Filtering Isomorphic Models by Invariants

João Araújo
NOVA University Lisbon, Portugal

Choiwah Chow
Universidade Aberta, Lisbon, Portugal

Mikoláš Janota
Czech Technical University in Prague, Czech Republic

Abstract

The enumeration of finite models of first order logic formulas is an indispensable tool in computational algebra. The task is hindered by the existence of isomorphic models, which are of no use to mathematicians and therefore are typically filtered out a posteriori. This paper proposes a divide-and-conquer approach to speed up and parallelize this process. We design a series of invariant properties that enable us to partition existing models into mutually non-isomorphic blocks, which are then tackled separately. The presented approach is integrated into the popular tool Mace4, where it shows tremendous speed-ups for a variety of algebraic structures.

2012 ACM Subject Classification Computing methodologies; Theory of computation → Constraint and logic programming

Keywords and phrases finite model enumeration, isomorphism, invariants, Mace4

Digital Object Identifier 10.4230/LIPIcs.CP.2021.4

Category Short Paper

Funding *João Araújo*: Fundação para a Ciência e a Tecnologia, through the projects UIDB/00297-/2020 (CMA), PTDC/MAT-PUR/31174/2017, UIDB/04621/2020 and UIDP/04621/2020.
Mikoláš Janota: The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902. This scientific 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.

1 Introduction

There are many types of relational algebras (groups, semigroups, quasigroups, fields, rings, MV-algebras, lattices, etc.) using operations and relations of many arities, but the overwhelming majority of the most popular only use operations of arity at most 2; in the words of two famous algebraists, *It is a curious fact that the algebras that have been most extensively studied in conventional (albeit modern!) algebra do not have fundamental operations of arity greater than two.* (See page 26 of [4])

To study and get intuition on them, mathematicians resort to libraries of all order n models of the algebra they are interested in (for small values of n). These libraries allow experiments such as testing and/or forming conjectures etc., to gain insights. Therefore, it comes as no surprise that GAP [8], the most popular computational algebra system, has many such libraries. For groups it has the list of almost all small groups up to order a few thousands and the list of all primitive groups up to degree a few thousands, among others; for semigroups it has the list of all small models up to order 8 [6]; for quasigroups up to order 6 [16]; there is also a library of Lie algebras and many others. These libraries are so important that the search for them has a long history in mathematics predating for many years the use of computers. For example, the search for libraries of degree n primitive groups started long ago: Jordan (1872) for $n \leq 7$; Burnside (1897) for $n \leq 8$; Manning (1906-1929) for $n \leq 15$; Sims (1970) for $n \leq 20$, Pogorelov (1980) for $n \leq 50$; Dixon and Mortimer (1988) for $n \leq 1000$. (See Appendix B of [7]; and for more recent results in OEIS [17]).

Many more such libraries are needed. For example, SMALLSEMI [6] has the list of semigroups up to order 8 (there are too many semigroups of order 9 to be storable), but if we impose extra properties on the semigroup (such as being inverse, a band, regular, or Clifford, etc. – there are tens of classes of semigroups –) their numbers decrease and hence libraries of models of higher orders could be produced and stored.

Many of these algebras can be defined in first order logic (FOL) and there are tools to allow mathematicians to encode their algebras and produce a meaningful library. The problem is that usually the tools that can be easily learned and used by mathematicians generate too many isomorphic models, thus wasting time generating redundant models and then wasting more time to get rid of them. For example, Mace4 [13], a very popular finite model enumerator among mathematicians due to its very intuitive and user-friendly language, would produce 28,947,734 inverse semigroups of order 8 when given the following simple first-order formulas as input [1] (with binary operation $*$ and unary operation $'$).

$$\begin{array}{lll} (x * y) * z = x * (y * z). & (x * x') * x = x. & 0 * 0 = 0. \\ ((x * x') * y) * y' = ((y * y') * x) * x'. & x'' = x. & \end{array}$$

During the search, the number of output models in this example is already greatly reduced by the Least Number Heuristic (LNH) and the special symmetry breaking input clause $0 * 0 = 0$. Out of the almost 29 millions output models, only 4,637 ($\approx 0.016\%$) are pairwise non-isomorphic. The proportion of non-isomorphic models in the outputs tends to get smaller very fast as the order of the algebraic structure goes higher.

