SAT-Based Techniques for Lexicographically Smallest Finite Models

Mikoláš Janota[1], Choiwah Chow[1], João Araújo[2, 3], Michael Codish[4], Petr Vojtěchovský[5]

[1]Czech Technical University in Prague, Czechia
[2]Center for Mathematics and Applications (NOVA Math), Portugal
[3]Department of Mathematics, NOVA FCT, Portugal
[4]Ben-Gurion University of the Negev, Beer-Sheva, Israel
[5]University of Denver, USA
mikolas.janota@cvut.cz

Abstract

This paper proposes SAT-based techniques to calculate a specific normal form of a given finite mathematical structure (model). The normal form is obtained by permuting the domain elements so that the representation of the structure is lexicographically smallest possible. Such a normal form is of interest to mathematicians as it enables easy cataloging of algebraic structures. In particular, two structures are isomorphic precisely when their normal forms are the same. This form is also natural to inspect as mathematicians have been using it routinely for many decades.

We develop a novel approach where a SAT solver is used in a black-box fashion to compute the smallest representative. The approach constructs the representative gradually and searches the space of possible isomorphisms, requiring a small number of variables. However, the approach may lead to a large number of SAT calls and therefore we devise propagation techniques to reduce this number. The paper focuses on finite structures with a single binary operation (encompassing groups, semigroups, etc.). However, the approach is generalizable to arbitrary finite structures. We provide an implementation of the proposed algorithm and evaluate it on a variety of algebraic structures.

Introduction

Finite model finding of first-order or higher-order logic has a long-standing tradition in automated reasoning. A number of techniques have been researched in SAT (Claessen and Sörensson 2003), constraint programming (Audemard, Benhamou, and Henocque 2006; Zhang 1996; Zhang and Zhang 1995), or SMT (Reynolds et al. 2013a,b). In theorem proving and software verification, finite models are typically used to identify incorrectly stated theorems. In computational algebra, mathematicians use finite model finding to study fundamental algebraic structures.

This paper does not focus on calculating specific models but on providing a normal form for a given model. This is one of the most prevalent problems in mathematics, i.e., assigning a canonical representative to an equivalence class. For example, the canonical form of a rational fraction is the quotient with the common prime factors removed (reduced fraction); Jordan's canonical form for matrices assigns a matrix to an equivalence class of similar matrices; there are ways of assigning a canonical form to a graph so that any two are isomorphic if and only if their canonical forms are the same, etc. But helping with decision problems is just one of the applications of canonical forms. When we want to enumerate all structures of a given type (e.g., all triangulated 3-manifolds) up to some size (e.g., on 11 vertices (Lutz 2008, 2009)), it suffices to generate the canonical forms and ignore all the rest. These are just a few examples as the applications of canonical forms are countless, including applications to topics as far as chemistry (Weininger, Weininger, and Weininger 1989; Schneider, Sayle, and Landrum 2015). The key feature of canonical systems of representatives is that two objects belong to the same equivalence class if and only if their canonical forms are equal.

A largely widespread technique to assign canonical forms to mathematical objects is to associate each object in the class with a vector and then order all the vectors lexicographically: the canonical object will be the object with the smallest vector. We will call this the *lexicographically smallest representative*, *lexmin* for short. In constraint programming literature, a related term *lex-leader* is defined, cf. Walsh (2012); Peter et al. (2014). Lexmin for graphs is also extensively studied in the literature, cf. Babai and Luks (1983); Crawford et al. (1996).

In computational algebra, this idea naturally translates to concatenating the rows of a multiplication table into a single vector. This canonical form was used as early as 1955 to calculate all the distinct[1] semigroups of order 4. More recently, Jipsen maintains an online database of a variety of mathematical structures stored as lexmin (Jipsen 2016), the GAP package Smallsemi enables calculating lexmin semigroups (Distler and Mitchell 2022).

Figure 1 shows a motivating example of a possible multiplication table for an operation $*$ together with its lexicographically smallest representative $\diamond$. It is relatively easy for a human to detect that $*$ is a quasigroup (aka Latin square), however, further properties are harder to see. In contrast, the multiplication table of $\diamond$ is much easier to comprehend—we see that the operation corresponds to addition modulo 7, which is in fact the unique group of order 7 (the cyclic group $\mathbb{Z}_7$).

[1]Two semigroups are distinct if they cannot be mapped to one another by an isomorphism or by an anti-isomorphism.

$*$	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	7	5	6	1	4	2	3
2	5	3	1	2	6	7	4
3	6	1	5	3	7	4	2
4	1	2	3	4	5	6	7
5	4	6	7	5	2	3	1
6	2	7	4	6	3	1	5
7	3	4	2	7	1	5	6

$\diamond$	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	1	2	3	4	5	6	7
2	2	3	4	5	6	7	1
3	3	4	5	6	7	1	2
4	4	5	6	7	1	2	3
5	5	6	7	1	2	3	4
6	6	7	1	2	3	4	5
7	7	1	2	3	4	5	6

Figure 1: $(D, *)$ and its lexmin $(D, \diamond)$ for $D = \{1..7\}$.

Developing efficient algorithms for calculating the lexmin form is paramount in the field of computational algebra:

- It enables presenting a concrete algebra in a familiar way to researchers.
- Computational algebra systems, such as GAP (GAP4), contain a large number of packages for handling algebras for specific forms and lexmin provides a uniform exchange format between these packages.
- Lexmin provides a uniform way of storing and recalling algebras. The form is especially interesting for prefix trees (tries) since inherently, many algebras will share the same prefix in the lexmin form.

This paper presents the following contributions.

- We develop a SAT-based algorithm that enables calculating the normal form on the fly, rather than working with explicit representation of the target normal form.
- We design a variety of propagation techniques that enable avoiding SAT calls in a large number of cases, which has proven indispensable in many real-world problems.
- We provide a prototype implementation of the proposed algorithm, using state-of-the-art SAT solvers in a black box fashion. This prototype is evaluated on a number of algebras that mathematicians deal with on daily basis.

Preliminaries

Throughout the paper we focus on finite mathematical structures with a single binary operation, hereafter referred to as *magmas* (the term *groupoid* is also used in the literature). For instance, any finite group or semigroup is a magma. Magmas are denoted by a pair $(D, \circ)$ where D is the domain and $\circ$ a binary operation on D. We rely on the well-established term of isomorphism.

