

# Automatic Construction and Verification of Isotopy Invariants

Volker Sorge ([v.sorge@cs.bham.ac.uk](mailto:v.sorge@cs.bham.ac.uk))  
*School of Computer Science, University of Birmingham, UK*  
<http://www.cs.bham.ac.uk/~vxs>

Andreas Meier ([ameier72@web.de](mailto:ameier72@web.de))  
*DFKI GmbH, Saarbrücken, Germany*  
<http://www.ags.uni-sb.de/~ameier>

Roy McCasland\* ([rmccasla@inf.ed.ac.uk](mailto:rmccasla@inf.ed.ac.uk))  
*School of Informatics, University of Edinburgh, UK*  
<http://www.inf.ed.ac.uk/~rmccasla>

Simon Colton ([sgc@doc.ic.ac.uk](mailto:sgc@doc.ic.ac.uk))  
*Department of Computing, Imperial College London, UK*  
<http://www.doc.ic.ac.uk/~sgc>

**Abstract.** We extend our previous study of the automatic construction of isomorphic classification theorems for algebraic domains by considering the *isotopy* equivalence relation. Isotopism is an important generalisation of isomorphism, and is studied by mathematicians in domains such as loop theory. This extension was not straightforward, and we had to solve two major technical problems, namely generating and verifying isotopy invariants. Concentrating on the domain of loop theory, we have developed three novel techniques for generating isotopic invariants, by using the notion of universal identities and by using constructions based on sub-blocks. In addition, given the complexity of the theorems which verify that a conjunction of the invariants form an isotopy class, we have developed ways of simplifying the problem of proving these theorems. Our techniques employ an interplay of computer algebra, model generation, theorem proving and satisfiability solving methods. To demonstrate the power of the approach, we generate isotopic classification theorems for loops of size 6 and 7, which extends the previously known enumeration results. This result was previously beyond the capabilities of automated reasoning techniques.

**Keywords:** Automated Mathematics, Automated Theorem Proving, SAT Solving, Computer Algebra, Model Generation, Isotopy, Invariant generation, Classification Theorems.

## 1. Introduction

The classification of abstract algebraic objects such as groups and rings has been a major driving force in pure mathematics. For the majority of algebraic domains, however, full classifications have been elusive (with

---

\* The author's work was supported by EPSRC MathFIT grant GR/S31099.



notable exceptions being Abelian groups [12], finite simple groups [2] and Abelian quasigroups [24]). This is partially due to the sheer number of classes in domains such as quasigroups, where the axioms are not particularly restrictive. Due to this volume of classes, automatic techniques have much potential to add to mathematical knowledge. In section 2, we describe some projects in which the use of automated reasoning techniques has led to additions to mathematical knowledge. In particular, we describe some quantitative approaches to classification related tasks, such as counting the number of isomorphism classes of a particular size for particular algebraic domains.

We are interested in automatically generating *qualitative* results which describe not just how many equivalence classes there are, but how each class differs from the others. In particular, we aim to discover and prove theorems which provide an algebraic description of a property which uniquely describes each equivalence class for particular orders of particular algebraic domains such as loops, groups and quasigroups. To this end, we have developed a bootstrapping algorithm for the construction of such classification theorems in finite algebraic domains. While it is largely generic, the power of this algorithm relies upon the automatic discovery of algebraic invariants which are able to discriminate between members of different classes defined by the equivalence relation under consideration. When considering isomorphism equivalences, we were able to employ machine learning techniques to generate the invariants, and this led to novel theorems which achieve classifications up to isomorphism of quasigroups and loops of small order [7]. We provide a brief overview of the bootstrapping algorithm in section 3.

We present here the application of the bootstrapping algorithm to the production of classification theorems up to *isotopism*, an equivalence relation of some importance in certain domains, in particular algebraic loop theory. Unfortunately, the machine learning approach did not suffice for this application, as finding isotopy invariants is a much more complex task. Hence, concentrating on the domain of loop theory, we have developed new methods for generating isotopy invariants, as described in section 4. Using results from the literature, we show how universal identities can be used to generate invariants via an interplay of model generation and theorem proving. In addition, we present two new sets of invariants derived by systematically examining sub-blocks<sup>1</sup> of loops, and we describe their construction, which uses computer algebra techniques. As described in section 5, it has also been a challenge to automatically verify that a conjunction of invariant properties defines an isotopy class (another important

---

<sup>1</sup> Note that in [27], we use the term substructure, rather than sub-block.

aspect of the bootstrapping algorithm). For this reason, we have developed methods for simplifying the problems with computer algebra techniques, before providing proof via a satisfiability solver. As a by-product, these methods significantly simplify the task of generating non-isotopic models.

Having solved the problems of generating and verifying isotopic invariants, we have employed the bootstrapping algorithm to generate new results. In particular, in section 6, we present the details of an isotopic classification theorem for loops of order 6, which extends the known enumeration theorem by providing a full set of classifying properties, and we summarise a similar result for loops of order 7. In addition to adding to mathematical knowledge, we also see this project as a case study in how to combine automated reasoning systems. Note that our emphasis is on combining existing programs which perform different aspects of mathematical reasoning such as proving theorems (e.g., by ATP systems), finding counterexamples to non-theorems (e.g., by model generators), inducing properties from examples (e.g., by machine learning systems), solving equations (e.g., by SAT solvers) and simplifying expressions (e.g., by computer algebra systems). This is a more practical approach to combining reasoning systems than, for example, the more theoretical approach taken in combining decision procedures [14], where the emphasis is on improving reasoning power by multiplying approaches for performing the same form of reasoning. In total, we have used and experimented with various combinations of 15 third party systems in our efforts to achieve classification theorems. In section 7, we conclude by arguing that the way forward for automating discovery tasks in mathematics is to continue to combine systems so that the whole is more than the sum of the parts, as advocated by Bundy in [4].

