the enumeration of magmas for orders that remain unknown to this day.

Counting algebraic structures is of interest to mathematicians. The On-line Encyclopedia of Integer Sequences (OEIS) maintains a record of the currently known counts of various algebraic structures. The history of counting *Latin squares* dates back to at least the 18th century. Euler knew the number of reduced Latin squares of order 5 [Eul82], and this number was also known to Cayley [Cay90].

We begin by introducing the basic concepts of first-order logic in the Preliminaries. This includes setting the notation used in the thesis and providing concepts and definitions related to Cayley tables of magmas. In Chapter 2, we describe the method by which we encode the model finding problem into SAT. This method, known as MACE, is adapted to our problem using some modifications presented in Paradox. Chapter 3 details our approach to avoiding the search for isomorphic solutions. We employ the well known lexleader method for canonically labeling algebraic structures, which, to our knowledge, has not been previously used in the context of MACE-style model finding. Chapter 4 includes a modification to the lexleader method, enhancing its effectiveness. Originally used in the context of graph search problems, canonizing sets make the lexleader method more compact. Finally, we present our experimental results in the last chapter.

1 Preliminaries

Throughout the thesis, we work with first-order logic (FOL) with equality. Let us begin by introducing basic terminology and notation used in the thesis. *Arity* is a function, assigning to each function and predicate symbol the fixed number of arguments it takes. We use a fixed *signature* (sometimes called *language*) $\Sigma = \langle \Sigma_p, \Sigma_f, ar \rangle$ consisting of a disjoint union of:

- a set Σ_p of predicate symbols, each $P \in \Sigma_p$ has associated arity $ar(P) \in \mathbb{N}_0$, we use one special predicate symbol $=$ of arity 2, representing equality,
- a set Σ_f of function symbols, each $F \in \Sigma_f$ has associated arity $ar(F) \in \mathbb{N}_0$.

Constants are regarded as function symbols of arity 0. *Variables* form a countable set disjoint from Σ. *Terms* are defined inductively:

- every variable is a term,
- whenever $F \in \Sigma_f$ is of arity n and $t_1, \ldots, t_n$ are terms, then $F(t_1, \ldots, t_n)$ is a term.

A *ground term* is a term that does not contain any variables. An *atom* is defined inductively:

- whenever t_1, t_2 are terms, $t_1 = t_2$ is an atom,
- whenever $P \in \Sigma_p$ is of arity n and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is an atom.

A *literal* is an atom, or its negation. Negated atoms are written using the negation operator $\neg$ as $\neg A$ for $A \in \Sigma_p$ and negated equalities as $t_1 \neq t_2$ for terms t_1, t_2. A *formula* is defined inductively:

- every atom is a formula (sometimes called an *atomic formula*),
- whenever φ, ψ are formulas, then $(\varphi \wedge \psi), (\varphi \vee \psi), \neg\varphi, (\varphi \Rightarrow \psi), (\varphi \Leftrightarrow \psi)$, *true*, *false* are formulas,
- whenever φ is a formula, x is a variable, then $\exists x\, \varphi, \forall x\, \varphi$ are formulas.

A variable is called *free* if it is not in the scope of any quantifier, otherwise it is called *bound.* A formula is *closed* if it has no free variables. A *theory* is a set of closed formulas. A *clause* is a set of literals connected by disjunction. All variables in a clause not in the scope of any quantifier are implicitly universally quantified. That is, a clause has no free variables. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. For brevity, we often refer to a formula in CNF simply as a CNF. A CNF is often regarded as a set of clauses, thus a formula in CNF is sometimes called a theory. A formula is in *negation normal form* (NNF) if the negation operator is applied only to atoms and the only logical connectives used are conjunction and disjunction. Similarly, we often refer to a formula in NNF simply as an NNF.

For the signature Σ we define a Σ-*interpretation* (alternatively Σ-*structure*) $\mathcal{I} = \langle \mathcal{D}, \mathcal{A} \rangle$ consisting of:

- a non-empty set $\mathcal{D}$ called the *domain*,
- an *assignment* $\mathcal{A}$ which assigns:
  - to each $P \in \Sigma_p$ a subset $P^{\mathcal{I}} \subseteq \mathcal{D}^{ar(P)}$, where $=^{\mathcal{I}} (d_1, d_2)$ evaluates to *true* iff d_1 is the same domain element as d_2,
  - to each $F \in \Sigma_f$ a function $F^{\mathcal{I}} : \mathcal{D}^{ar(F)} \to \mathcal{D}$.

An interpretation is called *finite* if $\mathcal{D}$ is a finite set. For a clause C, we say $\mathcal{I}$ *satisfies* C iff C evaluates to *true* under standard semantics. We say $\mathcal{I}$ *satisfies* a CNF ψ iff for every clause $C \in \psi$, $\mathcal{I}$ satisfies C. Such $\mathcal{I}$ is called a *model* of ψ, which we denote by

$$\mathcal{I} \models \psi.$$

Given a Σ-interpretation $\mathcal{I}$ with a domain $\mathcal{D}$ and a bijection $f : \mathcal{D} \to \mathcal{D}'$, we construct a Σ-interpretation $\mathcal{I}'$ with a domain $\mathcal{D}'$ isomorphic to $\mathcal{I}$ as follows:

$$P^{\mathcal{I}}(a_1, \dots, a_n) \Leftrightarrow P^{\mathcal{I}'}(f(a_1), \dots, f(a_n))$$

for all n-ary predicate symbols $P \in \Sigma_p$ and $a_1, \dots, a_n \in \mathcal{D}$,

$$F^{\mathcal{I}'}(f(a_1), \dots, f(a_n)) = f(F^{\mathcal{I}}(a_1, \dots, a_n))$$

for all n-ary function symbols $F \in \Sigma_f$ and $a_1, \dots, a_n \in \mathcal{D}$. We denote the fact that $\mathcal{I}$ and $\mathcal{I}'$ are isomorphic by $\mathcal{I} \simeq \mathcal{I}'$. Now, given ψ, we have $\mathcal{I} \models \psi$ iff $\mathcal{I}' \models \psi$. This implies that in finite model finding only the size of the domain matters; we can always construct a model in which the domain elements are renamed according to some bijection between domains. Thus, for the remainder of the thesis we simply choose $D = \{0, 1, \dots, n-1\}$ for all domains $\mathcal{D}$ of size n. We call D a *numerical domain*. An interpretaion with a numerical domain is called a *numerical interpretation*. From the choice of our domain, we extend Σ by a set of new distinct constants $\{\overline{0}, \overline{1}, \dots, \overline{n-1}\}$, each interpreted as itself, that is, $\overline{k}$ is interpreted as k for every $k \in D$. We call these *domain constants*.

In our setting we consider Σ-formulas in CNF where Σ_p contains a single predicate symbol $=$ (such system is sometimes called *equational logic*), Σ_f contains a function symbol $*$ of arity 2 (written in infix notation), a function symbol $'$ of arity 1 (written after the term it applies to) and constants *a, b, c, e*. We note, however, that our initial formulas do not contain domain constants. We use *w, x, y, z* to denote variables. Our formulas do not contain any quantifiers; existential quantifiers are removed via a process called Skolemization and all variables in a clause are implicitly universally quantified, thus universal quantifiers are omitted. We focus on CNF formulas that are the axioms of a particular type of a magma[1], such as the non-abelian group axioms

$$\begin{gathered}(e * x = x) \wedge (x * e = x) \wedge (x * x' = e) \wedge (x' * x = e) \\ \wedge (x * (y * z) = (x * y) * z) \wedge (a * b \neq b * a).\end{gathered}$$

Models to such formulas are clearly the respective algebraic structures, in this case non-abelian groups. For the remainder of the thesis, we consider only finite models.

[1]By a magma we mean an algebraic structure that includes an underlying domain equipped with a single binary operation that is closed under this operation, but it can also include functions of other arities. These other functions, however, only propose additional requirements on the binary function symbol.

	0	1	2
0	$1st$	$2nd$	$5th$
1	$3rd$	$4th$	$6th$
2	$7th$	$8th$	$9th$

Figure 1.1 Concentrically sorted cells with highlighted layers.

1.1 Cayley Tables

Let A be a finite model (alternatively viewed as the respective magma), with some fixed domain size n. A finite model is often represented by the operation tables of its function and predicate symbols. Consequently, our primary focus is on the Cayley tables of the respective magmas.

Consider the Cayley table of A. It can be viewed as an $n \times n$ table representing the function $*^A$ assigned to the binary function symbol $*$ by model A. For $d_1, d_2 \in D$ we denote by $A(d_1, d_2)$ the *cell* of the Cayley table with indices d_1, d_2. We write $A(d_1, d_2) = d$ for $d \in D$ iff $d_1 *^A d_2 = d$. From the cells of the Cayley table of A we can form a sequence $A(x_1, y_1), A(x_2, y_2), \ldots, A(x_{n^2}, y_{n^2})$, where $A(x_i, y_i)$ denotes the i-th cell of the sequence, containing each cell of the Cayley table of A exactly once. Such sequence of cells is referred to as the *ordering* of cells of the Cayley table of A. By the ordering of cells of an $n \times n$ Cayley table, we mean an ordering of cells that applies to all $n \times n$ Cayley tables.

We say that the cells of the Cayley table of A are ordered *row by row* when the sequence of cells is of the form

$$A(0,0), A(0,1), \ldots, A(0, n-1), A(1,0), \ldots, A(n-1, n-1).$$

We construct *concentric* ordering as follows. The *k-th layer* of the Cayley table consists of those cells whose maximal index is k. We order the cells first by layers in ascending order. Then, within each layer, the cells are ordered row by row. An example of concentrically ordered cells is given in Figure 1.1.

We define a total order on magmas of a given order as follows. Assume a fixed ordering of cells of an $n \times n$ Cayley table (such as row by row, concentric, or any arbitrary ordering of cells). For magmas A, B of some order n. We write

$$A \preceq B$$

for when the sequence of domain elements $x_1 *^A y_1, x_2 *^A y_2, \ldots, x_{n^2} *^A y_{n^2}$ is lexicographically smaller or equal to the sequence $x_1 *^B y_1, x_2 *^B y_2, \ldots, x_{n^2} *^B y_{n^2}$. We write

$$A \prec B$$

for when the sequence $x_1 *^A y_1, x_2 *^A y_2, \ldots, x_{n^2} *^A y_{n^2}$ is lexicographically strictly smaller than the sequence $x_1 *^B y_1, x_2 *^B y_2, \ldots, x_{n^2} *^B y_{n^2}$.

Usually, we do not order arbitrary magmas but rather isomorphic ones. For a magma A and a permutation of numerical domain elements $\pi \in S_D$ we write $\pi(A)$ to denote a magma defined as

$$x *^{\pi(A)} y = \pi(\pi^{-1}(x) *^A \pi^{-1}(y)).$$

$\diamond$	0	1	2	3	4
0	0	1	0	3	0
1	2	2	0	4	1
2	0	0	0	4	0
3	4	3	1	0	4
4	0	3	0	1	0

$\circ$	0	1	2	3	4
0	0	1	0	3	0
1	4	0	3	1	4
2	0	4	0	0	0
3	2	4	0	2	3
4	0	3	0	1	0

Figure 1.2 Application of $\tau = (1\ 3)$ on the Cayley table of $A = \langle D, \diamond \rangle$.

$\diamond$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

$\circ$	0	1	2	3
0	1	0	3	2
1	0	1	2	3
2	3	2	1	0
3	2	3	0	1

Figure 1.3 $A = \langle D, \diamond \rangle$ and $\pi(A) = \langle D, \circ \rangle$ for $\pi = (1\ 2\ 0)$.

That is, a magma isomorphic to A under the permutation π. It is easy to observe how π affects the Cayley table of $\pi(A)$; it is obtained by simultaneously permuting with π the rows and columns of the Cayley table of A and then applying π on the entries of the resulted table. Also observe that since we choose all domains to be numerical, $A \simeq B$ iff $B = \pi(A)$ for some $\pi \in S_D$.

Example 1. Figure 1.2 depicts the effect of a transposition $\tau = (1\ 3)$ on a magma $A = \langle D, \diamond \rangle$ of order 5. Rows 1, 3 and columns 1, 3 of the respective Cayley table are simultaneously swapped. This results in the cells $A(1,1), A(3,3)$ and $A(1,3), A(3,1)$ being swapped crosswise. The non-highlighted cells are not affected by the permutation of rows and columns. Finally, τ is applied on all entries of the Cayley table of A. The result is a magma $\tau(A) = \langle D, \circ \rangle$.

Example 2. Another example of a magma $A = \langle D, \diamond \rangle$ and an isomorphic magma $\pi(A) = \langle D, \circ \rangle$ for $\pi = (1\ 2\ 0)$ is given in Figure 1.3. Here A is the Klein four-group, therefore $\pi(A)$ is also the Klein four-group. Notice that when we consider cells are ordered row by row, we have $A \prec \pi(A)$. In fact, we have $A \preceq \rho(A)$ for all $\rho \in S_D$, that is, A is the least element with respect to $\preceq$ in its isomorphism class. We will further study such models in the following chapters.

2 MACE-style Model Finding

Let us recall that the objective of this thesis is to develop an algorithm that solves the following problem.

Model Finding Problem

Input: FOL axioms of an algebraic structure, domain size n

Output: All non-isomorphic models to the given axioms of order n

Our strategy is as follows. Assume that the input always provides an FOL formula in CNF. We encode the given set of FOL axioms into a propositional clausal formula. Then, a SAT solver is applied to this propositional formula. The encoding ensures that there exists a model for the original FOL axioms if and only if there exists a satisfying assignment for the new propositional formula. If the SAT solver determines that the propositional formula is satisfiable, we can request a satisfying assignment and extract the corresponding model for the input axioms. However, our goal is to find all models up to isomorphism. We will address this problem later in this chapter.

This idea is originally due to McCune [McC94], where he developed a tool called MACE (Models And CounterExamples) that searches for small finite models of FOL theories. At its core is a DPLL based SAT solver. Modifications to this tool were presented by Claessen and Sörensson [CS03]. Using their novel techniques, they developed a tool called Paradox that uses an incremental DPLL SAT solver. We implement some of the techniques used in Paradox. However, at the core of our program run various state-of-the-art CDCL SAT solvers, which provide a more efficient way in deciding propositional problems given in CNF.

We introduce the following propositional variables, which we later use to construct a Σ-interpretation A. For each constant symbol $c \in \Sigma$ and each domain element d, we introduce a propositional variable $[c = d]$ representing the fact that $c^A = d$. For each pair of domain elements (d_1, d_2), we introduce a propositional variable $[d_1' = d_2]$ representing the fact that $d_1'^A = d_2$. For each triple of domain elements (d_1, d_2, d_3), we introduce a propositional variable $[d_1 * d_2 = d_3]$ representing the fact that $d_1 *^A d_2 = d_3$. In other words, these variables represent that the equalities of the respective ground terms are satisfied by A. For example, $[d_1 * d_2 = d_3]$ is *true* means $A \models \overline{d_1} * \overline{d_2} = \overline{d_3}$. Consequently, $[d_1 * d_2 = d_3]$ is *false* means $A \models \overline{d_1} * \overline{d_2} \neq \overline{d_3}$.