Definition 1 (isomorphism)**.** A bijection $f : D_1 \to D_2$ is an isomorphism from a magma $(D_1, *)$ to $(D_2, \diamond)$ if $f(a * b) = f(a) \diamond f(b)$, for all $a, b \in D_1$. Two magmas are *isomorphic* iff there exists at least one isomorphism between them.

Throughout the paper, we consider a finite domain $D = \{1, \ldots, n\}$ for $n \in \mathbb{N}^+$. The goal is to obtain the lexicographically smallest $(D, \diamond)$ isomorphic to the given $(D, *)$.

Definition 2 ($\preceq$)**.** Define a total order $\preceq$ on magmas on domain D as follows. For magmas $A = (D, *)$ and $B = (D, \diamond)$, we have $A \preceq B$ iff $1{*}1, 1{*}2, \ldots, 1{*}n, 2{*}1, \ldots, n{*}n$ is lexicographically smaller or equal to $1 \diamond 1, 1 \diamond 2, \ldots, 1 \diamond n, 2 \diamond 1, \ldots, n \diamond n$.

Definition 3 (LEXMIN)**.** For magma $A = (D, *)$, magma $B = (D, \diamond)$ is the *lexicographically smallest representative (lexmin)* of A iff B is the $\preceq$-least element among all magmas $(D, \diamond')$ isomorphic to A. The LEXMIN problem is finding the lexicographically smallest representative of A.

In several cases we rely on the notion of an idempotent, which is invariant under isomorphism.

Definition 4 (idempotent)**.** For a magma $(D, *)$, an element $a \in D$ is an *idempotent* iff $a * a = a$.

Observation 5. Let $A = (D_1, *)$ and $B = (D_2, \diamond)$ be isomorphic magmas under some isomorphism f, and let a be an idempotent of A, then $f(a)$ is an idempotent of B.

Example 6. This example shows a multiplication table for a small magma $(D, *)$ with $D = \{1, 2\}$ together with an extensive representation as a set of assignments. On the right-hand side, we see its lexicographically smallest representative $\diamond$. The corresponding isomorphism swaps 1 and 2, i.e., $f(1) = 2, f(2) = 1$, alternatively represented as a permutation in the cyclic notation $(1\,2)$.

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

$1 * 1 = 1$
$1 * 2 = 2$
$2 * 1 = 2$
$2 * 2 = 2$

$\diamond$	**1**	**2**
1	1	1
2	1	2

$2 \diamond 2 = 2$
$2 \diamond 1 = 1$
$1 \diamond 2 = 1$
$1 \diamond 1 = 1$

Note that the isomorphism not only changes the contents of the table but also permutes rows and columns. In this example, to obtain $\diamond$ from $*$ means swapping rows 1 and 2, columns 1 and 2, and values 1 and 2 in the table.

Example 6 also illustrates that properties based on equality are preserved: both tables contain a row with all elements distinct, have 2 idempotents, etc. This is a more general property, which we state here informally.[2]

Observation 7. Any property of $A = (D, \diamond)$ that does not rely on the names of elements of D is preserved in all isomorphic copies of A.

Note that in the small Example 6, there is a unique isomorphism from the input magma to its lexmin but in general, there may be many—despite the fact that the lexmin is unique. We conclude the preliminaries by relating isomorphism to lexicographic representatives.

Observation 8. Magmas $A = (D, *)$ and $B = (D, \diamond)$ are isomorphic iff their lexicographically smallest representatives are equal.

The isomorphism problem for finite magmas is graph-isomorphism-hard (GI-hard) even if we consider only semigroups (Zemljachenko, Korneenko, and Tyshkevich 1982). Further, deciding whether an incidence matrix of a graph is lexmin is NP-hard (Babai and Luks 1983), despite the fact that GI is believed to be easier than NP. Therefore, we do not expect the LEXMIN problem for general magmas to be computationally easy.

[2] More precisely, a set S defined by an FOL formula in a magma A corresponds to the set $f(A)$ in B for an isomorphism f from A to B, *cf.* Theorem 1.1.10 in (Marker 2002).

Explicit Encoding

A straightforward approach to the lexmin problem is to encode to SAT that a target (unknown) magma $(D, \diamond)$ is isomorphic to a given magma $(D, *)$. Then, we can apply standard algorithms for finding the lexicographically smallest magma $(D, \diamond)$, cf. (Nadel and Ryvchin 2016; Trentin 2019; Petkovska et al. 2016; Marques-Silva et al. 2011).

Effectively, $(D, \diamond)$ is represented in 1-hot encoding. First represent an isomorphism $f : D \to D$ by introducing Boolean variables $x_{i \to j}$ meaning that $f(i) = j$ for $i, j \in D$. Second, introduce additional Boolean variables $x_{i,j:v}$ meaning that $i \diamond j = v$.

To ensure that the $x_{i \to j}$ variables represent a bijection, generate cardinality constraints (converted to CNF by standard means (Roussel and Manquinho 2021)).

$$\begin{aligned}&\mathtt{X}(D) := \\ &\quad \left\{ \sum\nolimits_{j \in D} x_{j \to i} = \sum\nolimits_{j \in D} x_{i \to j} = 1 \mid i \in D \right\}\end{aligned} \tag{1}$$

To ensure that $x_{i,j:v}$ represent an isomorphic $(D, \diamond)$, generate implications covering possible mappings between rows, columns, and values.

$$\begin{aligned}&(x_{r \to r'} \wedge x_{c \to c'} \wedge x_{r * c \to v'}) \Rightarrow x_{r',c':v'}, \\ &\text{for } r, r', c, c', v' \in D\end{aligned} \tag{2}$$

Note that for row r and column c, the value $r * c$ is given.

An advantage is that we can easily apply any bitlevel lexicographic optimization algorithms over the vector of variables representing the magma $(D, \diamond)$, in the following order $x_{1,1:n}, x_{1,1:n-1}, \ldots, x_{1,1:1}, x_{1,2:n}, \ldots, x_{n,n:1}$. A significant disadvantage is the sheer size of the encoding, which involves $\Theta(|D|^5)$ clauses. Therefore we propose a solution where the explicit representation of $(D, \diamond)$ is not necessary.

Gradual Construction

Instead of introducing variables for the unknown $(D, \diamond)$, we construct it gradually starting from its top-left corner, continuing by filling the first row and then the second, and so on. Here we avail of the concept of isomorphic copy, which is a magma induced by an isomorphism.