Redundant models may either be eliminated during search or filtered out afterwards. Guaranteeing that search never produces isomorphic models is a hard problem and is rarely seen in modern solvers. This paper therefore tackles the second problem, i.e., the removal of redundant models from an already enumerated set.

In our context, the complexity of checking whether two models are isomorphic is only part of the problem. Another source of complexity is the large number of models that need to be checked. If all pairs of models are checked, the performance degrades rapidly as the total number of models increases (see Section 5).

To tackle this problem, we explored many different strategies eventually concluding that the best one is to assign to every generated model a vector that is invariant under isomorphism. This allows us to partition the output with all the isomorphic models living inside the same block (or part). This splits the problem into substantially smaller sub-problems. Moreover, processing inside each block can easily be done in parallel as models across blocks cannot be isomorphic. This is an important facet of the approach since modern-day computers are more often than not equipped with multiple cores.

What made this project take off was the identification of a large number of general algebra properties invariant under isomorphism coupled with experiments to identify a small subset of these properties without losing discriminating power. This approach will help mathematicians on two levels: first, it provides them with a tool on their desktop that quickly produces a library for the algebra they are working with; second, the tool may be run on a cluster of computers to pre-compute libraries for the most famous classes of algebras, and add them to GAP [8] or a similar system.

Our contributions to the area of isomorphic model elimination are (see Section 3):

- Devise an invariant-based algorithm that can be applied to algebras defined in FOL and containing at least one binary operation.
- Design a small set of invariant properties that in practice have high discriminating power, and yet are inexpensive to compute.
- Use a hash-map to store models partitioned by the invariant-based algorithm to allow fast storage and retrieval of models in the same block.

We apply the proposed partitioning technique to Mace4's isomorphic model filtering programs, and observe orders of magnitude speed-up in its isomorphic model elimination step (see Section 4).

2 Mathematical Background

Algebra is a pair (A, Ω), where A is a set and Ω is a set of operations, that is, functions $f : A^n \to A$ (in this case f is said to be an operation of arity n). Let $A = (D, *_A)$ and $B = (D, *_B)$ be two algebras, each with one binary operation on a finite domain (or universe) D. An isomorphism of these two algebras is a bijective function $f : A \to B$ such that $f(a *_A b) = f(a) *_B f(b)$, for all $a, b \in A$. Two models are said to be isomorphic if there exists an isomorphism between them. The relation A *is isomorphic to* B is clearly an equivalence relation and hence induces a partition of the algebras considered. Only one representative algebra in each block is needed.

The definition of isomorphism can easily be extended to cover algebras with multiple binary operations. Formally, suppose A and B are algebras of type $(2^m, 1^n)$, where m, n are non-negative integers; then we can assume that the binary operations are $(*_1, \dots, *_m)$ and the unary operations are $(g_1, \dots, g_n)$. An isomorphism between them is a bijection $f : A \to B$ such that $f(a *_i b) = f(a) *_i f(b)$, for all $a, b \in D$ and every binary operation $*_i$, and for any unary operation g_i, we have $f(g_i(a)) = g_i(f(a))$, for all $a \in D$.

3 Invariant-based Algorithm

Let A and B be two algebras and $f : A \to B$ an isomorphism between them; in addition, suppose $e^2 = e \in A$ is an idempotent. Then $f(ee) = f(e)$ implies that $f(e)f(e) = f(e)$, that is, $f(e)$ under an isomorphism is also an idempotent. As isomorphisms map idempotents onto idempotents, it follows that the number of idempotents in A must be smaller or equal to the number of idempotents on B. Since the inverse of an isomorphism is an isomorphism, A and B must have the same number of idempotents. We call these properties that are preserved by isomorphisms (such as the *number of idempotents*) *invariant properties* or *invariants* for short. These invariant properties are the basis of our proposed algorithm.

Guided by fundamental concepts heavily appearing in different parts of mathematics, we design 10 invariant properties that collectively have high discriminating powers, and yet are inexpensive to compute. For a binary operation in a model with finite domain D, we compute the invariant properties for each domain element x as:

1. The smallest integer n such that $x^n = x^k, n > k >= 1$ where we define x^n to be $(\dots(x * x) * x) * x)\dots$ for n x's (*periodicity*).
2. The number of $y \in D$ such that $x = (xy)x$ (*number of inverses*).
3. The number of distinct xy for all $y \in D$ (*size of right ideal*).
4. The number of distinct yx for all $y \in D$ (*size of left ideal*).
5. 1 if $xx = x$, 0 otherwise (*idempotency*).