Observe that we need to encode with these variables that each function symbol in Σ is, in fact, a total function on D. For example, we want to avoid satisfying assignments such that both $[d_1 * d_2 = d_3]$ and $[d_1 * d_2 = d_4]$ for $d_3 \neq d_4$ are set *true.* On the other hand, at least one of $[d_1 * d_2 = d]$ for $d \in D$ has to be *true* for every pair d_1, d_2. That is, we aim to encode that exactly one of the variables $[d_1 * d_2 = d]$ is set *true.* Such encoding is referred to as *one-hot* in the literature.

To ensure one-hot encoding of a function symbol F with arity n we impose the following constraint

$$\begin{aligned} \textit{one-hot}(F) \equiv \bigwedge_{a_1,\ldots,a_n \,\in\, D} (\bigvee_{d \,\in\, D} & [F(a_1,\ldots,a_n) = d] \\ & \wedge \bigwedge_{\substack{d,\, d' \,\in\, D \\ d \,<\, d'}} \neg[F(a_1,\ldots,a_n) = d] \vee \neg[F(a_1,\ldots,a_n) = d']\,. \end{aligned} \tag{2.1}$$

The constraint $\bigvee_{d \,\in\, D}[F(a_1,\ldots,a_n) = d]$ is the *totality definition.* It represents the fact that a function must return at least one value for each argument n-tuple. The constraint $\bigwedge_{d,\, d' \,\in\, D,\, d \,<\, d'} \neg[F(a_1,\ldots,a_n) = d] \vee \neg[F(a_1,\ldots,a_n) = d']$ is the *functional definition*, representing the fact that a function can not return two different values for the same argument.

We remark that the constraint 2.1 is an encoding of a so called *cardinality constraint.* When we regard a *true* propositional variable as having value 1 and value 0 if it is *false*, we aim to find an assignment such that

$$\bigwedge_{a_1,\ldots,a_n} \sum_{d} [F(a_1,\ldots,a_n) = d] = 1\,.$$

Using these constraints, we can conveniently encode that, for example, the function symbol F is injective

$$\bigwedge_{a_1,\ldots,a_n} \sum_{d} [F(a_1,\ldots,a_n) = d] = 1 \,\wedge\, \bigwedge_{d} \sum_{a_1,\ldots,a_n} [F(a_1,\ldots,a_n) = d] \leq 1\,. \tag{2.2}$$

Observe that since D is finite, the above constraint also encodes that F is surjective. That is, for F with $ar(F) = 1$, the constraint 2.2 encodes that F is a permutation on D. The various methods for encoding cardinality constraints to CNF are presented in [RM21; Sin05].

2.1 Flattening

Our objective is to create the necessary constraints on the above propositional variables. First, we require the FOL clauses contain only literals of certain form.

Example 3. Consider the literal $x * c = c$ and domain $D = \{0, 1\}$. Notice that we cannot directly encode this literal with the above propositional variables, because we do not know the value of c^A. However, we can condition the encoding on c^A

$$\begin{aligned} ([c = 0] \Rightarrow [0 * 0 = 0]) \wedge ([c = 0] \Rightarrow [1 * 0 = 0]), \\ ([c = 1] \Rightarrow [0 * 1 = 1]) \wedge ([c = 1] \Rightarrow [1 * 1 = 1]). \end{aligned}$$

Definition 1 (*Shallow Literal*)**.** *A FOL literal is* shallow *iff it has one of the following forms*

1. *$c = x$ or $c \neq x$, for all constant symbols $c \in \Sigma$,*
2. *$x' = y$ or $x' \neq y$,*
3. *$x * y = z$ or $x * y \neq z$,*
4. *$x = y$.*

The next step in the encoding, after introducing the new propositional variables, is to transform the input FOL clauses into clauses only containing shallow literals. We call this process *flattening*.

There are two cases when a literal is not shallow

1. it contains a subterm t which is not a variable, although it should be,
2. it is of the form $x \neq y$.

In the first case, we can replace the term t in any literal occuring in a clause C with a new variable v not occuring in C. That is, we apply the following rewrite rule

$$C[t] \longrightarrow t \neq v \vee C[v] \qquad (v \text{ not in } C).$$

We note that if t occurs more than once in C, we introduce only one new variable v for t and replace all occurrences of t in C. In the second case, since all variables in a clause are implicitly universally quantified, for all instances of a clause containing the literal $x \neq y$, in which x and y are not the same domain element, the clause is automatically satisfied. We can thus remove the literal $x \neq y$ and replace all occurrences of y in the clause with x. That is, we apply the following rewrite rule

$$C[x, y] \vee x \neq y \longrightarrow C[x, x].$$

We repeatedly apply the above rules until all literal in the formula are shallow.

Example 4. Consider the literal $x * c = c$ from the previous example. After flattening, the clause looks as follows

$$(c \neq v_0) \vee (x * v_0 = v_0) .$$

Example 5. Consider the semigroup axiom $(x * y) * z = x * (y * z)$. After flattening, the clause looks as follows

$$(x * y \neq v_1) \vee (v_1 * z \neq v_0) \vee (y * z \neq v_2) \vee (x * v_2 = v_0) .$$

2.2 Grounding

After all clauses in our FOL formula are flattened, we generate all instances of each clause for the given domain size n, a process called *grounding*.

Example 6. Consider a subset of the quasigroup axioms

$$(x * y = x * z \Rightarrow y = z) .$$

First, we transform this formula to a clause

$$(x * y \neq x * z \vee y = z) .$$

After flattening, we obtain the the following clause with 4 variables

$$(x * z \neq v_0 \vee x * y \neq v_0 \vee y = z) .$$

The grounding for $D = \{0, 1\}$ then generates the following CNF with 16 clauses

$$\begin{aligned}
&(\overline{0} * \overline{0} \neq \overline{0} \vee \overline{0} * \overline{0} \neq \overline{0} \vee \overline{0} = \overline{0}) \wedge (\overline{1} * \overline{0} \neq \overline{0} \vee \overline{1} * \overline{0} \neq \overline{0} \vee \overline{0} = \overline{0}) \\
\wedge\, &(\overline{1} * \overline{0} \neq \overline{0} \vee \overline{1} * \overline{1} \neq \overline{0} \vee \overline{1} = \overline{0}) \wedge (\overline{1} * \overline{1} \neq \overline{0} \vee \overline{1} * \overline{1} \neq \overline{0} \vee \overline{1} = \overline{1}) \\
\wedge\, &(\overline{1} * \overline{1} \neq \overline{1} \vee \overline{1} * \overline{1} \neq \overline{1} \vee \overline{1} = \overline{1}) \wedge (\overline{0} * \overline{1} \neq \overline{1} \vee \overline{0} * \overline{1} \neq \overline{1} \vee \overline{1} = \overline{1}) \\
\wedge\, &(\overline{0} * \overline{1} \neq \overline{1} \vee \overline{0} * \overline{0} \neq \overline{1} \vee \overline{0} = \overline{1}) \wedge (\overline{0} * \overline{0} \neq \overline{1} \vee \overline{0} * \overline{0} \neq \overline{1} \vee \overline{0} = \overline{0}) \\
\wedge\, &(\overline{0} * \overline{1} \neq \overline{0} \vee \overline{0} * \overline{0} \neq \overline{0} \vee \overline{0} = \overline{1}) \wedge (\overline{0} * \overline{0} \neq \overline{0} \vee \overline{0} * \overline{1} \neq \overline{0} \vee \overline{1} = \overline{0}) \\
\wedge\, &(\overline{1} * \overline{0} \neq \overline{1} \vee \overline{1} * \overline{0} \neq \overline{1} \vee \overline{0} = \overline{0}) \wedge (\overline{1} * \overline{1} \neq \overline{0} \vee \overline{1} * \overline{0} \neq \overline{0} \vee \overline{0} = \overline{1}) \\
\wedge\, &(\overline{0} * \overline{0} \neq \overline{1} \vee \overline{0} * \overline{1} \neq \overline{1} \vee \overline{1} = \overline{0}) \wedge (\overline{0} * \overline{1} \neq \overline{0} \vee \overline{0} * \overline{1} \neq \overline{0} \vee \overline{1} = \overline{1}) \\
\wedge\, &(\overline{1} * \overline{1} \neq \overline{1} \vee \overline{1} * \overline{0} \neq \overline{1} \vee \overline{0} = \overline{1}) \wedge (\overline{1} * \overline{0} \neq \overline{1} \vee \overline{1} * \overline{1} \neq \overline{1} \vee \overline{1} = \overline{0})\,.
\end{aligned}$$

Generally, we substitute each variable in a flattened clause C with a domain constant and generate all such substitutions. Let $\{x_1, \ldots, x_n\}$ be the set of all variables in C. We use the following rewrite rule

$$C[x_1, \ldots, x_n] \longrightarrow \bigwedge_{d_1, \ldots, d_n \in D} C[x_1 \leftarrow \overline{d_1}, \ldots, x_n \leftarrow \overline{d_n}]\,.$$

Thus, we obtain a formula containing only ground terms. After grounding, we simplify all clauses containing literals of the form $\overline{d_1} = \overline{d_2}$; we either remove the whole clause if d_1 and d_2 are equal, or simply remove the literal if d_1 and d_2 are not equal.

Recall that ground shallow literals function as the propositional literals we introduced in the beginning of this chapter. From the simplified FOL clauses, we generate a propositional CNF by replacing each ground shallow literal with its corresponding propositional literal.

Example 6 (Continued). Next, we simplify the formula and replace the ground shallow literals with the corresponding propositional variables

$$\begin{aligned}
&(\neg[1 * 0 = 0] \vee \neg[1 * 1 = 0]) \wedge (\neg[0 * 1 = 1] \vee \neg[0 * 0 = 1]) \\
\wedge\, &(\neg[0 * 1 = 0] \vee \neg[0 * 0 = 0]) \wedge (\neg[0 * 0 = 0] \vee \neg[0 * 1 = 0]) \\
\wedge\, &(\neg[1 * 1 = 0] \vee \neg[1 * 0 = 0]) \wedge (\neg[0 * 0 = 1] \vee \neg[0 * 1 = 1]) \\
\wedge\, &(\neg[1 * 1 = 1] \vee \neg[1 * 0 = 1]) \wedge (\neg[1 * 0 = 1] \vee \neg[1 * 1 = 1])\,.
\end{aligned}$$

Altogether, we first flatten the input FOL clauses. Next, we perform grounding on the flattened clauses for the given domain size n. The resulting clauses are then simplified and translated into a propositional CNF. Lastly, we add to this CNF the one-hot constraint 2.1 for each function symbol occurring in the input formula. If we can find a propositional model that satisfies all of these clauses, we can construct a finite model A satisfying the input FOL clauses; for each function symbol F with $ar(F) = n$ and every n-tuple of domain elements $(d_1, \ldots, d_n)$, the value of $F^A(d_1, \ldots, d_n)$ is then the unique $d \in D$, for which $[F(d_1, \ldots, d_n) = d]$ is set *true*.

2.3 Clause Splitting

Observe that, in grounding, the number of all instances of a clause is exponential in the number of variables it contains. For a clause containing k variables, the number of instances needed is n^k. Furthermore, clause flattening introduces new auxiliary variables to the clause. The idea is to replace a clause by several new clauses, each containing fewer variables than the original clause.

Definition 2 (Binary Split)**.** *Assume a clause $(C[\overline{x}] \vee D[\overline{y}])$ where $\overline{x}, \overline{y}$ are the sets of variables occurring in the subclauses C, D respectively. Then C and D constitute a* binary split *of the given clause iff there exists at least one variable $x \in \overline{x}$ such that $x \notin \overline{y}$, and at least one variable $y \in \overline{y}$ such that $y \notin \overline{x}$. For a new predicate symbol S, the resulting two clauses are*

$$(C[\overline{x}] \vee S(\overline{x} \cap \overline{y})) \wedge (\neg S(\overline{x} \cap \overline{y}) \vee D[\overline{y}])\,.$$

We note that we need to introduce new propositional variables for the new predicate symbol S to encode the resulting FOL clauses into SAT. Assume S is of arity m. For every m-tuple $(d_1, \ldots, d_m)$ we introduce the propositional variable $[S(d_1, \ldots, d_m)]$. Observe that each of the resulting two clauses contains fewer variables from the properties of x, y. In addition, the resulting two clauses can possibly be split further. However, there may be various ways how to split a clause, and some of them could destroy the possibilities for further splitting. We now describe the heuristic presented in [CS03] that addresses this issue.

Definition 3. *Given a clause C, we say that two variables in C are* connected *if there is some literal in C in which they both occur.*

This heuristic finds a variable x which is connected to the least number of other variables in C. When such x is found, we take all literals containing x on the left side of the split, and all other literals on the right side.

Lemma 1. *Using the described heuristic, all the variables in the left split are connected to each other.*

Proof. First, observe that the splitting variable x is trivially connected to all the other variables in the split. Now, let us consider a variable v different from x. If it does not occur as an argument in the fresh predicate symbol S, then v is not connected to any variable occurring in the right split. By our choice of x, v must be connected to at least as many variables as x (that is, all the variables in the left split). If it does occur as an argument of S, then it is connected to all the other variables occurring in S, but also to all the variables not occurring in S by previous argument. □

From the above lemma, we know that the left side of the split cannot be split any further. It therefore suffices to focus only on the right side of the split.

Example 7. Consider the semigroup axiom $(x * y) * z = x * (y * z)$. From an earlier example, we know that after flattening, the clause looks as follows,

$$(x * y \neq v_1) \vee (v_1 * z \neq v_0) \vee (y * z \neq v_2) \vee (x * v_2 = v_0)\,.$$

That is, there are n^6 instances of this clause. After splitting, we obtain the following two clauses

$$\begin{gathered}((v_1 * z \neq v_0) \vee (x * v_2 = v_0) \vee S(v_1, v_2, x, z)),\\ (\neg S(v_1, v_2, x, z) \vee (x * y \neq v_1) \vee (y * z \neq v_2)).\end{gathered}$$

Notice that by splitting, we decreased the number of variables from 6 to 5 in each clause. Thus, we reduced the number of instances from n^6 to $2n^5$.

2.4 Symmetry Breaking

For the remainder of the thesis, it is convenient to define the propositional formula generated for an instance (that is, a set of axioms and a natural number n) of the model finding problem.

Definition 4. *We say φ* encodes an instance of the model finding problem *if φ is a propositional CNF formula generated by the above constructions.*

We will denote such formula by φ. Since there is a one-to-one correspondence between an instance of the model finding problem and the propositional formula φ encoding it, it is natural to think of magmas that are models of the input axioms as models of φ.

Definition 5. *For a magma A, model to an instance of the model finding problem encoded by formula φ, we say A is a* model *of φ. We denote this by $A \models \varphi$.*

We remark that in the above definition A is formally a model of a FOL theory, while φ is a propositional formula. However, a magma A can be viewed as an assignment to propositional variables in the sense of the above constructions. We introduced the notation $A \models \varphi$ merely for convenience. We can lift the notion to any formula σ containing the newly introduced propositional variables and write $A \models \sigma$ iff the assignment of propositional variables induced by A satisfies σ.

Our encoding produces a formula φ that defines a set $\mathcal{C}$ of magmas of some given order n, such that $A \models \varphi$ for every magma $A \in \mathcal{C}$. However, the way we have encoded an instance of the model finding problem as a propositional CNF formula makes the SAT problem unnecessarily difficult. For every model $A \in \mathcal{C}$ of φ and every $\pi \in S_D$, the isomorphic model $\pi(A)$ is also in $\mathcal{C}$. Therefore, the original encoding introduces $n!$ symmetries[1] to every model, as well as to non-models. To break these symmetries, we introduce additional constraints to φ.