Definition 9 (isomorphic copy)**.** Consider a magma $(D_1, *)$ and a bijection $f : D_1 \to D_2$ then the *isomorphic copy* $(D_2, \diamond)_f$ is defined as $a \diamond b = f(f^{-1}(a) * f^{-1}(b))$. In the remainder of the paper, we omit the subscript f from $(D_2, \diamond)_f$, whenever it is clear from the context that f is present.

The intuition behind an isomorphic copy is that to obtain the value $a \diamond b$, we first obtain the pre-images of a and b, then apply the (known) operation $*$ to the pre-images in the context of $(D_1, *)$, and finally map the result back to $(D_2, \diamond)$. This is well-defined because f is a bijection.

Observation 10. Magmas $A = (D_1, *)$ and $B = (D_2, \diamond)$ are isomorphic iff there exists a bijection $f : D_1 \to D_2$ such that B is an isomorphic copy of A by $f : D_1 \to D_2$.

To construct $(D_2, \diamond)$, we will need to encode the constraints of the shape $r \diamond c = v$, e.g., $1 \diamond 1 = 1$ means placing 1 in the top left corner of the multiplication table. Since

Algorithm 1: Calculate lexmin $(D, \diamond)$ for given $(D, *)$ by gradual construction.

```
A ← ∅      // empty set of assignments
for r, c ∈ 1..|D|, 1..|D| do
    v ← 1
    while ¬SAT(X(D) ∪ enc(A ∪ {r ⋄ c = v})) do
        v ← v + 1
    A ← A ∪ {r ⋄ c = v}          // extend A
```

$(D, \diamond)$ must be an isomorphic copy of $(D, *)$, the constraint $r \diamond c = v$ can be written as follows:

$$f(f^{-1}(r) * f^{-1}(c)) = v, \tag{3}$$

where f is an unknown permutation of D. As in the previous encoding, we encode f as Boolean variables $x_{i \to j}$ coupled with the appropriate cardinality constraints (see (1)). The equality (3) yields a set of implications covering all possible values of f.[3]

$$\begin{aligned}&\mathtt{enc}(r \diamond c = v) := \\ &\quad \{(x_{i \to r} \wedge x_{j \to c}) \Rightarrow x_{i * j \to v} \mid i, j \in D\}\end{aligned} \tag{4}$$

Algorithm 1 shows how the lexmin representative is calculated by maintaining a set of equalities A of the form $r \diamond c = v$ for which we already know that they must hold in the multiplication table of $(D, \diamond)$ (this is a loop invariant of the outer loop). The inner loop attempts to extend the set of assignments A for the next cell of the multiplication table going from 1 to higher values. The call to the function `enc` conjoins the encoding of the assignments according to the equation (4) along with the bijection constraints (1).

The algorithm first tries placing 1 in the top left corner and if that is possible it moves onto the next column. Otherwise, it tries placing 2 in the top left corner, and so forth. Once it succeeds in placing a value in a cell, the value is fixed. The algorithm leads to $O(|D|^3)$ SAT calls. The permutation f, represented by the Boolean variables $x_{i \to j}$, spans all permutations and therefore enables the creation of any isomorphic copy of $(D, *)$ on the domain D. This also justifies termination of the inner loop because one of the SAT calls is bound to succeed since the set of isomorphic copies is always nonempty—it, for instance, contains the input magma itself. Since $|A| \in O(|D|^2)$ and (4) requires $O(|D|^2)$ clauses, Algorithm 1 requires space for $O(|D|^4)$ clauses.

Efficiency Improvements

Algorithm 1 faces two major pitfalls: a high number of SAT calls, and hard individual SAT calls. The upper bound of $O(|D|^3)$ on SAT calls in Algorithm 1 is tight. For instance, for quasigroups (aka Latin squares) it is also $\Omega(|D|^3)$.[4] The second issue, where an individual SAT call might be too hard, is potentially even more worrisome.

[3] The implementation avoids repeated and tautologous clauses.

[4] Each row of a quasigroup contains all the elements of D, therefore each row requires $\frac{n(n-1)}{2}$ SAT calls as v calls are needed for a cell containing the value v.

Indeed, we have a reason to believe that some SAT calls will be hard due to an underlying pigeonhole principle. For instance, if the original magma $(D, *)$ does not contain any element more than k-times on any given row, the same must hold for the target magma $(D, \diamond)$. Then, for the SAT solver to prove that it cannot place an element for the $k+1$-th time on the same row is indeed reminiscent of the pigeonhole principle formulas, which are well known to be difficult for SAT (and resolution in general) (Haken 1985; de Rezende et al. 2020). Such hard SAT calls could get the Algorithm 1 simply stuck on a single cell.

Here we focus on designing new propagation techniques that let us bypass calls to the SAT solver in specific scenarios. We focus mainly on techniques that rely on counting because that is a famous Achilles' heel for modern SAT solvers. We begin with a technique that enables in some cases identifying the first row.

Identification of the First Row

Recall that any row r of the original magma $(D, *)$ must be projected to some row $r' = f(r)$ in the target magma. Here we show that in certain cases it is possible to identify possible candidates that might be mapped to the first row, i.e., we construct a set $C_1 \subseteq D$, s.t. $f(a) = 1$ only if $a \in C_1$. This is encoded into the SAT solver as a set of unit clauses: $\{\{\neg x_{a \to 1}\} \mid a \notin C_1\}$ before Algorithm 1 starts.

Suppose that $4 * x = 4$, for all $x \in D$, i.e., the 4th row is entirely filled with 4's. If 4 is renamed to 1, i.e., pick an isomorphic copy with $f(4) = 1$, the first row of $\diamond$ becomes all 1's, i.e., lexicographically smallest first row possible. We generalize this idea to find candidates for the first row of $\diamond$.

Definition 11. Let $A = (D, *)$ be a magma with some idempotents. The *idempotent apex* of A is the largest value of $|\{x \in D \mid e * x = e\}|$, for $e \in D$ idempotent of A.

Possible rows that can be mapped to the first row are obtained by calculating for each row r of $*$ that contains an idempotent, how many times r appears in it, i.e., $o_r := |\{c \in D \mid r * c = r\}|$, if $r * r = r$. We claim that only a row that maximizes this number can become the first row in the smallest representative $\diamond$, i.e., $f(r) = 1$ implies o_r is the apex of the input magma. If the input magma does not contain any idempotents, this technique is not applied. Note that in the example of Figure 1 only row 4 contains an idempotent and therefore it necessarily must become the first one.