6. The number of $y \in D$ such that $x(yy) = (yy)x$ (*number of commuting squares*).
7. The number of $y \in D$ such that $x = yy$ (*number of square roots*).
8. The number of $y \in D$ such that $x(xy) = (xx)y$ (*number of square associatizers*).
9. The number of pairs of $y, z \in D$ such that $zy = yz = x$ (*number of symmetries*).
10. The number of $y \in D$ such that there exists pairs of $s, t \in D$ where $x = st$ and $y = ts$ (*number of conjugates*).

Invariant 5 is the idempotent property of the domain element and is preserved by isomorphisms as discussed before. The correctness of invariants in general hinges on the following lemma (folklore). Let F be a FOL formula on the signature of the algebra and M and M' two isomorphic models. It holds that the sets S and S' defined by F in M and M', respectively, are of the same cardinality. This is because the isomorphism induces a bijection between the two sets (*cf.* Theorem 1.1.10 in [12]). In other words, invariants based on solution counting are guaranteed to be correct.

We call the ordered list of invariant properties so calculated the invariant vector of that domain element. Each model with n domain elements will be associated with n invariant vectors. Isomorphic models must have the same set of invariant vectors.

To facilitate comparisons of invariant vectors, we sort the invariant vectors by the lexicographical order of their elements (see the example below for more explanations). It follows that models isomorphic to each other must have the same sorted invariant vectors. If the model has multiple binary operations, then invariant vectors are calculated for each of the binary operations, and all the invariant vectors of the same domain element are concatenated to form a combo invariant vector for that domain element. The combo invariant vectors will then be sorted to yield the final ordered list of invariants.

Often we are not only to compare 2 models for isomorphism, but to extract all non-isomorphic models from a list of models. In that case, we set up a hash map to store the blocks of the models. We use the invariant vectors for each model to send the model quickly to the block (in the hash map) to which it belongs. That is, the keys in this hash map are the invariant vectors, and the values are the blocks of the models. After all models are hashed into the hash map, the blocks stored in the hash map can be processed separately, and possibly in parallel, to extract one representative model from each isomorphism class.

Note that our invariant-based algorithm does not compare models for isomorphism. It only cuts down the size of the problem to improve the speed of existing isomorphism filters such as Mace4's *isofilter*.

As an example to show how invariant vectors are constructed and used, suppose we want to find all non-isomorphic models in a list of 3 quasigroups, A, B, and C, of order 4. Suppose further that their domain is $D = \{0, 1, 2, 3\}$ and their operation tables are given in Table 1.

Table 1 Operation tables of Quasigroups A, B and C.

Model A

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

Model B

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

Model C

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

The 10 invariant properties can easily be calculated for each of the domain elements of these models. Note that while the invariant vector for each domain element is calculated separately, it is not important exactly which domain element gives a particular invariant vector. It is the set of invariant vectors as a whole that matters.

Invariant vectors of Model A

0:	2	1	4	4	1	4	2	4	4	1
1:	3	1	4	4	0	4	2	4	4	1
2:	5	1	4	4	0	4	0	4	4	1
3:	5	1	4	4	0	4	0	4	4	1

Invariant vectors of Model B

0:	2	1	4	4	1	4	2	4	4	1
2:	3	1	4	4	0	4	2	4	4	1
1:	5	1	4	4	0	4	0	4	4	1
3:	5	1	4	4	0	4	0	4	4	1

Invariant vectors of Model C

0:	2	1	4	4	1	4	4	4	4	1
1:	3	1	4	4	0	4	0	4	4	1
2:	3	1	4	4	0	4	0	4	4	1
3:	3	1	4	4	0	4	0	4	4	1

Figure 1 Lexicographically sorted invariant vectors with discerning properties highlighted.

Next we sort the invariant vectors of each model by their elements lexicographically. Invariant vectors of models A and C need no change as they are already in the desired sort order. Invariant vectors of model B will be in sort order by interchanging the invariant vectors of elements 1 and 2, which are the second and third row. The final invariant vectors are shown in the Figure 1. Note that the first column in the tables is the domain element, and the next 10 columns are its invariant properties.