Definition 6 (*Symmetry Breaking Constraint*)**.** *Let φ be a formula encoding an instance of the model finding problem. We say a propositional CNF σ is a* symmetry breaking constraint *for φ iff*

(1) for every A, such that $A \models \varphi$, there exists $\pi \in S_D$, such that $\pi(A) \models \varphi \wedge \sigma$.

We say σ is complete *if all the models of $\varphi \wedge \sigma$ are pairwise non-isomorphic. Otherwise, σ is* partial*.*

For a set $\mathcal{C}$ of magmas defined by the original encoding φ, the objective is to find σ, such that the formula $\varphi \wedge \sigma$ induces a new set $\mathcal{C}' \subset \mathcal{C}$, such that for every $A \in \mathcal{C}$ there exists an isomorphic model $\pi(A) \in \mathcal{C}'$. That is, σ makes the new set smaller, but also preserves all the isomorphism classes of $\mathcal{C}$. Note, that any isomorphism class on the set of magmas of order n either contains only models of φ, or contains no models of φ. This indicates why a symmetry breaking constraint can be used to reduce search and thus make the SAT problem easier; instead of visiting each truth assignment, we can determine if φ has a model by visiting each isomorphism class.

[1]For a propositional CNF formula T, we say that a permutation of its variables θ is a symmetry, or automorphism, of T iff $\theta(T) = T$ (when regarded as a set of clauses). In our encoding, every $\pi \in S_D$ induces a permutation on the set of ground flattened clauses. To see this, recall that we generate all instances of a clause during grounding. Transforming these clauses to a propositional CNF thus yields the original set of propositional clauses.

2.4.1 The Least Number Heuristic

Originally introduced in [Zha96], the *least number heuristic* (LNH) was designed to address the symmetry problem in finite model finding. In this subsection, we adapt this heuristic to our specific context. The underlying concept is as follows. Assume an ordering of cells of an $n \times n$ Cayley table. Suppose M is a magma of order n. The *maximal designated number* for a cell $M(x_i, y_i)$ is

$$mdn_i := \begin{cases} 1 & \text{if } i = 1, \\ \max(x_1, \dots, x_i, y_1, \dots, y_i, x_1 *^M y_1, \dots, x_{i-1} *^M y_{i-1}) + 1 & \text{if } i > 1. \end{cases}$$

We say that M satisfies the LNH iff for each i, $M(x_i, y_i) = d$, where $d \leq mdn_i$. Observe that once for some k the $mdn_k = n - 1$, the LNH does not impose any constraints on the values of the cells $M(x_{k+1}, y_{k+1}), \dots, M(x_{n^2}, y_{n^2})$. When encoded to CNF, the LNH is a symmetry breaking constraint in the sense of Definition 6.

Lemma 2. *The LNH is a symmetry breaking constraint for any φ encoding an instance of the model finding problem.*

Proof. We prove that the LNH preserves all isomorphism classes of models and thus satisfies (1). Assume $M \models \varphi$. We construct a magma $\pi(M)$, for some $\pi \in S_D$, that satisfies the LNH. We proceed as follows.

Suppose i is the smallest number such that there exists a cell $M(x_i, y_i)$ with $M(x_i, y_i) = d$, where $d > mdn_i$. If no such i exists, then M satisfies the LNH. Otherwise, denote $m = mdn_i$. Let $\tau = (m\ d)$ and denote $M' = \tau(M)$. We have that the sequences $x_1 *^M y_1, \dots, x_{i-1} *^M y_{i-1}$ and $x_1 *^{M'} y_1, \dots, x_{i-1} *^{M'} y_{i-1}$ are identical, since for $k \in \{1, \dots, i-1\}$, all of $x_k, y_k, x_k *^M y_k, x_k *^{M'} y_k$ are smaller than m and thus remain unaffected by τ. What is more, $M'(x_i, y_i) = m$. We repeat this process on M'. $\square$

Notice that the effectiveness of LNH varies with different cells orderings. For example, when rows are ordered row by row, the maximal designated number trivially increases with each cell, and for all $i \geq n - 1$, $mdn_i = n - 1$. Conversely, when cells are ordered concentrically, the LNH becomes a substantially stronger constraint. For example, when we consider a Cayley table of order 9, $\max(x_1, \dots, x_9, y_1, \dots, y_9)$ is 8 when cells are ordered row by row, but only 2 when cells are ordered concentrically. We note that, generally, the LNH is a partial symmetry breaking constraint.

Next, we demonstrate how to encode the LNH as a propositional constraint. We introduce the following notation. Since we now focus on the Cayley tables of models of φ it is intuitive to denote by $A(x_i, y_i) = d$ the propositional variable $[x_i * y_i = d]$ introduced in the beginning of this chapter. Similarly, $A(x_i, y_i) \neq d$ denotes the propositional literal $\neg[x_i * y_i = d]$.

The value of the cell $M(x_i, y_i)$ can, without any condition, be in the range $0, \dots, m_i$, where $m_i := \max\{x_1, \dots, x_i, y_1, \dots, y_i\} + 1$. Note that we do not know the values $x_1 *^M y_1, \dots, x_{i-1} *^M y_{i-1}$ when building the formula, therefore we also do not know the maximal value of the preceding cells. However, we always know that the value can be k, where $k > m_i$, if there exists some $j < i$, such that $x_j *^M y_j = k - 1$.

It is thus straightforward how to encode the LNH to SAT. Suppose, for the cell $M(x_i, y_i)$, the set of all values larger than m_i is D'_i. We generate the following constraints for the cell $M(x_i, y_i)$

$$\bigwedge_{d \in D'_i} A(x_i, y_i) \neq d \vee \bigvee_{j < i} A(x_j, y_j) = d - 1\,.$$

2.5 Finding All Non-isomorphic Models

In this section, we briefly describe the standard method for finding all non-isomorphic finite models for our model finding problem. The LNH, or another partial symmetry breaking constraint, is used to partially eliminate symmetries from the search space. After a model M has been found, a so called *blocking clause* is generated. Originally due to McMillan [McM02], this technique is simplistic but suffices for our purposes. The blocking clause is of the form

$$\bigvee_{d_1, d_2 \in D} \neg[d_1 * d_2 = d_1 *^M d_2]\,. \tag{2.3}$$

In other words, we look at all pairs (d_1, d_2) and the values $d_1 *^M d_2$, and propose a constraint blocking the found model M. Recall that the CDCL-based solvers at the core of our program support incremental SAT solving. Therefore, we can add the blocking clause to the SAT solver without having to rebuild the formula. The SAT solver is then called again, and if it determines the formula is satisfiable, the new satisfying assignment will produce a model different from M. We can repeat this process to generate all the different models. Note that the blocking clause alone does not guarantee that the new model is non-isomorphic to the previously found models.

The standard procedure is as follows. First, block the model M and then construct a special graph G_M that represents M. Next, apply a graph isomorphism tool to determine the canonical form of G_M. Here we highlight the `nauty` system [McK90; MP14], which is considered state-of-the-art for graph isomorphism. This form is canonical in the sense that the two constructed graphs from any two isomorphic magmas will give the same canonical graph. When a new model is found, its canonical graph is computed and compared with previously obtained canonical graphs. If the new graph is distinct from all previous ones, it indicates we have found a new, non-isomorphic model.

Instead, we take a different approach. In the next chapter, we propose a complete symmetry-breaking formula that is satisfied only by a specific form of a model. This constraint is augmented to the encoding of the instance of the model finding problem. After a model is found, we simply add the blocking clause to the SAT solver. This approach eliminates the need to address graph isomorphism after a new model is found, as all solutions found by the SAT solver will be pairwise non-isomorphic. Moreover, the standard method is impractical in our setting. For instance, using our approach, we were able to find 34,810,736 different isomorphism classes of implication zroupoids of order 6. Finding such a large number of non-isomorphic models using the standard method would clearly be inefficient; we would need to store a canonical representative from each isomorphism class and compare every new model with millions of previous models.

3 Lexleader Models

To find all non-isomorphic magmas that satisfy a given set of axioms, we aim to generate a single representative from each isomorphism class. In the previous chapter, we explained that the standard methods for generating all non-isomorphic models are not effective in our context. To address this issue, we propose a complete symmetry breaking constraint that is satisfied only by a specific form of magma.

3.1 Canonical Forms

We previously suggested that there exists a canonical form of a magma, where two magmas are isomorphic if and only if their corresponding canonical forms are identical. Now, we will provide a formal definition of this canonical form.

Definition 7. *Let $\mathcal{C}$ be a set of all magmas that are models to an instance of the model finding problem. A function $CF : \mathcal{C} \to \mathcal{C}$ is a* canonical form *iff*

- *for all $A \in \mathcal{C}$, $A \simeq CF(A)$ and*
- *for all $A, B \in \mathcal{C}$, $A \simeq B$ iff $CF(A) = CF(B)$.*

We base our canonical form of a magma A on its Cayley table. Given a total order on magmas $\preceq$, as the canonical form of A we choose the least magma with respect to $\preceq$ that is isomorphic to A.

Definition 8 (*Lexleader*)**.** *Magma A is the* lexleader *in its isomorphism class iff for all $\pi \in S_D$ we have $A \preceq \pi(A)$. We say magma A is the* lexleader of *magma B iff A is the lexleader and $A \simeq B$.*

Definition 9 (*Canonicity*)**.** *We say a magma A is* canonical *iff A is the lexleader in its isomorphism class.*

We remark that assigning the lexleader to a magma is a canonical form in the sense of Definition 7. Indeed, since we choose all domains to be numerical, all models in $\mathcal{C}$ are numerical. Therefore, for all $A \in \mathcal{C}$, the lexleader of A is also in $\mathcal{C}$ and is of the form $\rho(A)$ for some $\rho \in S_D$. Furthermore, $\preceq$ is total, so in each isomorphism class the lexleader is unique. Thus, to enumerate all non-isomorphic magmas that are models to a given formula, it suffices to generate the lexleaders as the representatives of each isomorphism class.

We note that a similar idea has been used as early as 1955 to enumerate all distinct[1] semigroups of order 4 [For55]. More recently, an approach based on this idea has also been used to find the complete number of distinct semigroups of order 10 [Dis+12].

Assigning a canonical form to a mathematical structure is beneficial for many reasons. Canonical forms provide a simplified version of mathematical structures, making them easier to understand, manipulate, and analyze. They offer a standard way to represent structures, which aids in consistency and clarity.

[1]Two semigroups are here considered distinct if they are neither isomorphic nor anti-isomorphic.

$\diamond$	0	1	2	3	4	5	6
0	3	5	1	4	6	0	2
1	5	2	6	0	3	1	4
2	1	6	4	5	0	2	3
3	4	0	5	6	2	3	1
4	6	3	0	2	1	4	5
5	0	1	2	3	4	5	6
6	2	4	3	1	5	6	0

$\circ$	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Figure 3.1 $A = \langle D, \diamond\rangle$ and its lexleader $\rho(A) = \langle D, \circ\rangle$ for $\rho = (5\ 0\ 6\ 3)$.

Example 8. [Jan+24] A motivating example is shown in Figure 3.1. It is relatively easy to observe that the Cayley table of magma $A = \langle D, \diamond\rangle$ obeys the Latin square property, therefore A is a quasigroup. It is also apparent that that A is a loop with identity element 5. However, the further structure of this magma is not immediately clear. On the other hand, if we take the lexleader of A with respect to row by row ordering, magma $\rho(A) = \langle D, \circ\rangle$ for $\rho = (5\ 0\ 6\ 3)$, we can see that the operation $\circ$ corresponds to addition modulo 7. Therefore, A is, in fact, the group $\mathbb{Z}_7$.

Generally, we associate each object in an isomorphism class with a vector and then order all the vectors lexicographically; the canonical form will be the object with the smallest vector (alternatively, we can choose the largest vector too). The lexleader canonical form has been applied to various mathematical structures. To list a few examples, the lexicographically largest string of the non-diagonal elements of an adjacency matrix of a graph has been used as the canonical form in [Rea78; IC16; Itz23]. In constraint programming, lexleader has been utilized to find the smallest assignment to a matrix model [Wal12; KNW10; NW13].

Our goal is to find a formula that is satisfied by only the lexleader in each isomorphism class. That is, we aim to identify a specific, complete symmetry breaking constraint.

Definition 10 (*Canonizing Symmetry Breaking Constraint*)**.** *Let φ encode an instance of the model finding problem. We say σ, a complete symmetry breaking constraint for φ, is* canonizing *iff all the models of $\varphi \wedge \sigma$ are canonical.*

We note that in the following lemma, we provide a symmetry breaking constraint σ defined in a metalanguage, rather than as a propositional formula as required by the definition. In addition, so far we have defined $\preceq$ only for concrete magmas of order n. Here, however, A is a placeholder for any magma of order n. That is, $A \preceq \pi(A)$ represents a propositional formula such that for every magma M we have $M \models A \preceq \pi(A)$ iff $M \preceq \pi(M)$. The specific form of this formula is described in the next chapter.

Lemma 3. *Define a symmetry breaking constraint σ as*

$$\sigma = \bigwedge_{\pi \in S_D} A \preceq \pi(A).$$

Then for any instance of the model finding problem with domain size n, σ is a canonizing symmetry breaking constraint.

Proof. Let φ encode an instance of the model finding problem with domain size n. To prove that σ is canonizing, we need to prove that any model A of the formula $\varphi \wedge \sigma$ is canonical. This trivially follows from our definitions of the lexleader and canonicity and the fact that any magma isomorphic to A is of the form $\pi(A)$. $\square$

What remains to be shown is the encoding of the constraint

$$A \preceq \pi(A)$$

into a propositional CNF. Once this is done, we can augment the formula φ with the canonizing symmetry breaking constraint $\bigwedge_{\pi \in S_D} A \preceq \pi(A)$ and apply a SAT solver to this new formula to incrementally generate all non-isomorphic models by adding a blocking clause after each SAT call. This then gives an algorithm solving the model finding problem.

Algorithm 1 Model Finding Problem Algorithm - Naive Approach

```
$\varphi \leftarrow$ encoding of the input instance
$\sigma \leftarrow$ encoding of $\bigwedge_{\pi \in S_D} A \preceq \pi(A)$
$BC \leftarrow true$                                        // Empty blocking clause
while SAT($\varphi \wedge \sigma \wedge BC$) do
    find $M$ such that $M \models \varphi \wedge \sigma \wedge BC$
    print model $M$
    $BC \leftarrow BC \wedge \bigvee_{d_1, d_2 \in D} \neg[d_1 * d_2 = d_1 *^M d_2]$      // Block the current model
end while
```

3.2 Encoding Lexleader to SAT

In this chapter, we give the promised encoding of $A \preceq \pi(A)$ to a propositional CNF. We do this in two parts. First, we encode the original definitions of $A \preceq B$ and $A \prec B$, which involves linearizing the Cayley tables and then lexicographically comparing the respective sequences. From this, we then propose the encoding of $A \preceq \pi(A)$ as a symmetry breaking constraint to SAT. We remind the reader that we assume a fixed ordering $M(x_1, y_1), M(x_2, y_2), \ldots, M(x_{n^2}, y_{n^2})$ of cells of an $n \times n$ Cayley table (that is, an ordering that applies to all magmas of order n) and the induced total order $\preceq$ on magmas of order n.