We proceed with the correctness proof of this statement. For succinctness we introduce the following notation. We write $[(D, *)]$ for the set of isomorphic copies $(D, \diamond)$ isomorphic to $(D, *)$. We write $\downarrow(D, *)$ for the lexicographically smallest representative according to the ordering $\prec_r$ (by-rows). We write $1 \diamond \{1, \ldots, k\} = \{1\}$ as a shorthand for $1 \diamond i = i$, for $i \in 1..k$, which effectively means that the first k columns of the first row of $\diamond$ are equal to 1.

Proposition 12. *Let $A = (D, *)$ be a magma with idempotents and idempotent apex k. Let $\mathcal{M}_k := \{(M, \diamond) \in [(D, *)] \mid 1 \diamond \{1, \ldots, k\} = \{1\}\}$. Then*

1. *$\mathcal{M}_k \neq \emptyset$;*
2. *$\downarrow(D, *) \in \mathcal{M}_k$.*

Proof. Let $e \in D$ such that $e * e = e$ and $D_0 := \{x \in D \mid e * x = e\}$ has size k. Pick g, a permutation of D, such that $g(D_0) = \{1, \ldots, k\}$ and $g(e) = 1$. Define on D the following operation: $x \diamond y := g(g^{-1}(x) * g^{-1}(y))$, for all $x, y \in D$. For all $x \in \{1, \ldots, k\}$, we have

$$1 \diamond x = g(g^{-1}(1) * g^{-1}(x)) = g(e * g^{-1}(x)) = g(e),$$

because $g^{-1}(x) \in D_0$ and $e * a = e$, for all $a \in D_0$. It is proved that $1 \diamond x = g(e) = 1$, for all $x \in \{1, \ldots, k\}$. In addition, $x \diamond y := g(g^{-1}(x) * g^{-1}(y))$ implies that (replacing x with $g(x)$ and y with $g(y)$) $g(x) \diamond g(y) = g(g^{-1}(g(x)) * g^{-1}(g(y))) = g(x * y)$. It is proved that g is an isomorphism of the magmas $(D, \diamond)$ and $(D, *)$. Therefore $(D, \diamond) \in \mathcal{M}_k$. The first claim follows.

Regarding the second claim, suppose that $(D, \times)$ is a lexmin of $(D, *)$. Since M_k is not empty, we must have $1 \times j = 1$ for all j in $1, \ldots, i$ and some $i \geq k$. Since the idempotent apex is preserved by isomorphism (see Observation 7), we have $i \leq k$. Hence $i = k$ and $(D, \times)$ is in M_k. □

Budgeting

Next, we describe a technique that is invoked for every SAT call of Algorithm 1. Roughly speaking, each element $a \in D$ is assigned a budget, which is decremented whenever a is placed in the target table. SAT calls $r \diamond c = v$ with values v that have 0 budget are not invoked (and deemed unsatisfiable). We consider budgets per row/column or for the whole table. In the context of constraint programming, similar propagation techniques are abundantly used for global constraints (Peter et al. 2014, Chapter 3).

For intuition, consider a situation where each row of the multiplication table of $*$ contains *at most one* occurrence of any given element (as in the example Figure 1). Then the same property will hold in the rows of $\diamond$ by Observation 7. This means that if Algorithm 1 has placed an element a in a certain row, it does not need to try placing it in the same row again. This enables the algorithm to skip SAT calls on values that are no longer possible (in that row).

This idea is readily generalized to an arbitrary number of occurrences. Define $o_*(r, a) = |\{c \mid r * c = a,\ c \in D\}|$ and calculate $\max\{o_*(r, a) \mid r, a \in D\}$ to give a budget for an arbitrary element in an arbitrary row of $(D, \diamond)$. The same can be applied to columns and the total number of occurrences in the table. This is especially useful for quasigroups, where each element appears precisely once in each row/column.

The budget calculated as described above is an upper bound, which can sometimes be improved. Consider the case when the first row was uniquely identified by the technique outlined in the previous section. Then we have established that $f(k) = 1$ for some $k \in D$, for any f yielding the lexmin copy. This enables splitting budgets for the element 1 and the rest of the elements according to the following equalities.

$$\max\{o_*(r, k) \mid r \in D\} = \max\{o_\diamond(r, 1) \mid r \in D\} \tag{5}$$

$$\begin{aligned}\max\{o_*(r, a) \mid a \neq k \wedge r, a \in D\} = \\ \max\{o_\diamond(r, 1) \mid a \neq 1 \wedge r, a \in D\}\end{aligned} \tag{6}$$

Row Invariants

As shown above, the budgeting technique can benefit from knowing which element has been mapped to the first row. More generally, once it is established that $f(k) = j$, for some $k, j \in D$, it must hold that the number of occurrences of k in $(D, *)$ will be equal to the number of occurrences of j in the copy $(D, \diamond)$. But how to establish such correspondence? Note that the variables $x_{i \to j}$ determine the permutation on the elements of D but this permutation may change over the course of the algorithm.

From the definition of isomorphic copy (Definition 9), the contents of a row of the original table of $(D, *)$ must correspond to the contents of some row of the table of $(D, \diamond)$. More precisely, the bag of elements $[r * c \mid c \in D]$ is equal to the bag of elements $[f(r) \diamond c \mid c \in D]$. In some cases, this lets us unequivocally identify that a row r in the original magma maps to a row r' in the isomorphic copy. This is done by calculating *invariants* (properties invariant under isomorphism) and matching pairs of rows with unique invariants. Currently, we use the following invariants bundled into a single one. Similar invariants have been used before for isomorphism testing (Araújo, Chow, and Janota 2021, 2022; Nagy and Vojtěchovský 2018).

- $|\{r \circ c = c \mid c \in D\}|$, for fixed $r \in D$ and $\circ \in \{*, \diamond\}$
- $|\{r \circ c = r \mid c \in D\}|$, for fixed $r \in D$ and $\circ \in \{*, \diamond\}$
- $|\{r \circ r = r\}|$, for fixed $r \in D$ and $\circ \in \{*, \diamond\}$
- define $g_r(a) = r \circ a$ and $m_r(a)$ as minimal k s.t. $g_*^k(a) = g_*^j(a)$ for some $j < k$. Take the bag $[m_r(c) \mid c \in D]$ as invariant, for fixed r and $\circ \in \{*, \diamond\}$.