The highlighted numbers in the figure are the discerning invariant properties in the example. All other invariant properties are the same from domain element to domain element. For example, Invariant properties 3, *size of right ideal*, and 4, *size of left ideal*, always equal to the size of the domain D because the operation table of a quasigroup is a Latin square. This highlights the need for multiple invariant properties targeting different areas of algebraic structures to increase their collective discriminating powers. In fact, our algorithm depends more on the orthogonality of the invariants than on the splitting power of any one individual invariant. See Table 2 for the top invariants in different algebras.

It should be easy to see that models A and B have the same sorted invariant vectors, and thus are possibly isomorphic to each other. They are indeed isomorphic to each other because applying the permutation $(1, 2)$ to model B will give model A. However, invariant vectors alone cannot prove that they are isomorphic models. It is also easy to see that the invariant vectors of model C are different from those of the other 2 models, and from this fact alone, we can conclude that model C is not isomorphic to any of A and B.

Finally, for ease of comparison and hashing, we concatenate the sorted invariant vectors into a single string. The string representation of the invariant vectors for the models are:

A, B: 2,1,4,4,1,4,2,4,4,1,3,1,4,4,0,4,2,4,4,1,5,1,4,4,0,4,0,4,4,1,5,1,4,4,0,4,0,4,4,1
C: 2,1,4,4,1,4,4,4,4,1,3,1,4,4,0,4,0,4,4,1,3,1,4,4,0,4,0,4,4,1,3,1,4,4,0,4,0,4,4,1

Since we are to extract all non-isomorphic models from this list of models, we use the string representations of the invariant vectors as the keys for the hash map. Both models A and B will therefore go to the same block in the hash map, but C will go to a different block. Now that all 3 models are deposited in their blocks in the hash map, each block can be processed separately in parallel as we only need to compare models in the same block for isomorphism. This step can be performed by many existing programs such as Mace4's isomorphism filters (see Section 4).

Finally, if the models have multiple binary operations, we compute the unsorted invariant vectors for each binary operation as described above, then concatenate the invariant vectors of the same domain element into one combo invariant vector, sort these combo invariant vectors in lexicographical order, and finally concatenate the sorted invariant vectors into their string format.

It is important to note that the hash map in our algorithm obviates the need to compare invariant vectors among the models during the partitioning process. If we do pairwise comparison of models by their invariant vectors in any step, we would end up with a $O(n^2)$ worst-case scenario.

4 Experimental Results

We have implemented an invariant-based pre-processor to the Mace4's isomorphic models filters. We run the experiments on a 6-core Intel® Core™ i7-9850H CPU computer. We shall show results of tests on 3 algebraic structures, namely, quasigroup, inverse semigroup, and quandle [1]. They are chosen because of their importance in the mathematical world. Quasigroup is the most prominent non-associative algebra, inverse semigroup is probably the most studied associative algebra with a unary operation, and quandles is probably most important algebra with 2 binary operations.

The results show that when the size of the output models is more than just a few hundreds of thousands, the invariant vectors often give an order or two magnitudes of improvements in the speed of the isomorphism elimination process even without running them in parallel. A very desirable feature of our algorithm is that the improvement increases dramatically as the size of the problem grows. Furthermore, Mace4's *isofilter2* is not able to handle input size beyond a few million quasigroups of order 6 (see Table 2), but our invariant-based algorithm can partition the models into smaller blocks of sizes within Mace4's limits.

Table 2 Isomorphism Eliminations.

			Time (s)	
	Order	# of Mace4 Outputs	With Invariants	Without Invariants
Quasigroups	5	10,944	1	1
	6	11,543,040	1,182	N/A
Inverse Semigroups	5	2,151	<1	<1
	6	38,828	3	2
	7	929,923	73	81
	8	28,947,734	2,873	150,703
Quandles	6	1,833	2	1
	7	22,104	6	374
	8	359,859	450	267,463

We show the results of the non-parallel runs to demonstrate the improvements due solely to the invariant vectors. The performance can be improved further if the blocks are processed in parallel. For example, the processing time for the biggest block for quandles of order 8 is only 20 seconds, so if we have enough processors to process all the blocks in parallel, then the processing time can theoretically be cut down close to $24 + 19.937 \approx 44$ seconds from 450 seconds, more than 90% reduction (see Table 3).

Table 3 Isomorphism Eliminations in Parallel.