3.2.1 Encoding of the Total Order

In this subsection, we demonstrate how to encode the definition of $A \preceq B$. The following constraints, written in a metalanguage, should help to better understand the encoding of $A \preceq \pi(A)$ described in the next subsection. First, we introduce new variables $r_1, r_2, \ldots, r_{n^2-1}$ and new notation. We write $A(x_i, y_i) < B(x_i, y_i)$ when $x_i *^A y_i < x_i *^B y_i$ and $A(x_i, y_i) = B(x_i, y_i)$ when $x_i *^A y_i = x_i *^B y_i$. Then, we propose the following constraints on the entries of the cells of the Cayley tables of A and B. For the first cell the constraints

$$A(x_1, y_1) < B(x_1, y_1) \vee \left(A(x_1, y_1) = B(x_1, y_1) \wedge r_1\right),$$

for all $i \in \{2, \dots, n^2-1\}$ the constraints

$$r_{i-1} \Leftrightarrow (A(x_i, y_i) < B(x_i, y_i) \vee (A(x_i, y_i) = B(x_i, y_i) \wedge r_i)),$$

and for the last cell the constraints

$$r_{n^2-1} \Leftrightarrow (A(x_{n^2}, y_{n^2}) < B(x_{n^2}, y_{n^2}) \vee A(x_{n^2}, y_{n^2}) = B(x_{n^2}, y_{n^2})).$$

The variable r_i checks if the last n^2-i cells of A are lexicographically less than or equal to the last n^2-i cells of B. Indeed, if r_i is set *true*, for these constraints to be satisfied either $A(x_{i+1}, y_{i+1}) < B(x_{i+1}, y_{i+1})$ holds or $A(x_{i+1}, y_{i+1}) = B(x_{i+1}, y_{i+1})$ holds and r_{i+1} is set *true* as well.

We now introduce two new variables. First, we need to define the inequality $A(x_i, y_i) < B(x_i, y_i)$ by variable *less*,

$$less_i \equiv \bigwedge_{d \in D} (A(x_i, y_i) = d \Rightarrow \bigvee_{d < d'} B(x_i, y_i) = d'),$$

then, we define the equality $A(x_i, y_i) = B(x_i, y_i)$ by variable *equal*,

$$equal_i \equiv \bigwedge_{d \in D} (A(x_i, y_i) = d \Leftrightarrow B(x_i, y_i)) = d).$$

Altogether, we get an equivalent definition of $\preceq$

$$\begin{aligned} A \preceq B \equiv (less_1 \vee (equal_1 \wedge r_1)) \wedge \bigwedge_{i \in \{2, \dots, n^2-1\}} r_{i-1} \Rightarrow (less_i \vee (equal_i \wedge r_i)) \\ \wedge (r_{n^2-1} \Rightarrow (less_{n^2} \vee equal_{n^2})). \end{aligned} \tag{3.1}$$

The translation of $A \prec B$ is constructed from $A \preceq B$ with the addition of the constraint $\bigvee_{1 \le i \le n^2} \neg equal_i$, ensuring that not all cells of A and B are equal

$$A \prec B \equiv A \preceq B \wedge \bigvee_{1 \le i \le n^2} \neg equal_i. \tag{3.2}$$

3.2.2 Encoding of the Symmetry Breaking Constraint

We base the encoding of $A \preceq \pi(A)$ as a symmetry breaking constraint on the encoding of $A \preceq B$. For the remainder of this subchapter, assume a fixed $\pi \in S_D$. Assume M is a magma of order n. Recall that $A \preceq \pi(A)$ represents a propositional formula such $M \models A \preceq \pi(A)$ iff $M \preceq \pi(M)$. That is, A represents a placeholder instead of a concrete magma of order n. We introduce new variables $r_{\pi,1}, r_{\pi,2}, \dots, r_{\pi,n^2-1}$ along with new notation, similar to the notation used in Subsection 2.4.1. We denote by $A(x_i, y_i) = d$ the propositional variable $[x_i * y_i = d]$, similarly for whenever any of x_i, y_i, d is mapped under some $\pi \in S_D$. By $A(x_i, y_i) \neq d$, we denote the propositional literal $\neg[x_i * y_i = d]$. We denote by $A(x_i, y_i) < \pi(A(\pi^{-1}(x_i), \pi^{-1}(y_i)))$ a propositional formula such that $M \models A(x_i, y_i) < \pi(A(\pi^{-1}(x_i), \pi^{-1}(y_i)))$ iff $x_i *^M y_i < x_i *^{\pi(M)} y_i$. Lastly, $A(x_1, y_1) = \pi(A(\pi^{-1}(x_1), \pi^{-1}(y_1)))$ denotes a propositional formula such that $M \models A(x_i, y_i) = \pi(A(\pi^{-1}(x_i), \pi^{-1}(y_i)))$ iff $x_i *^M y_i = x_i *^{\pi(M)} y_i$. The specific formulas are described below.

We impose the following propositional constraints on the cells of an $n \times n$ Cayley table. For the first cell the constraints

$$\begin{aligned}&A(x_1, y_1) < \pi(A(\pi^{-1}(x_1), \pi^{-1}(y_1)))\\ &\vee\ (A(x_1, y_1) = \pi(A(\pi^{-1}(x_1), \pi^{-1}(y_1)))\ \wedge\ r_{\pi,1}),\end{aligned}$$

for all $i \in \{2, \dots, n^2 - 1\}$ the constraints

$$\begin{aligned}&r_{\pi,i-1} \Leftrightarrow (A(x_i, y_i) < \pi(A(\pi^{-1}(x_i), \pi^{-1}(y_i)))\\ &\vee\ (A(x_i, y_i) = \pi(A(\pi^{-1}(x_i), \pi^{-1}(y_i)))\ \wedge\ r_{\pi,i})),\end{aligned}$$

and for the last cell the constraints

$$\begin{aligned}&r_{\pi,n^2-1} \Leftrightarrow (A(x_{n^2}, y_{n^2}) < \pi(A(\pi^{-1}(x_{n^2}), \pi^{-1}(y_{n^2})))\\ &\vee\ A(x_{n^2}, y_{n^2}) = \pi(A(\pi^{-1}(x_{n^2}), \pi^{-1}(y_{n^2})))).\end{aligned}$$

Just like in Constraint 3.1, the variable $r_{\pi,i}$ checks if the last $n^2 - i$ cells of A are lexicographically less than or equal to the last $n^2 - i$ cells of $\pi(A)$. Again, if $r_{\pi,i}$ is set *true*, for these constraints to be satisfied either $A(x_{i+1}, y_{i+1}) < \pi(A(\pi^{-1}(x_{i+1}), \pi^{-1}(y_{i+1})))$ holds or $A(x_{i+1}, y_{i+1}) = \pi(A(\pi^{-1}(x_{i+1}), \pi^{-1}(y_{i+1})))$ holds and $r_{\pi,i+1}$ is set *true* as well. In this way, the variables $r_{\pi,i}$ chain together the cells of A.

We represent $A(x_i, y_i) < \pi(A(\pi^{-1}(x_i), \pi^{-1}(y_i)))$ by variable $less_\pi$,

$$less_{\pi,i} \equiv \bigwedge_{d \in D} (A(x_i, y_i) = d\ \Rightarrow \bigvee_{\substack{d' \in D\\ d < \pi(d')}} A(\pi^{-1}(x_i), \pi^{-1}(y_i)) = d'),$$

then, we represent $A(x_i, y_i) = \pi(A(\pi^{-1}(x_i), \pi^{-1}(y_i)))$ by variable $equal_\pi$,

$$equal_{\pi,i} \equiv \bigwedge_{d \in D} (A(x_i, y_i) = d\ \Leftrightarrow A(\pi^{-1}(x_i), \pi^{-1}(y_i)) = \pi^{-1}(d)).$$

Finally, the symmetry breaking constraint $A \preceq \pi(A)$ becomes

$$\begin{aligned}&(less_{\pi,1} \vee (equal_{\pi,1}\ \wedge\ r_{\pi,1}))\ \wedge \bigwedge_{i \in \{2,\dots,n^2-1\}} r_{\pi,i-1} \Rightarrow (less_{\pi,i} \vee\ (equal_{\pi,i}\ \wedge\ r_{\pi,i}))\\ &\wedge\ (r_{\pi,n^2-1} \Rightarrow (less_{\pi,n^2} \vee\ equal_{\pi,n^2})).\end{aligned} \tag{3.3}$$

In order to apply a SAT solver to this formula, we need to translate it into CNF. First, the definitions of variables $less_\pi$ and $equal_\pi$ are easily translated into NNF by the semantics of $\Rightarrow$ and $\Leftrightarrow$:

$$less_{\pi,i} \equiv \bigwedge_{d \in D} (A(x_i, y_i) \neq d\ \vee \bigvee_{\substack{d' \in D\\ d < \pi(d')}} A(\pi^{-1}(x_i), \pi^{-1}(y_i)) = d'),$$

$$\begin{aligned}equal_{\pi,i} \equiv \bigwedge_{d \in D} &(A(x_i, y_i) \neq d \vee A(\pi^{-1}(x_i), \pi^{-1}(y_i)) = \pi^{-1}(d)))\\ &\wedge (A(x_i, y_i) = d \vee A(\pi^{-1}(x_i), \pi^{-1}(y_i)) \neq \pi^{-1}(d)))\,.\end{aligned}$$

Then, we add clauses that substitute the variables $less_\pi$ and $equal_\pi$ with their definitions using the *Tseytin transformation*[2]. For variable $less_\pi$, the clauses

$$\bigwedge_{d\in D} \neg less_{\pi,i} \vee A(x_i,y_i) \neq d \vee \bigvee_{\substack{d'\in D\\ d<\pi(d')}} A(\pi^{-1}(x_i),\pi^{-1}(y_i)) = d')$$

for the direction $\Rightarrow$ and the clause

$$less_{\pi,i} \vee \bigvee_{d\in D} l_{\pi,i,d}$$

for the direction $\Leftarrow$. Here we have used new variables $l_{\pi,i,d}$ which are defined as

$$l_{\pi,i,d} \equiv A(x_i,y_i) = d \wedge \bigwedge_{\substack{d'\in D\\ d<\pi(d')}} A(\pi^{-1}(x_i),\pi^{-1}(y_i)) \neq d'$$

for all $d \in D$. We then again apply the Tseytin transformation similarly. For the variable $equal_\pi$, we add the clauses

$$\begin{aligned}\bigwedge_{d\in D} &((\neg equal_{\pi,i} \vee A(x_i,y_i) \neq d \vee A(\pi^{-1}(x_i),\pi^{-1}(y_i)) = \pi^{-1}(d))\\ &\wedge (\neg equal_{\pi,i} \vee A(x_i,y_i) = d \vee A(\pi^{-1}(x_i),\pi^{-1}(y_i)) \neq \pi^{-1}(d)))\end{aligned}$$

for the direction $\Rightarrow$ and the clause

$$equal_{\pi,i} \vee \bigvee_{d\in D} e_{\pi,i,d} \vee \overline{e_{\pi,i,d}}$$

for the direction $\Leftarrow$. Here we have used new variables $e_{\pi,i,d}, \overline{e_{\pi,i,d}}$ which are defined as

$$\begin{aligned} e_{\pi,i,d} &\equiv A(x_i,y_i) = d \wedge A(\pi^{-1}(x_i),\pi^{-1}(y_i)) \neq \pi^{-1}(d)),\\ \overline{e_{\pi,i,d}} &\equiv A(x_i,y_i) \neq d \wedge A(\pi^{-1}(x_i),\pi^{-1}(y_i)) = \pi^{-1}(d))\end{aligned}$$

for all $d \in D$. These definitions are easily translated into CNF using the semantics of and $\Leftrightarrow$ and then distribution. Lastly, we distribute the conjuncts in Constraint 3.3:

$$\begin{aligned}&A \preceq \pi(A) \equiv \qquad (3.4)\\ &((less_{\pi,1} \vee equal_{\pi,1}) \wedge (less_{\pi,1} \vee r_{\pi,1})) \wedge (\neg r_{\pi,n^2-1} \vee less_{\pi,n^2} \vee equal_{\pi,n^2})\\ &\wedge \bigwedge_{i\in\{2,\dots,n^2-1\}} ((\neg r_{\pi,i-1} \vee less_{\pi,i} \vee equal_{\pi,i}) \wedge (\neg r_{\pi,i-1} \vee less_{\pi,i} \vee r_{\pi,i}))\,.\end{aligned}$$

Observe that for $\pi \in S_D$ and $k,l \in D$ such that $\pi(k) = k$ and $\pi(l) = l$ these constraints add a redundant overhead; $M(k,l)$ is the same cell as $M(\pi(k),\pi(l))$. In these cells, it is impractical to consider the whole domain as a candidate for d in variables $less_\pi$ and $equal_\pi$. Let us denote by $D_\pi = \{d \in D : \pi(d) = d\}$ the set of all *fixed points* of π. For such fixed points, we instead propose constraints

[2]Due to Tseytin [Tse68], the Tseytin transformation allows for an easier translation into CNF. It works by substituting a sub-formula φ of a formula ψ with a new variable v. To substitute, generate clauses in both directions of $\Leftrightarrow$ that make v equivalent to φ. Then replace every occurrence of φ in ψ with v and add the generated clauses to ψ.

restricting the possible candidates for d in the respective cells. In particular, for each i such that $x_i, y_i \in D_\pi$, the variable $less_\pi$ becomes

$$less_{\pi,i} \equiv \bigvee_{\substack{d \in D \\ d < \pi(d)}} A(x_i, y_i) = d\,,$$

and the variable $equal_\pi$ becomes

$$equal_{\pi,i} \equiv \bigvee_{d \in D_\pi} A(x_i, y_i) = d\,.$$

These definitions are once again translated into CNF using the Tseytin transformation, which we omit here for brevity.

3.3 Insight into Complexity

Notice, however, the excessive size of this symmetry breaking formula. If we wanted to break all symmetries to find exactly the lexleaders in each isomorphism class, using this naive approach, we would need to augment the original formula with the canonizing symmetry breaking constraint

$$\bigwedge_{\pi \in S_D} A \preceq \pi(A)\,.$$

That is, we would require a symmetry breaking formula of size $\Omega(n!)$, which even for not too large n becomes intractable for most commercial computers.

The set of all symmetries of a propositional theory T over a set of variables L is a subgroup of S_L and is denoted by $\mathrm{Aut}(T)$. It has been proven that the problem of computing, for any T, a predicate true of only the lexicographic leader in each equivalence[3] class of models is NP-hard [Cra+96]. This result is particularly interesting because it is closely related to our problem. In our model finding problem, symmetries of the input axioms are permutations of the domain constants. Each such $\pi \in S_D$ when applied to a ground clause, produces the same clause but with different grounding. That is, each such π corresponds to some $\theta \in \mathrm{Aut}(\varphi)$ where φ encodes an instance of the model finding problem. Thus, it is reasonable to assume that the lexleader verification in our setting is also NP-hard.

Babai and Luks [BL83] have observed that the general problem of deciding whether the adjacency matrix of a graph is the lexleader is NP-hard. In particular, from the knowledge of this matrix, we can deduce the size of the largest independent set of the respective graph. To see this, assume the cells of the adjacency matrix are ordered in a way that each upper left square always precedes its complement (e.g. concentric ordering). Then the upper left square of the lexleader consists of a block of zeros, which represents an independent set with the largest size.