For the example in Figure 1, only row 4 has 7 columns c s.t. $4 * c = c$ and $m_4(c) = 1$. The invariants are used in Algorithm 1 as follows. Each time a row r of $(D, \diamond)$ is entirely filled, its invariant is calculated and if there is a *unique* row r' in the input table $(D, *)$ with the same invariant, set $f(r') = r$, add the corresponding unit clause $\{x_{r' \to r}\}$ and recalculate budgets.

We also exploit invariants even if they do not give us a unique correspondence of rows. In the case that an invariant is shared by k rows in $*$ and it already appears k times in the partially filled copy $\diamond$, subsequent rows will never be mapped to the ones that gave rise to the invariant in question. More concretely, if there is a set of rows $R \subseteq D$ with $|R| = k$ that correspond to a certain invariant I and the invariant I already appears k times in the first r rows of $\diamond$ then for $f(r') \neq j$ for $j \in R$ and $r' > r$. In the implementation, corresponding unit clauses are inserted into the SAT encoding once that takes place. We remark the same technique could be applied to columns but it would not be useful since columns are never complete until the very end.

Mid-Row Budgeting Refinement

The techniques described in the previous section enable refining budgets after a row of the target table is filled. Here we also show that this can be done mid-row. We propose a cheap technique that is easy to implement where we split the rows of $*$ into rows containing an idempotent and into rows that do not. Note that row r contains an idempotent iff $r * r = r$. This lets us calculate three types of budgets: (1) for all rows of $*$; (2) for rows of $*$ containing an idempotent; (3) for rows of $*$ not containing an idempotent.

When Algorithm 1 starts filling a row r, it does not know in which group the row falls and therefore starts with the global budget. Once the r^{th} position is filled, the budget can be refined accordingly. In the row-based traversal, in the first row, the refinement happens once the top left corner has been filled (the first column of the first row).

Upper Bound by Last Value

A simple improvement is obtained by inspecting the model obtained from satisfiable SAT calls. Even though Algorithm 1 only imposes assignments to the table $\diamond$ for those cells that have been traversed so far, any SAT model represents a permutation for all the elements in the domain D, from which one can infer the rest of the table of $\diamond$. The remainder (untraversed) of the table does not necessarily guarantee that it is lexicographically smallest but it gives us an upper bound. This means that for each cell $(r, c) \in D \times D$ there is always a tentative value v_u for which we already have a witnessing permutation. This lets us avoid the SAT call for the query $r \diamond c = v$ for any $v \geq v_u$. This upper bound is also used in different search strategies described in the upcoming section. We remark that an analogous technique has also been used for explicit representation-based calculation of lexicographically smallest SAT assignment (Knuth 2015).

Search Strategies

Algorithm 1 performs $|D|$ tests for a single cell of the table $\diamond$ in the worst case. It is tempting to apply standard techniques for minimization, such as binary search. However, these are not directly applicable because the behavior is not monotone, e.g., it might be possible to place 3 and 7 in a specific cell, but not 5. Nevertheless, monotone behavior can be obtained by constructing SAT queries over a *disjunction* of values. Hence, instead of querying $r \diamond c = v$, we query $\bigvee_{v \in V} r \diamond c = v$ over some set of $V \subseteq D$. In terms of the SAT encoding, one could calculate a disjunction over the encoding for a single value (equation (4)) but we are able to avail of the common part and $r \diamond c \in V$ is encoded as follows.

$$\{(x_{i \to r} \wedge x_{j \to c}) \Rightarrow \bigvee_{v \in V} x_{i * j \to v}, \mid i, j \in D\} \qquad (7)$$

This approach has monotone behavior in the sense that if $r \diamond c \in V$ is satisfiable then also $r \diamond c \in V'$ is satisfied for any $V \subseteq V'$. This enables us to use standard MaxSAT iterative techniques, where the basic Algorithm 1 is in fact a linear UNSAT-SAT strategy. Additionally, taking into account values obtained from satisfiability calls enables improving the upper bound for linear SAT-UNSAT or binary search.

In our experiments, standard binary search did not perform well because it still requires $\Omega(\log_2 |D|)$ SAT calls to prove an optimum. Therefore we apply a modified binary search where first we test if the optimum has not already been reached. In the case that the optimum has not been reached, the upper bound is updated. If the upper bound reduced the search space by a factor of 2, we simply recur. If the upper bound falls into the top half of the possible values, another SAT call is issued.

Structure	Definition in FOL
Groups	$x * (y * z) = (x * y) * z,\ x * e = x,$ $x * e = x,\ x * x' = e,\ x' * x = e$
Loops	$x * y = x * z \to y = z,\ y * x = z * x \to$ $y = z, e * x = x, x * e = x$
Quasigroups	$x * y = x * z \to y = z,\ y * x = z * x \to$ $y = z$
Semigroups	$x * (y * z) = (x * y) * z$
Magmas	no requirement

Table 1: FOL definitions of the used algebraic structures.

Experiments

The experiments are run on an Intel® Xeon®CPU E5-2630 v2 2.6 GHz ×24 computer, with 64 Gb RAM. We call our tool `mlex` and it supports two SAT solvers, `minisat` (Eén and Sörensson 2003) and `cadical` (Biere 2017). Unless otherwise stated, `minisat` is used in our experiments. Both SAT solvers are used incrementally and `cadical` is used via the `IPASIR` interface (Balyo et al. 2016). We excluded the Explicit Encoding from the evaluation since it led to unwieldy memory consumption (dozens of gigabytes even for small problems). The GAP package Smallsemi (Distler and Mitchell 2022) provides a function to calculate lexmin semigroup, which is not included in the evaluation because the ordering used traverses the table by the diagonal first and the implementation suffers from timeouts and large memory consumption even on small problem instances (order 20). Hence, the evaluation is based on Algorithm 1 and its extensions described in the Efficiency Improvements section.

The tool was evaluated on several popular algebraic structures (algebras) defined in Table 1. In this table, "e" is a constant, "$*$" is a binary operation, and "$'$" unary; all clauses are implicitly universally quantified. Even though `mlex` currently supports only a single binary operation, it can handle all these algebras. This is because in many finite algebraic structures, such as those listed here, the constant and the unary function are uniquely determined by the binary operation. Hence, they can be removed from the inputs to `mlex`.