			Time (s)	
	Order	#Blocks	Generating Invariants	Processing Biggest Block
Quasigroups	6	1,129,129	265	0.0106132
Inverse Semigroups	8	4,582	1,031	2.807
Quandles	8	1,143	24	19.937

One reason for the dramatic improvement in the run-time by our invariant-based algorithm is that the invariant vectors chosen have great discriminating power as shown by the fact that the average number of non-isomorphic models per block is very close to 1 (see Table 4). The top 4 contributing invariants for the highest order of each class are also listed in Table 4.

Table 4 Discriminating Power of Invariant Vectors.

			Non-isomorphic Models		
	Order	#Blocks	Total	Avg per Block	Top 4 Invariants
Quasigroups	5	1,402	1,411	1.01	
	6	1,129,129	1,130,531	1.00	6, 1, 8, 10
Inverse Semigroups	5	52	52	1.00	
	6	208	208	1.00	
	7	908	911	1.02	
	8	4,582	4,637	1.01	9, 3, 2, 1
Quandles	6	66	73	1.11	
	7	250	298	1.19	
	8	1,143	1,581	1.38	8, 3, 6, 10

5 Related Work

The proposed approach falls into the class of *divide-and-conquer* algorithms; most notably Heule et al. [10] recently applied the *cube-and-conquer* approach [9] to solve the Boolean Pythagorean triples problem.

There are a large number of techniques to break symmetries during the search phase [5], such as the Least Number Heuristic (LNH) [18] and the eXtended LNH (XLNH) [2]. The LNH, for example, is a very popular dynamic symmetry breaker implemented in Mace4, FALCON [18], and SEM [19], etc., to help reduce the number of isomorphic models. However, these techniques do not guarantee isomorph-freeness. Systems that try to generate isomorph-free models, such as SEMK [3, 14] and SEMD [11], are either yet to be complete, or are better off allowing some isomorphic models in the outputs for some problem sets. Thus, post-processing tools such as our invariant-based algorithm have an important role in isomorphism elimination as total elimination of isomorphism in the model search phase may not always be the best option.

Invariants are widely used under different guises in many branches of mathematics. For example, in graph theory, node invariants can be used to help detect isomorphic graphs [15]. Interestingly, similar ideas can be seen in Mace4's isomorphism filters. Indeed, Mace4's *isofilter* uses the numbers of occurrences of domain elements in the operation tables as the lone invariant that serves 2 purposes: First is to do quick checks for non-isomorphism, as models having different occurrences of domain elements cannot be isomorphic. Second is to guide the construction of isomorphic functions between potential isomorphic models, as domain elements can only map to domain elements having the same occurrences in the operation tables. This reduces the number of permutations to try in the search of isomorphic functions. However, the lone invariant in *isofilter* would fail miserably if the models are quasigroups for which each domain element would appear the same of times in the operation table. To mitigate this problem, Mace4 provides another isomorphism filter, *isofilter2*, which transforms the models to their canonical forms based on the same algorithm [14] given by McKay as mentioned above in SEMK. Compared to *isofilter*, *isofilter2* performs much better for quasigroups, but worse on other algebraic structures such as semigroups due to its high overheads in computing canonical forms. Nevertheless, both filters compare every model against the list of non-isomorphic models found so far, and hence their performances degrade rapidly as the number of models increases. Therefore, both filters benefit immensely from the reduced number of models in the blocks created by our invariant-based algorithm.

The *loops* package [16] in GAP [8] uses invariant vectors of 9 invariants in many of its isomorphism-related functions. Like Mace4's *isofilter*, it uses invariant vectors to check for non-isomorphism, and to help guide the construction of isomorphic function between models using sophisticated algorithms that take advantage of other GAP functions. Their invariant vectors work on only one operation table, and exploit heavily specific properties of quasigroups and loops, which may be ineffective in other kinds of algebras. Our invariant-based algorithm targets different aspects of all algebraic structures including quasigroups, semigroups, and more. It also works with multiple binary operations, and does not rely on any built-in functionality of GAP. Moreover, given a list of models to find non-isomorphic models, the *loops* package would compare the invariant vector of every model against those of the list of all non-isomorphic models found so far to get the list of potential isomorphic models. Our hash map-based organization of models eliminates the need to compare invariant vectors repeatedly because all models having the same invariant vectors are already grouped into the same block in the hash map.