The isomorphism problem for magmas can be reduced to graph isomorphism [Boo78]. Recall that two magmas are isomorphic iff their lexleaders are identical. Thus, the problem of assigning the lexleader to a magma is GI-hard.

[3]A is equivalent to B iff $B = \theta(A)$ for some $\theta \in \mathrm{Aut}(T)$.

$\diamond$	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	2

$\circ$	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	1

Figure 3.2 A that satisfies the LNH with $\tau(A) \prec A$.

3.4 Transpositions

The complexity of complete symmetry breaking formulas should not discourage us from studying the lexleader for breaking symmetries in finite model finding. In the next chapter, we propose a surprising result; for a complete symmetry breaking constraint, in many cases it suffices to use only a small subset of S_D. For the remainder of this chapter, we focus on a subset of S_D for partial symmetry breaking. We show this approach can be thought of as an alternative to the LNH. We get positive results using just $\binom{n}{2}$ elements of S_D, specifically, the set of all *transpositions*.

Definition 11. *We say a magma A is* minimal with respect to the set of all transpositions *iff for all transpositions $\tau \in S_D$ we have $A \preceq \tau(A)$.*

Lemma 4. *Any model minimal with respect to the set of all transpositions also satisfies the LNH.*

Proof. Suppose we have a model A and the smallest i such that there exists a cell $A(x_i, y_i)$ with $A(x_i, y_i) = d$ where d is larger than the maximal designated number m. Then this model does not satisfy the LNH. Moreover, for this model also holds $\tau(A) \prec A$ for $\tau = (m\ d)$, therefore A also is not minimal with respect to the set of all transpositions. $\square$

Example 9. Suppose $A = \langle D, \diamond\rangle$ is a magma of order 4 with $3 *^A 3 = 2$ and $x *^A y = 0$ for all other $x, y \in D$. Then A is permitted by the LNH. However, A is not minimal with respect to the set of all transpositions since for $\tau = (1\ 2)$ and $\tau(A) = \langle D, \circ\rangle$ we have $\tau(A) \prec A$. A and $\tau(A)$ are depicted in Figure 3.2.

From Example 9 and Observation 4 we can conclude that requiring minimality under all transpositions is, in fact, a stronger symmetry breaking constraint than the LNH. We note that this holds for any arbitrary ordering of cells, including concentric or other natural orderings for the LNH, where the upper left square of the Cayley table always precedes its complement. We thus obtain a compact partial symmetry breaking constraint of polynomial size.

4 Canonizing Sets

We described a complete symmetry breaking constraint $\bigwedge_{\pi \in S_D} A \preceq \pi(A)$ in Chapter 3. However, this constraint was deemed intractable due to its factorial complexity. Originally due to Itzhakov and Codish [IC16], an algorithm has been developed for graph search problems that computes a compact canonizing symmetry breaking constraint. In this chapter, we propose modifications and expansions of this algorithm to make it applicable to our model finding problem.

Definition 12 (min_Π)**.** *For a set of permutations $\Pi \subseteq S_D$, we define the predicate over magmas with domain D*

$$min_\Pi(A) \;\equiv\; \bigwedge_{\pi \in \Pi} A \preceq \pi(A)\,.$$

Definition 13 (*Canonizing Set*)**.** *Let φ be a formula encoding an instance of the model finding problem and $\Pi \subseteq S_D$. We say Π is* canonizing for φ *iff for every magma A such that $A \models \varphi$ we have*

$$min_\Pi(A) \Leftrightarrow min_{S_D}(A)\,.$$

Observation. Let Π be canonizing for φ. Then $min_\Pi(A)$ is a canonizing symmetry breaking constraint for φ.

For every $\Pi \subseteq S_D$, the predicate $min_\Pi(A)$ is trivially a partial symmetry breaking constraint for any φ. It is also evident that Π is canonizing if we put $\Pi = S_D$. Our goal is to develop an algorithm that computes a small proper subset of $S_D \backslash \{id\}$ that is canonizing for given φ. We remark that a canonizing set is, in a sense, a form of quantifier elimination[1]; we eliminate the universal quantifier in $\forall \pi \in S_D\, A \preceq \pi(A)$ and replace the formula with $\bigwedge_{\pi \in \Pi} A \preceq \pi(A)$ where, desirably, $|\Pi|$ is significantly smaller than $n!$.

4.1 Computing Canonizing Sets

To compute a canonizing set of permutations for an instance of the model finding problem φ, we first start with an empty Π. We then repeatedly search for a magma A and a permutation π that serve as counterexamples, indicating that Π is not yet a canonizing set. The found π is then added to Π. We repeat this as long as such A and π exist.

Lemma 5. *Algorithm 2 always terminates and returns a canonizing set Π for the given instance of the model finding problem φ.*

Proof. First, observe that a permutation ρ cannot be found repeatedly by the algorithm. If it were, we would have for some A that $\rho(A) \prec A$ and simultaneously $A \preceq \rho(A)$, since A satisfies $min_\Pi(A)$ and ρ has already been added to Π in a previous iteration. Similarly, a magma M cannot be found repeatedly by

[1] A quantifier elimination procedure is an algorithm that constructs an equivalent quantifier-free formula from a given formula. It is predominantly used in deciding satisfiability in the theories of linear integer arithmetic [Coo72] and real arithmetic [FR75].

Algorithm 2 Compute a Canonizing Set for φ

```
1: $\Pi = \emptyset$
2: while $\exists A$ such that $A \models \varphi$, $\exists \pi \in S_D$ such that $min_\Pi(A)$ and $\pi(A) \prec A$ do
3:     $\Pi = \Pi \cup \{\pi\}$
4: end while
5: return $\Pi$
```

the algorithm. If it were, we would have for some π that $\pi(M) \prec M$ and simultaneously $M \preceq \pi(M)$, since M satisfies $min_\Pi(M)$ in one of the subsequent iterations. Because the search space is finite, the algorithm always terminates. In the last iteration of the while cycle, there are no counterexamples A, π such that both $min_\Pi(A)$ and $\pi(A) \prec A$ hold. That is, for A such that $min_\Pi(A)$ holds, we also have that $A \preceq \pi(A)$ for any π, therefore, $min_\Pi(A) \Leftrightarrow min_{S_D}(A)$ holds and Π is canonizing for φ. □

Definition 14. *We say that a canonizing set Π for φ is* redundant *if for some $\pi \in \Pi$, $\Pi \backslash \{\pi\}$ is also canonizing for φ. Otherwise, Π is* irredundant*.*

Observation. The identity permutation is not included in any irredundant canonizing set.

We remark that Algorithm 2 does not guarantee that Π is irredundant. For instance, a permutation added during the early iterations of the while loop may become obsolete in view of permutations added later. The respective canonizing symmetry breaking constraint min_Π is then unnecessarily large. We address this in the following algorithm by linearly traversing through Π and testing each permutation to see if it can be removed.

Algorithm 3 Reduce the Canonizing Set Π for φ

```
1: for each $\pi \in \Pi$ do
2:     if $\forall A$ such that $A \models \varphi$ we have $min_{\Pi \setminus \{\pi\}}(A) \Rightarrow A \preceq \pi(A)$ then
3:         $\Pi = \Pi \setminus \{\pi\}$
4:     end if
5: end for
6: return $\Pi$
```

Lemma 6. *Algorithm 3 always returns an irredundant subset of the given canonizing set Π.*

Proof. Suppose $|\Pi| = m$. Let $\Pi_0 = \Pi$ and denote by Π_i the set obtained after the i-th iteration of the for loop. Assume Π_i is canonizing. After the next iteration, either $\Pi_{i+1} = \Pi_i$ or for some $\pi \in \Pi_i$ we have $\Pi_{i+1} = \Pi_i \backslash \{\pi\}$. In the first case, Π_{i+1} is trivially canonizing. In the second case, the removed π satisfies for every A, such that $A \models \varphi$, that $min_{\Pi_i \setminus \{\pi\}}(A) \Rightarrow A \preceq \pi(A)$, that is $min_{\Pi_i}(A) \Leftrightarrow min_{\Pi_{i+1}}(A)$. Since $min_{\Pi_i}(A) \Leftrightarrow min_{S_D}(A)$, we have that Π_{i+1} is canonizing. At the end of the algorithm, Π_m is irredundant because for any $\pi \in \Pi_m$ there exists A such that $A \models \varphi$ for which we also have $min_{\Pi_m \setminus \{\pi\}}(A) \wedge \pi(A) \prec A$. Therefore, $\Pi_m \backslash \{\pi\}$ is not canonizing. □

Definition 15. *Canonizing set Π for φ is* minimal *if for any other canonizing set Δ for φ we have $|\Pi| \leq |\Delta|$.*

It is important to note that the canonizing set Π returned by Algorithm 3 is not necessarily minimal, although it is always irredundant. We will further discuss the size of these sets in Section 4.3.

Based on the discussion in Section 3.3 we do not expect to find a polynomial algorithm to compute a canonizing symmetry breaking constraint. Therefore, efficient implementations of Algorithm 2 are unlikely. As in [IC16], we base our implementation of Algorithms 2 and 3 on SAT solving. We describe the implementation in detail in the next section, including the encoding of all essential predicates to SAT and the specific techniques used to optimize the performance of our algorithms. Below, we propose Algorithm 4, a modification of Algorithm 1, that solves the model finding problem by utilizing canonizing sets.

Algorithm 4 Model Finding Problem Algorithm - Canonizing Set

```
$\varphi \leftarrow$ encoding of the input instance
$\Pi' \leftarrow$ Algorithm 2 ($\varphi$)
$\Pi \leftarrow$ Algorithm 3($\varphi, \Pi'$)
$\sigma \leftarrow$ encoding of $\bigwedge_{\pi \in \Pi} A \preceq \pi(A)$
$BC \leftarrow true$                                   // Empty blocking clause
while SAT($\varphi \wedge \sigma \wedge BC$) do
    find $M$ such that $M \models \varphi \wedge \sigma \wedge BC$
    print model $M$
    $BC \leftarrow BC \wedge \bigvee_{d_1,d_2 \in D} \neg[d_1 * d_2 = d_1 *^M d_2]$        // Block the current model
end while
```

4.2 SAT Based Implementation

In the previous section, we introduced Algorithm 2, which computes a canonizing set Π, and Algorithm 3, which computes an irredundant subset of Π. We implement these algorithms using SAT solving. To do so, we need to encode the loop conditions of the respective algorithms into propositional CNF. Recall that we already know how to encode $min_{\Pi}(A)$ from Chapter 3 by encoding $A \preceq \pi(A)$ for every $\pi \in \Pi$. Once we have a CNF formula encoding the loop condition, we can repeatedly apply a SAT solver to this formula to simulate the loop. However, here we present propositional formulas that are not in CNF, but rather in an easily understandable form. It is then straightforward to transform these formulas into CNF using the Tseytin transformation, as described in Subsection 3.2.2.

We remind the reader that we assume a fixed ordering of cells of an $n \times n$ Cayley table and the induced total order $\preceq$ on magmas of order n. We adapt the notation from Subsections 2.4.1 and 3.2.2, in particular, $A(x_i, y_i) = d$ denotes the propositional variable $[x_i * y_i = d]$.

4.2.1 Computation Algorithm

We begin by describing the implementation of Algorithm 2. First, we present the encoding of the loop condition

$$\exists A \text{ such that } A \models \varphi, \exists \pi \in S_D \text{ such that } \mathit{min}_{\Pi}(A) \text{ and } \pi(A) \prec A$$

based on the original idea proposed in [IC16]

$$\begin{aligned} \mathit{alg}_1(\Pi, \varphi) \equiv \varphi \;\wedge\; \mathit{one\text{-}hot}(B) \;\wedge\; \mathit{perm}(\pi) \;\wedge\; \mathit{iso}(A, B, \pi) \\ \wedge\; \mathit{min}_{\Pi}(A) \;\wedge\; B \prec A\,. \end{aligned} \tag{4.1}$$

Let us introduce the new notation. Recall that φ encodes an instance of the model finding problem. Specifically, it defines constraints for some magma A such that $A \models \varphi$ iff A satisfies the axioms and is of the order given on the input. The formula φ already defines constraints on the binary function of A, ensuring that it is a total function on D. Here, we are looking for two magmas, A and B. Therefore, it is also necessary to encode the totality of the binary function of B. We do that by adding the constraint *one-hot*(B) (constraint 2.1 applied to the binary operator of B). Again, think of the symbols A, B as placeholders for concrete magmas.

Note that in the loop condition of Algorithm 2, both A and π are variables. It is therefore necessary to encode the new function symbol π as a bijective function on D. We denote by $\pi(d_1) = d_2$, for $d_1, d_2 \in D$, a propositional variable which, based on its truth value, is later used to construct π. This is the same concept as in the beginning of Chapter 2 (for improved readability, square brackets have been omitted from the notation). Let us define a new propositional formula

$$\mathit{injective}(\pi) \equiv \bigwedge_{\substack{i,j,d \in D \\ i<j}} \pi(i) \neq d \vee \pi(j) \neq d\,.$$

We then use the *one-hot* constraint in conjunction with *injective* to produce the constraint

$$\mathit{perm}(\pi) \equiv \mathit{one\text{-}hot}(\pi) \;\wedge\; \mathit{injective}(\pi)\,,$$

which ensures[2] that π is in fact a permutation of the set of domain elements D.

To ensure B is an isomorphic to A under π generate the following constraint

$$\begin{aligned} \mathit{iso}(A, B, \pi) \equiv \bigwedge_{i,j,x \in D} (B(i,j) = x \;\Leftrightarrow \\ \bigvee_{i',j',y \in D} (\pi(i') = i \;\wedge\; \pi(j') = j \;\wedge\; \pi(y) = x \\ \wedge\; A(i',j') = y))\,. \end{aligned}$$

Observe that for each $i, j \in D$ we require that the value of the cell $B(i,j)$ is the same as the value of the cell $A(\pi^{-1}(i), \pi^{-1}(j))$ when mapped under some π. Thus, A and B are indeed isomorphic.

[2] We note that this can alternatively be done using cardinality constraints as in 2.2.

Combining these constraints with $min_{\Pi}(A)$ and $B \prec A$ (analogue of the constraint 3.2 where A, B are not conrete magmas) yields the proposed SAT encoding of the loop condition of Algorithm 2.

To simulate the algorithm, we incrementally apply a single invocation of a SAT solver to find a satisfying assignment to 4.1. From this assignment, we construct the permutation π using the techniques from Chapter 2, add it to Π, and update the formula $min_{\Pi}(A)$ (we add the clauses encoding $A \preceq \pi(A)$). We repeat this process until we obtain *unsatisfiable*, at which point Π consists of permutations comprising a canonizing set for φ.

Notice how cumbersome it is to work with the new symbol B; we need to first introduce new variables and then add the constraints *one-hot*(B) and *iso*(A, B, π). For this reason, we propose our modification to the Constraint 4.1 that avoids the use of B

$$alg_1^b(\Pi, \varphi) \equiv \varphi \ \land \ perm(\pi) \ \land \ min_{\Pi}(A) \ \land \ \pi(A) \prec A\,. \tag{4.2}$$

Specifically, we replace B with $\pi(A)$ and encode the corresponding properties of B within the constraint $\pi(A) \prec A$.