The evaluation was performed on randomly generated samples from five algebraic structures: groups, loops, general magmas, quasigroups, and semigroups.For groups, we randomly pick the groups given by the `AllSmallGroups` function in GAP. For magmas and semigroups, we generate them with the help of GAP functions such as `Random`. For quasigroups and loops, we use the `RandomQuasigroup` and `RandomLoop` functions in the LOOPS package in GAP. We make sure the models in each structure do not belong to a sub-structure in the list above. For example, the magmas we use are not semigroups or quasigroups. We consider a total of 210 random samples of the five algebraic structures listed in Table 1, of orders 16 to 128 in increments of 16. In addition, we include random samples of 5 magmas of each of the orders 192 and 256. Finally, a timeout of 30 minutes is used for calculating the lexmin copy of each model.

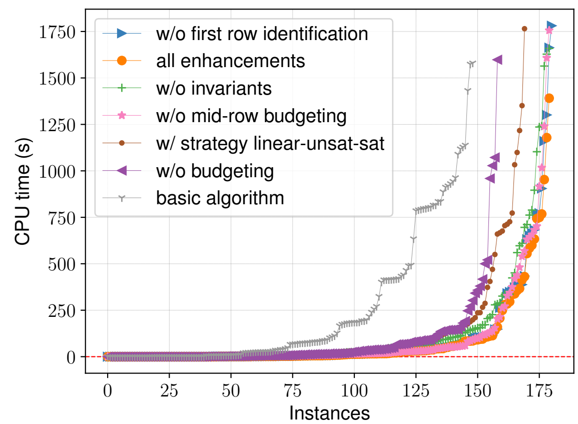

Figure 2: Performance of `mlex` with different options.

Ablation Study of Techniques

We test the introduced techniques in an ablation study. We consider basic Algorithm 1, a version with all improvements turned on, and the effect of turning off each one of them individually. For search strategies, we compare between linear-unsat-sat (`lus`) and modified binary search (`bin2`).

Figure 2 shows a cactus plot for the ablation study. Although all the techniques lead to an improvement in the tool, the most significant is the use of budgeting, which confirms our suspicion that hard SAT calls might occur due to counting arguments. Interestingly, the binary search technique also has a significant impact. Turning off the other techniques does not have a significant impact on the number of solved instances. However, there are specific classes of problems that cannot be solved without using all the techniques. Also, the "all enhancement" version of the solver appears to be the fastest and the most robust version.

It is well-known that `minisat` is simple and fast and that for more complex problems, `cadical` usually performs much better (Dutertre 2020). This pattern is also observed with `mlex`. As shown in the cactus diagram Figure 3, when enhancement features are turned on, then for simpler problems that take a shorter time, `minisat` usually solves more problems for the same time, but for more complex problems, the opposite is true. However, as also shown in the same diagram, the choice of other input options to `mlex` has a much more pronounced impact on the speed of `mlex` than the underlying SAT solver, as the curves corresponding to both SAT-solvers are very close for the same set of input options. Surprisingly, `cadical` performs poorly compared to `minisat` when all improvements are turned off.

Related Work

Finite model finding is ubiquitous to automated reasoning. Sometimes, users are interested in models rather than in proving a theorem (McCune 1994). In theorem proving, models serve as counterexamples to invalid conjectures (Blanchette 2010), which also appear in software veri-

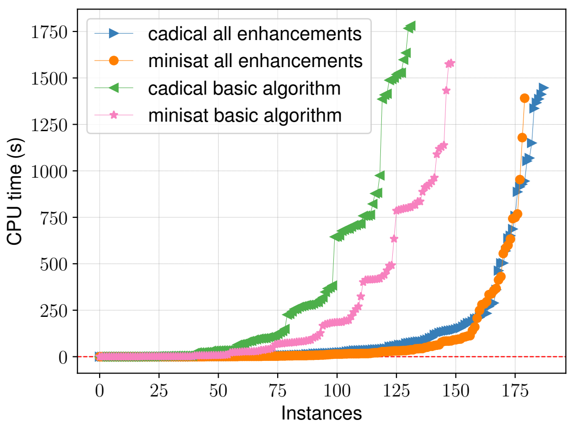

Figure 3: Comparison of `minisat` and `cadical` in `mlex`.

fication (Torlak and Jackson 2007). Finite models have also been used as a semantic feature for *lemma selection learning* (Urban et al. 2008). In certain fragments, finite model finding provides a complete decision procedure, e.g., the *Bernays-Schönfinkel* fragment (EPR). Throughout the years, CP, SAT, and SMT tools have been used in finite model finders (Audemard, Benhamou, and Henocque 2006; Claessen and Sörensson 2003; Reynolds et al. 2013a,b; Zhang 1996; Zhang and Zhang 1995; Araújo, Chow, and Janota 2023). SAT and CP are routinely used to solve algebraic problems (Heule 2018; Distler et al. 2012; Janota, Morgado, and Vojtechovský 2023).

It is important to note that finite models are also constructed by dedicated approaches based on deep domain knowledge. Notably, the algebraic system GAP (GAP4) contains a number of packages for specific types of algebraic structures. The Small Groups library (Besche, Eick, and O'Brien 2002) contains *all* ($\approx 4 \times 10^8$) non-isomorphic groups up to order 2000 (except for order 1024). Similarly, Smallsemi (Distler and Mitchell 2022) catalogues semigroups and LOOPS packages loops (Nagy and Vojtěchovský 2018). However, currently, these packages do not provide the lexicographically smallest representative. Adding our tool into GAP is a subject of future work.

Normal forms are ubiquitous in computer science and mathematics. Here we highlight the canonical labeling algorithms implemented in the `nauty` system (McKay and Piperno 2014). The system has been developed since the 80's and it is considered state-of-the-art for graph isomorphism (and more). It is possible to construct a canonical form of a magma by using `nauty`: for a magma A, construct a special graph G'_A and find its canonical graph G_A, cf. (Khan 2020). This form is canonical in the sense that two isomorphic magmas will give the same canonical graph but the resulting graph is opaque to the user. Hence, it cannot be used for solving the problem tackled in this paper.