6 Future Work and Conclusions

Currently, we only compute invariants based on binary operations, which are by far the most prevalent operations in algebraic structures [4]. However, unary operations are also quite common, and may be even less expensive to manipulate. The discriminating power of the invariant vectors of the model can be enhanced with the addition of invariant vectors based on unary operations, and will be part of our future focus.

The results of our research open a whole new line of research into using invariant properties to eliminate isomorphism in finite model enumeration:

- Identify more invariant properties and the cases for which each of them may be useful.
- Allow dynamic, and preferably automatic, selection of invariant properties to use in any given algebraic structure because different invariants work best for different algebraic structures (see example in Section 3, and also Table 4), so we need to allow dynamic, and preferably automatic, selection of invariant properties.
- Find the best sets of invariant properties to use for various sizes and types of models. A larger set of algebras (usually of higher orders) may need more invariants in the invariant vectors to provide enough discriminating power to separate the models into smaller blocks, but a smaller set of algebras may incur too much overhead in computing the invariant vectors with many invariant properties.

We observe that the invariant-based algorithm is efficient, scalable, and parallelizable. It is also compatible with most, if not all, existing finite model enumerators. The focus of future research will be on finding more good invariant properties, in binary and in unary operations, to be used as partitioning keys, and on adding the capability of dynamic and automatic selection of invariant properties to use.

References

1 João Araújo, David Matos, and João Ramires. Axiomatic library finder. URL: `https://axiomaticlibraryfinder.pythonanywhere.com/definitions` [cited 15.05.2021].

2 Gilles Audemard and Laurent Henocque. The extended least number heuristic. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning*, pages 427–442, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

3 Thierry Boy de la Tour and Prakash Countcham. An isomorph-free sem-like enumeration of models. *Electronic Notes in Theoretical Computer Science*, 125(2):91–113, 2005. Proceedings of the 5th International Workshop on Strategies in Automated Deduction (Strategies 2004). `doi:10.1016/j.entcs.2005.01.003`.

4 Stanley Burris and Hanamantagouda P. Sankappanavar. *A course in universal algebra*, volume 78 of *Graduate texts in mathematics*. Springer, 1981.

5 James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 148–159. Morgan Kaufmann, 1996.

6 A. Distler and J. Mitchell. Smallsemi, a library of small semigroups in GAP, Version 0.6.12, 2019. GAP package. URL: `https://gap-packages.github.io/smallsemi/`.

7 John D. Dixon and Brian Mortimer. *Permutation Groups*. Springer, 1996.

8 The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.11.1*, 2021. URL: `https://www.gap-system.org`.

9 Marijn Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, HVC, Revised Selected Papers*, volume 7261, pages 50–65. Springer, 2011. `doi:10.1007/978-3-642-34188-5_8`.

10 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the BooleanPythagorean triples problem via cube-and-conquer. In *Theory and Applications of Satisfiability Testing (SAT)*, 2016. `doi:10.1007/978-3-319-40970-2_15`.

11 Xiangxue Jia and Jian Zhang. A powerful technique to eliminate isomorphism in finite model search. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 318–331, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

12 David Marker. *Model Theory: An Introduction*. Springer, 2002.

13 William McCune. Mace4 reference manual and guide. Technical Report Technical Memorandum No. 264, Argonne National Laboratory, Argonne, IL, August 2003. URL: `https://www.cs.unm.edu/~mccune/prover9/mace4.pdf`.

14 Brendan D McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26(2):306–324, 1998. `doi:10.1006/jagm.1997.0898`.

15 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. `doi:10.1016/j.jsc.2013.09.003`.

16 Gábor Nagy and Petr Vojtěchovský. LOOPS, computing with quasigroups and loops in GAP, Version 3.4.1, November 2018. Refereed GAP package. URL: `https://gap-packages.github.io/loops/`.

17 Neil J. A. Sloane and The OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2020. URL: `http://oeis.org/?language=english`.

18 Jian Zhang. Constructing finite algebras with FALCON. *Journal of Automated Reasoning*, 17:1–22, August 1996. `doi:10.1007/BF00247667`.

19 Jian Zhang and Hantao Zhang. SEM: a system for enumerating models. In *IJCAI*, pages 298–303, 1995. URL: `http://ijcai.org/Proceedings/95-1/Papers/039.pdf`.