The encoding of $\pi(A) \prec A$ in 4.2, where both π and A are not concrete, is a bit more involved. First, let us define variables $greater^b$ and $equal^b$

$$\begin{aligned} greater_i^b \equiv \bigwedge_{d \in D} & (A(x_i, y_i) = d \Leftrightarrow \\ & \bigvee_{\substack{k,l,m,d' \in D \\ d' < d}} (\pi(k) = x_i \ \land \ \pi(l) = y_i \ \land \ \pi(m) = d' \\ & \land \ A(k,l) = m)), \end{aligned}$$

$$\begin{aligned} equal_i^b \equiv \bigwedge_{d \in D} & (A(x_i, y_i) = d \Leftrightarrow \\ & \bigvee_{k,l,m \in D} (\pi(k) = x_i \ \land \ \pi(l) = y_i \ \land \ \pi(m) = d \\ & \land \ A(k,l) = m)). \end{aligned}$$

Again, we need to add a clause that guarantees the sequences of cells of A and $\pi(A)$ are not equal:

$$\bigvee_{1 \le i \le n^2} \neg equal_i^b\,.$$

Let us introduce new variables $r_1, \ldots, r_{n^2-1}$. Finally, the constraint $\pi(A) \prec A$ in 4.2 becomes

$$\begin{aligned} \pi(A) \prec A \ \equiv & (greater_1^b \lor (equal_1^b \ \land \ r_1)) \\ & \land \bigwedge_{2 \le i \le n^2-1} r_{i-1} \Rightarrow (greater_i^b \lor \ (equal_i^b \ \land \ r_i)) \\ & \land \ (r_{n^2-1} \Rightarrow (greater_{n^2}^b \lor \ equal_{n^2}^b)) \\ & \land \bigvee_{1 \le i \le n^2} \neg equal_i^b\,. \end{aligned} \tag{4.3}$$

Unlike in 3.3, here the variable r_i checks if the last $n^2 - i$ cells of A are lexicographically greater than or equal to the last $n^2 - i$ cells of $\pi(A)$. What remains the same is the essence of this variable; it chains together the cells of A.

Another optimization we have developed is as follows. Consider a magma with a constant symbol e (such as group or loop). For each such magma, there exists an isomorphic magma in which e is equal to 0. For such instances of the model finding problem, it is sufficient to look for models where $e = 0$ and permutations π where $\pi(0) = 0$. The encoding of the loop condition is then the following

$$\begin{aligned} alg_1^b(\mathit{\Pi}, \varphi) \equiv \varphi \wedge perm(\pi) \wedge min_{\mathit{\Pi}}(A) \wedge \pi(A) \prec A \\ \wedge\ \pi(0) = 0 \wedge [e = 0]\,. \end{aligned}$$

4.2.2 Reduction Algorithm

We proceed by describing the implementation of Algorithm 3. The objective is to encode the loop condition

$$\forall A \text{ such that } A \models \varphi \text{ we have } min_{\mathit{\Pi}\setminus\{\pi\}}(A) \Rightarrow A \preceq \pi(A).$$

We choose $\pi \in \mathit{\Pi}$ and generate the following constraint

$$alg_2(\mathit{\Pi}, \pi, \varphi) = \varphi \wedge min_{\mathit{\Pi}\setminus\{\pi\}}(A) \wedge \pi(A) \prec A\,. \tag{4.4}$$

The SAT encoding of the loop condition works by negating the implication; if Constraint 4.4 is unsatisfiable then there is no A model of φ for which simultaneously $min_{\mathit{\Pi}\setminus\{\pi\}}(A)$ and $\pi(A) \prec A$ hold. That is, the negation of this conjunction, $\neg min_{\mathit{\Pi}\setminus\{\pi\}}(A) \vee A \preceq \pi(A)$ ($\Leftrightarrow min_{\mathit{\Pi}\setminus\{\pi\}}(A) \Rightarrow A \preceq \pi(A)$), holds for every A model of φ. Therefore, we check for unsatisfiability of 4.4, in which case the counterexample permutation π is redundant and removed from $\mathit{\Pi}$. The process is then repeated for the next permutation in $\mathit{\Pi}$. Once all permutations in $\mathit{\Pi}$ have been tested, we end up with an irredundant canonizing set for φ.

The encoding of $\pi(A) \prec A$ in 4.4 (note that here π is concrete) is similar to the encoding of $A \preceq \pi(A)$ (Constraint 3.3). We use the same definition of the variable $equal_\pi$, but instead of the variable $less_\pi$ we use the variable $greater_\pi$ defined as follows

$$greater_{\pi,i} \equiv \bigwedge_{d\in D} \Big(A(x_i, y_i) = d \Rightarrow \bigvee_{\substack{d'\in D\\ \pi(d')<d}} A(\pi^{-1}(x_i), \pi^{-1}(y_i)) = d'\Big)\,,$$

for the general case, and for all the fixed point pairs $x_i, y_i \in D_\pi$ we define it as

$$greater_{\pi,i} \equiv \bigvee_{\substack{d\in D\\ \pi(d)<d}} A(x_i, y_i) = d\,.$$

We then add a clause ensuring that the sequences of cells of A and $\pi(A)$ are not equal

$$\bigvee_{1\,\leq\, i\,\leq\, n^2} \neg equal_{\pi,i}\,.$$

Finally, with the use of new variables $s_{\pi,1}, \ldots, s_{\pi,n^2-1}$ we obtain the encoding of the constraint

$$\begin{aligned}\pi(A) \prec A \equiv\ & (greater_{\pi,1} \vee (equal_{\pi,1} \ \wedge \ s_{\pi,1})) \\ & \wedge \bigwedge_{2 \le i \le n^2-1} s_{\pi,i-1} \Rightarrow (greater_{\pi,i} \vee \ (equal_{\pi,i} \ \wedge \ s_{\pi,i})) \\ & \wedge \ (s_{\pi,n^2-1} \Rightarrow (greater_{\pi,n^2} \vee \ equal_{\pi,n^2})) \\ & \wedge \bigvee_{1 \le i \le n^2} \neg equal_{\pi,i} \,. \end{aligned} \tag{4.5}$$

Notice that after each iteration, the constraints $min_{\Pi \setminus \{\pi\}}(A)$ and $\pi(A) \prec A$ in 4.4 are completely altered. Unlike in 4.2, where we could simply add new clauses to the original constraint, here we need to check for unsatisfiability of a different constraint after each SAT call. To avoid creating new invocations of a SAT solver, we modify the original idea from [IC16] and introduce assumptions to our encoding.

Modern CDCL SAT solvers allow calls with a set of assumptions For propositional literals $a_1, \ldots, a_k$, the call SAT($\{a_1, \ldots, a_k\}$) checks the satisfiability of a formula under the assumption that all $a_1, \ldots, a_k$ are satisfied. Assumptions can be used to effectively disable a clause. Instead of a clause C, we add a clause $C \vee v_C$ for a new variable v_C. Then, the call SAT($\{\neg v_C\}$) enables C; the call behaves as if it were using the original formula. However, the call SAT($\{v_C\}$) disables the clause C; by assuming v_C is *true*, the clause $C \vee v_C$ is automatically satisfied.

Denote by ψ the propositional encoding of the constraint $A \preceq \pi(A)$ for $\pi \in \Pi$. Replace each clause $C \in \psi$ with $C \vee a_\pi^\preceq$, where $a_\pi^\preceq$ is a new variable. Similarly, replace each clause C in the propositional encoding of $\pi(A) \prec A$ with $C \vee a_\pi^\prec$ for a new variable $a_\pi^\prec$. Let us denote the new formulas $A \preceq \pi(A) \vee a_\pi^\preceq$ and $\pi(A) \prec A \vee a_\pi^\prec$ respectively. Finally, the new encoding of the loop condition is

$$alg_2^b(\Pi, \varphi) = \varphi \ \wedge \ \bigwedge_{\pi \in \Pi} A \preceq \pi(A) \vee a_\pi^\preceq \ \wedge \ \bigwedge_{\pi \in \Pi} \pi(A) \prec A \vee a_\pi^\prec \,. \tag{4.6}$$

The SAT implementation of the modified Algorithm 3 is then described in the following algorithm.

Algorithm 5 Reduce the Canonizing Set Π for φ - SAT Implementation

```
1: $\Pi' \leftarrow \Pi$
2: SAT $\leftarrow alg_2^b(\Pi, \varphi)$                // Add formula to SAT solver
3: **for** each $\pi \in \Pi$ **do**
4:     $\Pi' \leftarrow \Pi' \setminus \{\pi\}$
5:     r $\leftarrow$ SAT($\{a_\rho^\preceq \mid \rho \in \Pi \setminus \Pi'\} \cup \{\neg a_\rho^\preceq \mid \rho \in \Pi'\} \cup \{a_\rho^\prec \mid \rho \in \Pi \setminus \{\pi\}\} \cup \{\neg a_\pi^\prec\}$)
6:     **if** r = *satisfiable* **then**
7:         $\Pi' = \Pi' \cup \{\pi\}$
8:     **end if**
9: **end for**
10: **return** $\Pi'$
```

4.3 Size of Canonizing Sets

As previously mentioned, Algorithm 3 does not necessarily produce a minimal canonizing set. We remark that the problem of finding a minimal canonizing set is related to the problem of finding an *irredundant equivalent subset* of clauses. Given a set of clauses, we search for a subset that is both *equivalent* (having the same set of models as the original) and *irredundant* (where removing any clause changes the set of models). It has been proven that the problem of deciding the existence of an irredundant equivalent subset of a given size is Σ_2^p-complete [Lib05]. Consequently, the problem of actually finding such a subset is likely more complex, as it involves constructing the solution rather than merely deciding its existence.

Therefore, Algorithm 3 should not be considered an algorithm that finds a minimal canonizing set; rather, it linearly prunes a canonizing set generated by Algorithm 2. This theoretical insight is supported by our experimental results. In the next chapter, we will present a set of permutations produced by the canonizing set algorithms, as illustrated in Example 10, which clearly is not minimal.

Generally, we do not expect canonizing sets to be small. Itzhakov [Itz23] points out that if there exists a canonizing symmetry breaking constraint ψ for graph search problems of polynomial size, then deciding if a given adjacency matrix of a graph is the lexleader in its isomorphism class can be done efficiently; given the adjacency matrix A of a graph, take the formula $\psi(A)$. Then, A is the lexleader iff $\psi(A)$ reduces to *true*. Considering the lexleader problem for graphs is NP-hard [BL83] and unless P=NP, then canonizing sets are likely to be superpolynomial.

5 Experiments

We implemented our program in `Python 3.10.12` using the `PySAT` package [IMM18]. The repository containing our program implementation is included in Attachment A.1. `PySAT` enables fast prototyping with SAT oracles and related technologies and provides a simple API for interacting with several state-of-the-art SAT oracles. All propositional formulas in our program are constructed in `Python` and processed using `PySAT`. To determine satisfiability of propositional formulae and find satisfying assignments, we use the `CaDiCaL 1.9.5` [Bie+20] SAT solver via its API.

In our experiments, we use the axiomatic definition of an algebraic structure and consider small orders from 2 to 10. For each order, we follow these steps: first, we encode the respective instance of the model finding problem into SAT. Then, we compute a canonizing set using our implementation of Algorithm 2, which is subsequently reduced using our implementation of Algorithm 3. Finally, we build a canonizing symmetry breaking constraint that we add to the encoding of the instance. For our experiments, we adopt the natural row-by-row ordering of cells of an $n \times n$ Cayley table and the induced orders on magmas $\preceq$ and $\prec$.

Recall that our program builds a propositional formula, such that all of its possible satisfying assignments correspond to all the non-isomorphic models of the input instance of the model finding problem. Moreover, all these models are canonical in the sense of Definition 9. Our program can generate all the models by augmenting the formula with a blocking clause after each model is found (see Section 2.5). Enumerating models using `Python` is slow due to its interpreted nature and general-purpose design, which are not optimized for intensive computational tasks. For these experiments, we computed the counts of all satisfying assignments[1] of the final formula using the model counter tools `D4` [LM17] and `clingo` [Geb+19].

The experiments were run on a server with four AMD EPYC 7513 32-Core Processor @ 2.6GHz and with 504 GB of memory. For each problem, we set a memory limit of 50 gigabytes and a timeout of 24 hours.

5.1 Used Algebraic Structures

We tested our program on various algebraic structures, detailed in Table 5.1. To find the canonical forms of these structures, it is sufficient to search for the lexleader Cayley table. This approach is straightforward for structures involving only the function symbol $*$ or well-known magmas, like groups. We now provide explanations for the less intuitive cases: *inverse property loop* and *implication zroupoid.*

Inverse property loop: Suppose we have a Cayley table of a model of this theory. We show this table cannot correspond to two different models. Fix $x, y \in D$. Now, if there existed two different assignments to $'$, denoted $'^1$, $'^2$, such that $x'^1 \neq x'^2$ and the two corresponding models were non-isomorphic, then the Cayley table would need to have two occurrences of y in column $(x * y)$ and two occurrences of

[1]We note that it suffices to count different assignments only for the variables $[A(d_1, d_2) = d_3]$ corresponding to the distinct entries of the Cayley table.

Structure	**Definition in FOL**
AG-groupoid	$(x * y) * z = (z * y) * x$
Commutative quasigroup	$(x * y = x * z \Rightarrow y = z) \wedge (y * x = z * x \Rightarrow y = z)$ $\wedge (x * y = y * x)$
Group	$(x * (y * z) = (x * y) * z) \wedge (x * e = x) \wedge (e * x = x)$ $\wedge (x * x' = e) \wedge (x' * x = e)$
Implication zroupoid	$((x * y) * z = (((z * e) * x) * ((y * z) * e)) * e)$ $\wedge ((e * e) * e = e)$
Inverse semigroup	$(x * (y * z) = (x * y) * z)$ $\wedge (x = x * (x' * x)) \wedge (x' = x' * (x * x'))$ $\wedge ((x * x = x \wedge y * y = y) \Rightarrow x * y = y * x)$
Inverse property loop	$(x * y = x * z \Rightarrow y = z) \wedge (y * x = z * x \Rightarrow y = z)$ $\wedge (e * x = x) \wedge (x * e = x) \wedge (x' * (x * y) = y)$ $\wedge ((y * x) * x' = y)$
Loop	$(x * y = x * z \Rightarrow y = z) \wedge (y * x = z * x \Rightarrow y = z)$ $\wedge (e * x = x) \wedge (x * e = x)$
Magma	no requirement
Monoid	$(x * (y * z) = (x * y) * z) \wedge (x * e = x) \wedge (e * x = x)$
Quasigroup	$(x * y = x * z \Rightarrow y = z) \wedge (y * x = z * x \Rightarrow y = z)$
Rectangular groupoid	$(w * x = y * z) \Rightarrow (w * x = w * z)$
Right involutory magma	$(x * y) * y = x$
Semigroup	$x * (y * z) = (x * y) * z$

Table 5.1 FOL definitions of the used algebraic structures. For further reference, see: AG-groupoids [PS95], implication zroupoids [CS21], rectangular groupoids [Boy13], and right involutory magmas [CM23].

y in row $(y * x)$. This is in conflict with the latin square property of the Cayley table ensured by the formula

$$((x * y = x * z) \Rightarrow (y = z)) \wedge ((y * x = z * x) \Rightarrow (y = z)).$$

Implication zroupoid: This represents a special case of an algebra with only one constant symbol. For any model of this theory, there is an isomorphic model where e is equal to 0. Thus, without loss of generality, we can simplify our search by assuming $e = 0$ and focusing on the corresponding Cayley table to determine the canonical form. Consequently, we can add the unit clause $[e = 0]$ to the input formula and modify Algorithm 2 to only search for permutations π where $\pi(0) = 0$. This way, for a model A such that $e^A = 0$, we search only for models $\pi(A)$ with $e^{\pi(A)} = 0$. We remark that this modification can be automatically used for all input formulas with only one constant symbol (recall that our input formulas do not contain literals of the form $a \neq \overline{0}$). Note that originally, an implication zroupoid is defined as an algebra $\mathcal{A} = \langle A, \to, 0 \rangle$ where A satisfies the identities $(x \to y) \to z \approx ((z' \to x) \to (y \to z)')'$, where $x' := x \to 0$, and $0'' \approx 0$. Here, we simply translated the definition into our language.