A large body of research exists on *symmetry breaking* in SAT and CP (Peter et al. 2014; Sakallah 2021). In general, however, the objective of symmetry breaking is different from our objective: it is a means speeding up search by avoiding symmetric parts of the search space. In contrast, in our case, the normal form is the objective. Typically, symmetry breaking is meant to be fast, when used dynamically, or should add a small number of constraints, when used statically (Codish et al. 2018; Itzhakov and Codish 2020). Therefore, symmetry breaking is often incomplete. Even though, Heule investigates optimal complete symmetry breaking for small graphs (≈ 5 vertices) (Heule 2019). Kirchweger and Szeider (2021) develop a specific symmetry breaking, called *SAT Modulo Symmetries*, where a SAT solver is enhanced to look for the lexicographically smallest graph (similarly to lazy SMT). There, the objective is to enumerate non-isomorphic graphs with certain properties. More broadly, this paper fits into the SAT+CAS paradigm, where SAT is combined with *computer algebra systems*, cf. Bright, Kotsireas, and Ganesh (2022).

Conclusions and Future Work

This paper tackles the problem of calculating the lexicographically smallest representative of a given algebraic structure. This is a fundamental problem in computational algebra, where the user, a mathematician, requires a *specific canonical form*. A prominent feature of this canonical form is that it enables a "common language" between different mathematical libraries and it enables the mathematicians to identify familiar patterns and structures.

Our prototype of the proposed algorithms shows that the SAT technology is up to the task. The proposed encoding enables tackling large problem instances by avoiding explicitly representing the target structure. The SAT solver is used in a black box fashion with repeated SAT calls, which gradually construct the targeted structure (the lexicographically minimal representative). We further design a number of dedicated techniques that enable simplifying, or completely avoiding, certain SAT calls. The experimental evaluation shows that the approach decidedly benefits from this additional propagation (done outside of the SAT solver).

This work opens a number of avenues for further research. More powerful propagation techniques still could be considered—such as different invariants and more aggressive and fine-grained propagation. A tighter integration with the SAT solver and application to structures with several multiplication tables is more of an engineering effort but would further increase the practicality of the implemented tool. Rather than invoking the approach on a *given* structure, it would also be interesting to integrate it into the calculation of non-isomorphic structures under constraints.

Acknowledgements

We thank Brendan D. McKay for helpful comments. The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902 and by national funds through the FCT — Fundação para a Ciência e a Tecnologia, I.P., under the scope of the project UIDB/00297/2020 (doi.org/10.54499/UIDB/00297/2020) and the project UIDP/00297/2020 (doi.org/10.54499/UIDP/00297/2020)

(Center for Mathematics and Applications) and co-funded by the European Union under the project *ROBOPROX* (reg. no. CZ.02.01.01/00/22_008/0004590). This article is part of the *RICAIP* project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306. P. Vojtěechovský was supported by the Simons Foundation Mathematics and Physical Sciences Collaboration Grant for Mathematicians no. 855097.

References

Araújo, J.; Chow, C.; and Janota, M. 2023. Symmetries for Cube-And-Conquer in Finite Model Finding. In Yap, R. H. C., ed., *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Araújo, J. a.; Chow, C.; and Janota, M. 2021. Filtering Isomorphic Models by Invariants. In Michel, L. D., ed., *27th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 4:1–4:9. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-211-2.

Araújo, J. a.; Chow, C.; and Janota, M. 2022. Boosting isomorphic model filtering with invariants. *Constraints*, 27: 1–20.

Audemard, G.; Benhamou, B.; and Henocque, L. 2006. Predicting and Detecting Symmetries in FOL Finite Model Search. *J. Autom. Reason.*, 36(3): 177–212.

Babai, L.; and Luks, E. M. 1983. Canonical Labeling of Graphs. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, 171–183. New York, NY, USA: Association for Computing Machinery. ISBN 0897910990.

Balyo, T.; Biere, A.; Iser, M.; and Sinz, C. 2016. SAT Race 2015. *Artificial Intelligence*, 241: 45–65.

Besche, H. U.; Eick, B.; and O'Brien, E. A. 2002. A Millennium Project: Constructing Small Groups. *Int. J. Algebra Comput.*, 12(5): 623–644.

Biere, A. 2017. CaDiCaL, Lingeling, PLingeling, Treengeling and YalSAT Entering the SAT Competition 2017.

Blanchette, J. C. 2010. Nitpick: A Counterexample Generator for Isabelle/HOL Based on the Relational Model Finder Kodkod. In Voronkov, A.; Sutcliffe, G.; Baaz, M.; and Fermüller, C. G., eds., *Short papers for 17th International Conference on Logic for Programming, Artificial intelligence, and Reasoning, LPAR-17-short*, volume 13 of *EPiC Series in Computing*, 20–25. EasyChair.

Bright, C.; Kotsireas, I. S.; and Ganesh, V. 2022. When satisfiability solving meets symbolic computation. *Commun. ACM*, 65(7): 64–72.

Claessen, K.; and Sörensson, N. 2003. New Techniques that Improve MACE-style Finite Model Finding. In *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*.

Codish, M.; Miller, A.; Prosser, P.; and Stuckey, P. J. 2018. Constraints for symmetry breaking in graph representation. *Constraints*, 24(1): 1–24.

Crawford, J. M.; Ginsberg, M. L.; Luks, E. M.; and Roy, A. 1996. Symmetry-Breaking Predicates for Search Problems. In Aiello, L. C.; Doyle, J.; and Shapiro, S. C., eds., *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, 148–159.

de Rezende, S. F.; Nordström, J.; Risse, K.; and Sokolov, D. 2020. Exponential Resolution Lower Bounds for Weak Pigeonhole Principle and Perfect Matching Formulas over Sparse Graphs. In Saraf, S., ed., *35th Computational Complexity Conference, CCC*, volume 169 of *LIPIcs*, 28:1–28:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Distler, A.; Jefferson, C.; Kelsey, T.; and Kotthoff, L. 2012. The Semigroups of Order 10. In Milano, M., ed., *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, 883–899. Springer.

Distler, A.; and Mitchell, J. 2022. Smallsemi - A library of small semigroups Version 0.6.13. https://www.gap-system.org/Packages/smallsemi.html. GAP package.

Dutertre, B. 2020. An Empirical Evaluation of SAT Solvers on Bit-vector Problems. In Bobot, F.; and Weber, T., eds., *Proceedings of the 18th International Workshop on Satisfiability Modulo Theories co-located with the 10th International Joint Conference on Automated Reasoning*, volume 2854 of *CEUR Workshop Proceedings*, 15–25. CEUR-WS.org.