We also note that for the FOL definition of an *inverse semigroup*, we used the following equivalent formulation: An inverse semigroup S is a semigroup in which every element has at least one inverse, and idempotents commute.

5.2 Results

Next, we present the results of our experiments. We start with the example mentioned in Section 4.3, illustrating a reduced canonizing set for a specific magma, computed by the canonizing set algorithms, which is clearly not minimal.

Example 10. Consider the FOL unit clause $x * y = z * w$. For any of its model A we have $A(x, y) = d$ for all $x, y \in D$ for some fixed $d \in D$. For every domain size there exists exactly one isomorphism class and the lexleader Cayley table consists of all zeros (with respect to any ordering of cells). It is easy to check that there exists a minimal canonizing set for a domain of size n containing exactly the permutation $(n-1\ n-2\ \ldots\ 1\ 0)$. However, when our implementations of Algorithm 2 and Algorithm 3 are run on this formula encoded for domain size 8, they produce the following canonizing set of size 4:

$$\{(0\ 7)(1\ 6)(2\ 5)(3\ 4), (0\ 7\ 1\ 6\ 2\ 5\ 3), (0\ 7\ 1\ 6\ 2)(5\ 3), (0\ 7\ 1)(6\ 2)(5\ 3)\}.$$

We have created a table with the resulting data for each of the algebraic structures used. Let us now describe these tables in detail, using Table 5.2 for a magma as an example.

- First column: Contains domain sizes n, ordered in increasing order from top to bottom. This table lists data for domain sizes 2 to 6. Domain size 1 was not tested, as it only yields trivial models. The highest domain size listed is the largest size for which we could compute a canonizing set within the memory limit. None of our experiments ran out of time during the computation of a canonizing set.
- Second column: Lists the values of $n!$.
- Third and fourth columns: Show the sizes of the canonizing sets for the corresponding domain size computed by Algorithm 2, and the time taken for the computation, respectively.
- Fifth and sixth columns: Show the sizes of the reduced canonizing sets for the corresponding domain size computed by Algorithm 3, and the time taken for this computation, respectively. The size of the reduced canonizing set is shown in bold.
- Seventh, eighth, and ninth columns: Present data from the model counting tools `D4` and `clingo`. The seventh column shows the computed counts[2] for the corresponding domain sizes, while the eighth and ninth columns list the computation times for `D4` and `clingo`, respectively. The symbol (–) indicates that the computation either ran out of memory or exceeded the time limit.

[2]The counts obtained using both tools were consistent across all experiments, except in cases where one of the tools ran out of time or exceeded the memory limit, preventing it from computing a solution.

The time in all tables is measured in seconds and rounded to the nearest second. For example, a time of 0 seconds indicates that the computation took less than half a second.

		Algorithm 2		**Algorithm 3**			**D4**	**clingo**
n	$n!$	size	time	size	time	count	time	time
2	2	1	0	**1**	0	10	0	0
3	6	5	0	**5**	0	3,330	0	0
4	24	23	3	**23**	0	178,981,952	146	2,266
5	120	119	61	**119**	2	–	–	–
6	720	719	3,392	**719**	55	–	–	–

Table 5.2 Magma (no restrictions). The counts are consistent with those listed in the OEIS sequence for magmas [Inc24a].

The values of $n!$ are provided for comparison with the sizes of the reduced canonizing sets. Recall that the lexleader constraint can be encoded trivially by encoding $A \preceq \pi(A)$ for all $\pi \in S_D$. In most tables, the differences are substantial. However, in the case of a magma, the size of the reduced canonizing set is only smaller by 1.

Notice that if φ encodes only the one-hot restriction on $*$ (Constraint 2.1) for the given order n, then the solutions to φ are all magmas of order n. Applying Algorithms 2 and 3 to such φ then generates a set that is canonizing for all instances of the model finding problem of the given order. If we could build a compact canonizing symmetry breaking constraint for such φ, we could use this constraint for all instances of the model finding problem with order n. We refer to such a constraint as an *instance-independent* symmetry breaking constraint. With a compact, instance-independent canonizing symmetry breaking constraint, we could eliminate the need to compute a canonizing set in the model finding Algorithm 4. It suffices to precompute the instance independent symmetry breaking constraint for each n and then apply the constraint for order n whenever we are given an instance of that order.

However, our findings, as shown in Table 5.2, indicate that the canonizing symmetry breaking constraint for such φ requires encoding $A \preceq \pi(A)$ for all $\pi \in S_D$ except the identity permutation. This connects to the discussion in Section 4.3, where we noted that canonizing symmetry breaking constraints for graph search problems are expected to be superpolynomial, assuming P$\neq$NP. In contrast, we are addressing an even more challenging problem with magmas. The Cayley table of a magma typically contains a wider range of entries than just 0 and 1, unlike the adjacency matrix of a graph. Based on our findings, we expect that canonizing symmetry breaking constraints for magmas are of exponential size, unconditionally.

We also note that there is no non-trivial lower bound on the size of a minimal canonizing set. For example, in Example 10, we presented an algebra where the canonizing symmetry breaking constraint requires only a single permutation.

Next, we present the results in Table 5.3 for AG-groupoids and Table 5.4 for implication zroupoids. To our knowledge, the number of AG-groupoids of order 7 is unknown. However, we identified 643,460,323,187 distinct satisfying assignments for the propositional formula encoding this instance, which took 72,825 seconds to compute.

For implication zroupoids, the table lists canonizing sets for domain sizes 2 to 8. We were unable to compute the count for domain size 8, and the counts for domain sizes 6 and 7 are also unknown according to the surveyed literature. Using our method, we computed 34,810,736 non-isomorphic models of order 6 in 42 seconds and 600,767,308,670 non-isomorphic models of order 7 in 49,167 seconds.

		Algorithm 2		**Algorithm 3**			**D4**	**clingo**
n	$n!$	size	time	size	time	count	time	time
2	2	1	0	**1**	0	3	0	0
3	6	4	0	**3**	0	20	0	0
4	24	9	0	**6**	0	331	0	0
5	120	28	6	**21**	1	31,913	3	3
6	720	82	56	**62**	6	40,104,513	81	5,645
7	5,040	285	815	**240**	111	643,460,323,187	72,825	–
8	40,320	1,306	22,284	**1,154**	4,027	–	–	–

Table 5.3 AG-groupoid. The counts of AG-groupoids for orders 2 to 6 are consistent with those reported in [DSS11].

		Algorithm 2		**Algorithm 3**			**D4**	**clingo**
n	$n!$	size	time	size	time	count	time	time
2	2	0	0	**0**	0	3	0	0
3	6	1	0	**1**	0	17	0	0
4	24	4	0	**3**	0	249	0	0
5	120	12	1	**11**	0	22,707	2	10
6	720	41	11	**40**	3	34,810,736	42	14,988
7	5,040	188	307	**173**	50	600,767,308,670	49,167	–
8	40,320	1,007	6,730	**958**	1,489	–	–	–

Table 5.4 Implication zroupoid.

Below is the Table 5.5, which presents the results of testing rectangular groupoids. We attempted to count the number of non-isomorphic rectangular groupoids of order 6, a count that, to our knowledge, is not yet known. Unfortunately, both of our model counting tools exceeded the time limit for this computation. An indication of the problem's difficulty is the size of the reduced canonizing set for domain size 6, which is 633. This situation is similar to what we observed in Table 5.2 for a magma, where the size of the reduced canonizing set approaches $n!$.

Next, we present Table 5.6, which lists the results for groups. The findings are particularly notable because, for each order, the sizes of the reduced canonizing sets are the smallest among all the algebraic structures tested. This is intuitive, as the structure of a group imposes the most restrictions on the binary symbol $*$ compared to the other tested magmas. Therefore, for each order, the fewest magmas A satisfy the loop condition in Algorithm 2, which computes a canonizing set. Consequently, the fewest permutations π are found such that $\pi(A) \prec A$.

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	1	0	**1**	0	5	0	0
3	6	5	0	**5**	0	49	0	0
4	24	20	1	**20**	0	1,864	0	0
5	120	112	26	**107**	5	200,704	62	43
6	720	660	1,007	**633**	118	–	–	–

Table 5.5 Rectangular groupoid.

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	0	0	**0**	0	1	0	0
3	6	0	0	**0**	0	1	0	0
4	24	2	0	**1**	0	2	0	0
5	120	3	0	**2**	0	1	0	0
6	720	9	1	**4**	0	2	1	1
7	5,040	13	3	**8**	0	1	2	2
8	40,320	24	12	**10**	1	5	8	5
9	362,880	29	55	**12**	3	2	12	13
10	3,628,800	76	180	**18**	20	2	31	26

Table 5.6 Group.

To compare the sizes of the reduced canonizing sets across different algebraic structures, please refer to the tables presented in the remainder of this chapter. The captions of these tables include references to literature related to the enumeration of all non-isomorphic models for these structures.

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	0	0	**0**	0	2	0	0
3	6	1	0	**1**	0	7	0	0
4	24	4	0	**3**	0	35	0	0
5	120	13	1	**7**	0	228	0	0
6	720	39	11	**24**	2	2,237	5	4
7	5,040	114	99	**81**	17	31,559	76	42
8	40,320	469	1,370	**333**	250	1,668,997	2,398	1,865

Table 5.7 Monoid. The computed counts align with those in [Inc24f].

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	1	0	**1**	0	2	0	0
3	6	3	0	**3**	0	5	0	0
4	24	8	0	**7**	0	16	0	0
5	120	15	2	**12**	0	52	1	1
6	720	36	8	**24**	1	208	6	3
7	5,040	67	58	**51**	10	911	56	18
8	40,320	144	681	**92**	48	4,637	270	74
9	362,880	279	3,646	**164**	220	26,422	1,273	364
10	3,628,800	569	16,652	**317**	1,511	169,163	9,220	–

Table 5.8 Inverse semigroup. The computed counts are consistent with the data in [Mal19].

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	1	0	**1**	0	1	0	0
3	6	2	0	**2**	0	3	0	0
4	24	8	0	**7**	0	7	0	0
5	120	18	1	**10**	0	11	0	0
6	720	113	39	**78**	7	491	17	8
7	5,040	512	639	**379**	139	6,381	770	151

Table 5.9 Commutative quasigroup. The computed counts align with those reported in [Inc24e].

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	0	0	**0**	0	1	0	0
3	6	0	0	**0**	0	1	0	0
4	24	2	0	**1**	0	2	0	0
5	120	3	0	**2**	0	1	0	0
6	720	9	1	**6**	0	2	0	0
7	5,040	10	5	**9**	0	2	1	2
8	40,320	41	24	**17**	3	8	4	5
9	362,880	94	77	**44**	17	7	13	24
10	3,628,800	125	390	**46**	25	47	59	36

Table 5.10 Inverse property loop. The computed counts match those found in [AS08].

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	0	0	**0**	0	1	0	0
3	6	0	0	**0**	0	1	0	0
4	24	2	0	**1**	0	2	0	0
5	120	5	0	**4**	0	6	0	0
6	720	40	2	**29**	1	109	2	2
7	5,040	394	144	**350**	73	23,746	290	101

Table 5.11 Loop. The computed counts are consistent with [Inc24c].

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	1	0	**1**	0	1	0	0
3	6	3	0	**2**	0	5	0	0
4	24	13	0	**9**	0	35	0	0
5	120	83	12	**72**	3	1,411	7	4
6	720	582	440	**545**	91	1,130,531	4,236	1,380

Table 5.12 Quasigroup. The computed counts are consistent with [Inc24d].

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	1	0	**1**	0	3	0	0
3	6	4	0	**4**	0	16	0	0
4	24	23	1	**21**	0	475	0	0
5	120	112	21	**111**	2	100,666	23	16
6	720	702	828	**696**	95	267,954,164	44,517	–

Table 5.13 Right involutory magma. The computed counts match [Inc24g].

		Algorithm 2		Algorithm 3		D4		clingo
n	$n!$	size	time	size	time	count	time	time
2	2	1	0	**1**	0	5	0	0
3	6	4	0	**3**	0	24	0	0
4	24	12	1	**7**	0	188	0	0
5	120	35	8	**24**	1	1,915	2	2
6	720	108	98	**81**	8	28,634	37	33
7	5,040	449	1,102	**323**	182	1,627,672	1,232	1,689
8	40,320	2,017	20,387	**1,568**	6,274	–	–	–

Table 5.14 Semigroup. The computed counts are consistent with [Inc24b].

Conclusion

Calculating the number of non-isomorphic algebraic structures of a given type is a significant yet challenging mathematical problem, particularly for state-of-the-art automated tools. In this master's thesis, we developed a program to tackle this problem. We explored the implementation and testing of our program on various algebraic structures, demonstrating its effectiveness and providing valuable insights into the complexities involved.

Our program employs a novel technique by computing a canonizing set for a given instance of the model finding problem. This approach allows us to build a compact canonizing symmetry breaking constraint for most instances, which, in turn, enables the construction of propositional formulas whose satisfiability can be decided using modern SAT technologies. By analyzing the satisfying assignments, we can effectively construct the desired algebraic structures.

Notably, our technique enabled us to calculate the numbers of non-isomorphic implication zroupoids of orders 6 and 7, which are unknown according to the surveyed literature. Additionally, we determined the number of non-isomorphic AG-groupoids of order 7, another figure that is unknown to our knowledge. Our program also provided correct counts for well-known algebras and aligned with the counts listed in the literature for lesser-known algebras, showcasing the potential and accuracy of our approach.

However, our implementation generally struggles with algebras of orders higher than 10. We have not utilized many of the known optimization techniques for MACE-style finite model finding, and we have only implemented minor improvements that we discovered. Overall, our implementation is not highly optimized.

We also attempted to compute an instance-independent canonizing symmetry breaking constraint but found this approach to be intractable. From our implementation of the canonizing set algorithms, it appears that for any permutation π, there exists a magma A such that $\pi(A) \prec A$ and for all other permutations π', we have $A \preceq \pi'(A)$. Consequently, it seems that an instance-independent canonizing symmetry breaking constraint for magmas would be of size $\Omega(n!)$. We leave proving this conjecture for future work.

For future work, there are several potential improvements for our program. First, we have not explored advanced optimizations for the propositional encoding of the order relations $\preceq$ and $\prec$ on magmas, aside from optimizations related to fixed points of permutations. In the encoding, some clauses might be implied by others and could be removed from the formula. Additionally, the encoding of $\preceq$ and $\prec$ could potentially be instance-dependent, with the complexity varying depending on the specific algebraic structure. Another significant improvement would be to implement our program in a lower-level language. Currently, our implementation is in Python, which may limit its performance.

Additionally, we did not explore any heuristics regarding the computation of a canonizing set. Our program computes a canonizing set by searching for any counterexample π and then prunes the canonizing set by testing all permutations in the order they appear in the canonizing set (implemented as a list data structure). Based on Example 10, it would be reasonable to first impose additional constraints on the searched π. Similarly, using a heuristic to decide the order in which permutations are tested in the reduction algorithm could potentially result in a smaller set.

Lastly, our current input language is quite limited. We only allow structures with a single binary function, a single unary function, and a few constants. By generalizing the order relations $\preceq$ and $\prec$ to depend on all function tables of a finite model, rather than just the Cayley table of a magma, we could extend our program to find models for a broader range of algebraic structures.

Bibliography

[AS08] Ali Asif and John Slaney. “Counting loops with the inverse property”. In: *Quasigroups and Related Systems* 16.1 (2008), pp. 13–16.

[Bie+20] Armin Biere et al. “CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020”. In: *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*. Ed. by Tomas Balyo et al. Vol. B-2020-1. Department of Computer Science Report Series B. University of Helsinki, 2020, pp. 51–53.

[BL83] László Babai and Eugene M. Luks. “Canonical Labeling of Graphs”. In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*. STOC ’83. New York, NY, USA: Association for Computing Machinery, 1983, pp. 171–183. ISBN: 0897910990. DOI: 10.1145/800061.808746. URL: https://doi.org/10.1145/800061.808746.

[Boo78] Kellogg S. Booth. “Isomorphism Testing for Graphs, Semigroups, and Finite Automata are Polynomially Equivalent Problems”. In: *SIAM Journal on Computing* 7.3 (1978), pp. 273–279. DOI: 10.1137/0207023. eprint: https://doi.org/10.1137/0207023. URL: https://doi.org/10.1137/0207023.

[Boy13] Tim Boykett. “Rectangular groupoids and related structures”. In: *Discrete Mathematics* 313.13 (2013), pp. 1409–1418. ISSN: 0012-365X. DOI: https://doi.org/10.1016/j.disc.2013.03.012.

[Cay90] Arthur Cayley. “On Latin squares”. In: *Oxford, Cambridge and Dublin Messenger of Mathematics* 19.1 (1890), pp. 85–239.

[CM23] A. Chirvasitu and G. Militaru. *A universal-algebra and combinatorial approach to the set-theoretic Yang-Baxter equation*. 2023. arXiv: 2305.14138 [math.QA]. URL: https://arxiv.org/abs/2305.14138.

[Coo72] D. C. Cooper. “Theorem proving in arithmetic without multiplication”. In: *Machine Intelligence* 7 (1972), pp. 91–99.

[Cra+96] James M. Crawford et al. “Symmetry-Breaking Predicates for Search Problems”. In: *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR’96), Cambridge, Massachusetts, USA, November 5-8, 1996*. Ed. by Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro. Morgan Kaufmann, 1996, pp. 148–159.

[CS03] Koen Claessen and Niklas Sörensson. “New Techniques that Improve MACE-style Finite Model Finding”. In: *Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications*. 2003, pp. 11–27.

[CS21] J.M. Cornejo and H.P. Sankappanavar. “Implication Zroupoids and Birkhoff Systems”. In: *Journal of Algebraic Hyperstructures and Logical Algebras* 2.4 (2021), pp. 1–12. ISSN: 2676-6000. DOI: 10.52547/HATEF.JAHLA.2.4.1. eprint: https://jahla.hatef.ac.ir/article_130810_54f16a4d1639a37455de773b217c6191.pdf. URL: https://jahla.hatef.ac.ir/article_130810.html.

[Dis+12] Andreas Distler et al. “The Semigroups of Order 10”. In: *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings.* Ed. by Michela Milano. Vol. 7514. Lecture Notes in Computer Science. Springer, 2012, pp. 883–899. DOI: 10.1007/978-3-642-33558-7_63. URL: https://doi.org/10.1007/978-3-642-33558-7%5C_63.

[DSS11] Andreas Distler, Muhammad Shah, and Volker Sorge. “Enumeration of AG-Groupoids”. In: *Intelligent Computer Mathematics.* Ed. by James H. Davenport et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–14. ISBN: 978-3-642-22673-1.

[Eul82] Leonhard Euler. “Recherches sur une nouvelle espèce de quarrés magiques”. In: *Verhandelingen uitgegeven door het zeeuwsch Genootschap der Wetenschappen te Vlissingen* 9 (1782), pp. 85–239. URL: http://eulerarchive.maa.org/pages/E530.html.

[For55] George E. Forsythe. “SWAC Computes 126 Distinct Semigroups of Order 4”. In: *Proceedings of the American Mathematical Society* 6.3 (1955), pp. 443–447. ISSN: 00029939, 10886826. URL: http://www.jstor.org/stable/2032786.

[FR75] J. Ferrante and C. Rackoff. “A decision procedure for the first order theory of real addition with order”. In: *SIAM Journal on Computing* 4.1 (1975), pp. 69–76.

[Geb+19] Martin Gebser et al. “Multi-shot ASP solving with clingo”. In: *Theory Pract. Log. Program.* 19.1 (2019), pp. 27–82. URL: https://doi.org/10.1017/S1471068418000054.

[IC16] Avraham Itzhakov and Michael Codish. “Breaking Symmetries in Graph Search with Canonizing Sets”. In: *Constraints* 21 (2016), pp. 357–374.

[IMM18] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. “PySAT: A Python Toolkit for Prototyping with SAT Oracles”. In: *SAT.* 2018, pp. 428–437. DOI: 10.1007/978-3-319-94144-8_26. URL: https://doi.org/10.1007/978-3-319-94144-8_26.

[Inc24a] OEIS Foundation Inc. *Entry A001329 in The On-Line Encyclopedia of Integer Sequences.* https://oeis.org/A001329. Accessed: 2024-07-15. 2024.

[Inc24b] OEIS Foundation Inc. *Entry A027851 in The On-Line Encyclopedia of Integer Sequences.* https://oeis.org/A027851. Accessed: 2024-07-15. 2024.

[Inc24c] OEIS Foundation Inc. *Entry A057771 in The On-Line Encyclopedia of Integer Sequences.* https://oeis.org/A057771. Accessed: 2024-07-15. 2024.

[Inc24d] OEIS Foundation Inc. *Entry A057991 in The On-Line Encyclopedia of Integer Sequences.* https://oeis.org/A057991. Accessed: 2024-07-15. 2024.

[Inc24e] OEIS Foundation Inc. *Entry A057992 in The On-Line Encyclopedia of Integer Sequences.* https://oeis.org/A057992. Accessed: 2024-07-15. 2024.

[Inc24f] OEIS Foundation Inc. *Entry A058129 in The On-Line Encyclopedia of Integer Sequences.* https://oeis.org/A058129. Accessed: 2024-07-15. 2024.

[Inc24g] OEIS Foundation Inc. *Entry A362382 in The On-Line Encyclopedia of Integer Sequences.* https://oeis.org/A362382. Accessed: 2024-07-15. 2024.

[Itz23] Avraham Itzhakov. "Symmetry Breaking Constraints for Graph Search Problems". Doctoral Thesis. Ben-Gurion University of the Negev, 2023.

[Jan+24] Mikolás Janota et al. "SAT-Based Techniques for Lexicographically Smallest Finite Models". In: *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada.* Ed. by Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan. AAAI Press, 2024, pp. 8048–8056. DOI: 10.1609/AAAI.V38I8.28643. URL: https://doi.org/10.1609/aaai.v38i8.28643.

[KNW10] George Katsirelos, Nina Narodytska, and Toby Walsh. "On the Complexity and Completeness of Static Constraints for Breaking Row and Column Symmetry". In: *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings.* Ed. by David Cohen. Vol. 6308. Lecture Notes in Computer Science. Springer, 2010, pp. 305–320. DOI: 10.1007/978-3-642-15396-9_26. URL: https://doi.org/10.1007/978-3-642-15396-9%5C_26.

[Lib05] Paolo Liberatore. "Redundancy in logic I: CNF propositional formulae". In: *Artif. Intell.* 163.2 (2005), pp. 203–232. DOI: 10.1016/J.ARTINT.2004.11.002. URL: https://doi.org/10.1016/j.artint.2004.11.002.

[LM17] Jean-Marie Lagniez and Pierre Marquis. "An Improved Decision-DNNF Compiler". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17.* 2017, pp. 667–673. DOI: 10.24963/ijcai.2017/93. URL: https://doi.org/10.24963/ijcai.2017/93.

[Mal19] Martin E Malandro. "Enumeration of finite inverse semigroups". In: *Semigroup Forum.* Vol. 99. 3. Springer. 2019, pp. 679–723.

[McC94] William McCune. *A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems.* Tech. rep. Argonne National Lab., IL (United States). Mathematics and Computer Science Div., 1994.

[McK90] Brendan D. McKay. "*nauty* User's Guide (Version 1.5) Technical Report TR-CS-90-02 (Department of Computer Science". In: *Australian National University, Australian Capital Territory, Australia* (1990).

[McM02] Kenneth L. McMillan. "Applying SAT Methods in Unbounded Symbolic Model Checking". In: *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings.* Ed. by Ed Brinksma and Kim Guldstrand Larsen. Vol. 2404. Lecture Notes in Computer Science. Springer, 2002, pp. 250–264. DOI: `10.1007/3-540-45657-0\_19`. URL: `https://doi.org/10.1007/3-540-45657-0%5C_19`.

[MP14] Brendan D. McKay and Adolfo Piperno. "Practical graph isomorphism, II". In: *J. Symb. Comput.* 60 (2014), pp. 94–112. DOI: `10.1016/J.JSC.2013.09.003`. URL: `https://doi.org/10.1016/j.jsc.2013.09.003`.

[NW13] Nina Narodytska and Toby Walsh. "Breaking Symmetry with Different Orderings". In: *Principles and Practice of Constraint Programming.* Ed. by Christian Schulte. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 545–561. ISBN: 978-3-642-40627-0.

[PS95] Petar V. Protić and Nebojša Stevanović. "AG-test and some general properties of Abel-Grassmann's groupoids". English. In: *PU.M.A., Pure Math. Appl.* 6.4 (1995), pp. 371–383. ISSN: 1218-4586.

[Rea78] Ronald C. Read. "Every one a Winner or how to Avoid Isomorphism Search when Cataloguing Combinatorial Configurations". In: *Algorithmic Aspects of Combinatorics.* Ed. by B. Alspach, P. Hell, and D.J. Miller. Vol. 2. Annals of Discrete Mathematics. Elsevier, 1978, pp. 107–120. DOI: `https://doi.org/10.1016/S0167-5060(08)70325-X`. URL: `https://www.sciencedirect.com/science/article/pii/S016750600870325X`.

[RM21] Olivier Roussel and Vasco M. Manquinho. "Pseudo-Boolean and Cardinality Constraints". In: *Handbook of Satisfiability - Second Edition.* Ed. by Armin Biere et al. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 1087–1129. DOI: `10.3233/FAIA201012`. URL: `https://doi.org/10.3233/FAIA201012`.

[Sin05] Carsten Sinz. "Towards an Optimal CNF Encoding of Boolean Cardinality Constraints". In: *Principles and Practice of Constraint Programming - CP 2005.* Ed. by Peter van Beek. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 827–831. ISBN: 978-3-540-32050-0.

[Tse68] G. S. Tseytin. "On the complexity of derivation in propositional calculus". In: *Studies in Constructive Mathematics and Mathematical Logic, Part II.* Ed. by A. O. Slisenko. Reprinted in J. Siekmann and G. Wrightson (Eds.), Automation of Reasoning: Classical Papers on Computational Logic, 1967-1970, Vol. 2, Springer-Verlag, 1983, pp. 466-483. New York: Consultants Bureau, 1968, pp. 115–125.

[Wal12] Toby Walsh. “Symmetry Breaking Constraints: Recent Results”. In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. Ed. by Jörg Hoffmann and Bart Selman. AAAI Press, 2012. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4974`.

[Zha96] Jian Zhang. “Constructing Finite Algebras with FALCON”. In: *Journal of automated reasoning* 17 (1996), pp. 1–22.

List of Figures

1.1 Concentrically sorted cells with highlighted layers. 9
1.2 Application of $\tau = (1\ 3)$ on the Cayley table of $A = \langle D, \diamond \rangle$. 10
1.3 $A = \langle D, \diamond \rangle$ and $\pi(A) = \langle D, \circ \rangle$ for $\pi = (1\ 2\ 0)$. 10

3.1 $A = \langle D, \diamond \rangle$ and its lexleader $\rho(A) = \langle D, \circ \rangle$ for $\rho = (5\ 0\ 6\ 3)$. . . . 20
3.2 A that satisfies the LNH with $\tau(A) \prec A$. 26

List of Tables

5.1 FOL definitions of the used algebraic structures. 36
5.2 Magma (no restrictions) 38
5.3 AG-groupoid. 39
5.4 Implication zroupoid. 39
5.5 Rectangular groupoid. 40
5.6 Group. 40
5.7 Monoid. 40
5.8 Inverse semigroup. 41
5.9 Commutative quasigroup. 41
5.10 Inverse property loop. 41
5.11 Loop. 42
5.12 Quasigroup. 42
5.13 Right involutory magma. 42
5.14 Semigroup. 42

A Attachments

A.1 Repository

In the attachment, we provide a current snapshot of the GitHub repository containing the implementation of our program. The up-to-date repository is available at `https://github.com/MarekDanco/diplomka`. It contains all our Python source codes, text files with test inputs, compiled propositional formulas in DIMACS encoding the tested instances of the model finding problem, and all logs from our conducted experiments. The specific organization of the repository is the following.

The root directory contains three folders:

1. `dimacs`: Contains folders for each tested structure with all propositional formulas in DIMACS format that we used with the model counting tools `D4` and `clingo`.
2. `logs`: Contains three folders:
   (a) `canset`: Logs from the computation of canonizing sets for each instance.
   (b) `clingo`: Logs from the enumeration of all models with `clingo`.
   (c) `d4`: Logs from the enumeration of all models with `D4`.
3. `structures`: Text files with the FOL axioms of the tested algebraic structures.

Additionally, the root directory contains the following `Python` source codes:

1. `argparser.py`, `basics.py`: Basic command line arguments parsing and procedures used by the rest of the modules.
2. `parsing.py`, `splitting.py`, `grounding.py`: Codes that translate the input FOL axioms into the propositional encoding.
3. `minmod.py`: Builds the propositional encoding of the constraint $A \preceq \pi(A)$.
4. `mkdimacs.py`: Generates the CNF in DIMACS format encoding the given instance of the model finding problem.
5. `canset.py`: Generates a reduced canonizing set for the given instance of the model finding problem.
6. `mace.py`: Constructs all non-isomorphic models for the given instance of the model finding problem.

To use our program, please install the `PySAT` package available at `https://pysathq.github.io/`. The codes `mkdimacs.py`, `canset.py`, `mace.py` can be run from the terminal with command line arguments. The specific format of the command line arguments can be seen by running the codes with the `-h` argument.