Eén, N.; and Sörensson, N. 2003. An Extensible SAT-solver. In Giunchiglia, E.; and Tacchella, A., eds., *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT*, volume 2919 of *Lecture Notes in Computer Science*, 502–518. Springer.

GAP4. 2021. *GAP – Groups, Algorithms, and Programming, Version 4.11.1*. The GAP Group.

Haken, A. 1985. The intractability of resolution. *Theoretical Computer Science*, 39: 297–308.

Heule, M. J. H. 2018. Schur Number Five. In McIlraith, S. A.; and Weinberger, K. Q., eds., *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*, 6598–6606. AAAI Press.

Heule, M. J. H. 2019. Optimal Symmetry Breaking for Graph Problems. *Math. Comput. Sci.*, 13(4): 533–548.

Itzhakov, A.; and Codish, M. 2020. Incremental Symmetry Breaking Constraints for Graph Search Problems. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI*, 1536–1543. AAAI Press.

Janota, M.; Morgado, A.; and Vojtechovský, P. 2023. Computing generating sets of minimal size in finite algebras. *J. Symb. Comput.*, 119: 50–63.

Jipsen, P. 2016. Mathematical Structures. https://math.chapman.edu/~jipsen/uajs/. Accessed: 2024-02-13.

Khan, M. A. 2020. Efficient Enumeration of Higher Order Algebraic Structures. *IEEE Access*, 8: 41309–41324.

Kirchweger, M.; and Szeider, S. 2021. SAT Modulo Symmetries for Graph Generation. In Michel, L. D., ed., *27th International Conference on Principles and Practice of Constraint Programming, CP*, volume 210 of *LIPIcs*, 34:1–34:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Knuth, D. E. 2015. *The Art of Computer Programming: Satisfiability, Volume 4, Fascicle 6*. Addison-Wesley Professional. ISBN 9780134394572.

Lutz, F. 2008. Enumeration and Random Realization of Triangulated Surfaces. In Bobenko, A.; Sullivan, J.; Schröder, P.; and Ziegler, G., eds., *Discrete Differential Geometry*, volume 38 of *Oberwolfach Seminars*. Birkhäuser Basel.

Lutz, F. H. 2009. Isomorphism-free lexicographic enumeration of triangulated surfaces and 3-manifolds. *European Journal of Combinatorics*, 30(8): 1965–1979.

Marker, D. 2002. *Model Theory: An Introduction*. New York, NY: Springer.

Marques-Silva, J.; Argelich, J.; Graça, A.; and Lynce, I. 2011. Boolean lexicographic optimization: algorithms & applications. *Ann. Math. Artif. Intell.*, 62(3-4): 317–343.

McCune, W. 1994. A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical report, Argonne National Laboratory.

McKay, B. D.; and Piperno, A. 2014. Practical graph isomorphism, II. *J. Symb. Comput.*, 60: 94–112.

Nadel, A.; and Ryvchin, V. 2016. Bit-Vector Optimization. In Chechik, M.; and Raskin, J., eds., *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS*, volume 9636 of *Lecture Notes in Computer Science*, 851–867. Springer.

Nagy, G.; and Vojtěchovský, P. 2018. LOOPS, Computing with quasigroups and loops in GAP, Version 3.4.1. https://gap-packages.github.io/loops/. Refereed GAP package.

Peter, V. B.; Rossi, F.; Van Beek, P.; and Walsh, T., eds. 2014. *Handbook of constraint programming*. Foundations of Artificial Intelligence. Elsevier Science & Technology.

Petkovska, A.; Mishchenko, A.; Soeken, M.; Micheli, G. D.; Brayton, R. K.; and Ienne, P. 2016. Fast generation of lexicographic satisfiable assignments: enabling canonicity in SAT-based applications. In Liu, F., ed., *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*, 4. ACM.

Reynolds, A.; Tinelli, C.; Goel, A.; and Krstić, S. 2013a. Finite Model Finding in SMT. In *Computer Aided Verification - 25th International Conference, CAV*, 640–655.

Reynolds, A.; Tinelli, C.; Goel, A.; Krstić, S.; Deters, M.; and Barrett, C. 2013b. Quantifier Instantiation Techniques for Finite Model Finding in SMT. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, 377–391.

Roussel, O.; and Manquinho, V. M. 2021. Pseudo-Boolean and Cardinality Constraints. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 1087–1129. IOS Press.

Sakallah, K. A. 2021. Symmetry and Satisfiability. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 509–570. IOS Press.

Schneider, N.; Sayle, R. A.; and Landrum, G. A. 2015. Get Your Atoms in Order-An Open-Source Implementation of a Novel and Robust Molecular Canonicalization Algorithm. *Journal of Chemical Information and Modeling*, 55(10): 2111–2120.

Torlak, E.; and Jackson, D. 2007. Kodkod: A Relational Model Finder. In Grumberg, O.; and Huth, M., eds., *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS*, volume 4424 of *Lecture Notes in Computer Science*, 632–647. Springer.

Trentin, P. 2019. *Optimization Modulo Theories with OptiMathSAT*. Ph.D. thesis, University of Trento, Italy.

Urban, J.; Sutcliffe, G.; Pudlák, P.; and Vyskočil, J. 2008. MaLARea SG1 – Machine Learner for Automated Reasoning with Semantic Guidance. In Armando, A.; Baumgartner, P.; and Dowek, G., eds., *International Joint Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Computer Science*, 441–456. Springer. ISBN 978-3-540-71069-1.

Walsh, T. 2012. Symmetry Breaking Constraints: Recent Results. In Hoffmann, J.; and Selman, B., eds., *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press.

Weininger, D.; Weininger, A.; and Weininger, J. L. 1989. SMILES. 2. Algorithm for generation of unique SMILES notation. *Journal of Chemical Information and Modeling*, 29(2): 97–101.

Zemljachenko, V. N.; Korneenko, N. M.; and Tyshkevich, R. I. 1982. Problema Izomorfizma Grafov. *Zapiski nauchnyh seminarov POMI*, 118(0): 83–158.

Zhang, J. 1996. Constructing finite algebras with FALCON. *Journal of Automated Reasoning*, 17: 1–22.

Zhang, J.; and Zhang, H. 1995. SEM: a System for Enumerating Models. In *IJCAI*, 298–303.