## 2. Related Work

Looking at the context of efforts to classify algebraic objects, we note that our qualitative approach is related to a number of other quantitative approaches which have added to this effort. In particular, enumeration techniques have been used to count the instances of certain finite algebras [18], and model generating and constraint solving techniques have been used to solve open existence problems. For instance, the FINDER program [25] has been used to solve many quasigroup existence problems. In [26], Slaney used FINDER along with the DDPP program (a Davis-Putnam implementation) to solve numerous quasigroup existence problems. For example, they found an idempotent QG3

quasigroup (such that  $\forall a, b (a * b) * (b * a) = a$ ) of size 12, settling that existence problem. They had similar results for QG4 quasigroups, and solved many other existence questions, both by finding counterexamples and by exhausting the search to show that no quasigroups of given types and sizes exist.

Looking instead at our project in the context of automatic discovery of mathematical theorems, there have been a number of related projects where automated reasoning techniques have made discoveries. In particular, McCune et al. have found different axiom systems for well known algebras, such as groups and loops [15] [16]. In many cases, it has been possible to reduce the axioms to a single axiom. For instance, group theory can be expressed in terms of the multiplication and inverse operators in a single axiom:

$$\forall x, y, z, u \in G, (x(y(((zz')(uy)')x)'))' = u,$$

where  $a'$  indicates the inverse of  $a$ . These results were achieved by using Otter to prove the equivalence of the standard group axioms with exhaustively generated formulae. Similar results have been found for different operators in group theory and for Abelian groups, odd exponent groups and inverse loops. Kunen has subsequently proved that in many cases, there are no smaller single axiom schemes than those produced by Otter [13].

Also, in [5], Chou presents improvements on Wu's method for proving theorems in plane geometry [30]. He describes three approaches to using his prover to find new theorems: (i) ingenious guessing using geometric intuition (ii) numerical searching and (iii) a systematic approach based on the Ritt-Wu decomposition algorithm. Using the first approach, he found a construction based on Pappus' Theorem which led to a colinearity result believed to be new. Using the second method – suggesting additional points and lines within given geometric configurations – he started with a theorem of Gauss (that the midpoints of the three diagonals of a complete quadrilateral are collinear) and constructed a theorem about taking subsets of five lines from six, which he believed to be of importance. With the third method, he discovered a generalisation of Simson's theorem.

### 3. Background

In previous work we have developed a bootstrapping algorithm for generating classification theorems in algebraic domains of pure mathematics. Moreover, we have applied this approach to constructing new

classification results with respect to isomorphism, mainly in the algebraic domains of quasigroups and loops. In this section, we briefly outline this method, and summarise our previous results. For more a more detailed description of the algorithm, we refer the reader to [7].

The bootstrapping algorithm takes a set of properties,  $\mathcal{P}$ , a cardinality,  $n$ , and an equivalence relation,  $E$ , as input. The bootstrapping algorithm returns a decision tree that contains the classification theorem for the algebraic structures of order  $n$  that satisfy  $\mathcal{P}$  with respect to  $E$ , as well as a set of representants for each equivalence class. During the construction, a set of theorems are proved, which – taken together – prove the correctness and the completeness of the classification theorem. To illustrate the bootstrapping algorithm, we will use below an example where the properties are the axioms of quasigroup theory, the cardinality is 3 and the equivalence relation is isomorphism.

In detail, the method proceeds as follows. Firstly, it initialises a decision tree with root node  $\mathcal{N}$  labelled with the properties  $\mathcal{P}$ . We denote the properties that a node is labelled by as  $\mathcal{P}_{\mathcal{N}}$ . The algorithm then works iteratively, starting with the root node and moving on to successive nodes in the tree. It constructs an example of an algebraic structure of order  $n$  satisfying  $\mathcal{P}_{\mathcal{N}}$ . When no example can be produced, the algorithm will prove that no structure of size  $n$  with properties  $\mathcal{P}_{\mathcal{N}}$  can exist. When an example does exist, one of two things happen, either: (1) the process shows that the node represents an equivalence class with respect to  $E$ , i.e., it proves that all structures of order  $n$  that satisfy the properties  $\mathcal{P}_{\mathcal{N}}$  are equivalent to each other under  $E$ , or (2) the process constructs another algebraic structure satisfying  $\mathcal{P}_{\mathcal{N}}$ , which is not equivalent to the first one. Note that cases (1) and (2) are mutually exclusive and can be performed in parallel. For case (2), the algorithm computes a discriminating property  $P$  for the two structures, such that  $P$  holds for one structure and  $\neg P$  holds for the other.  $P$  is then used to further expand the decision tree by adding two new nodes  $\mathcal{N}'$  and  $\mathcal{N}''$  below  $\mathcal{N}$ , with labels  $\mathcal{P}_{\mathcal{N}'} = \mathcal{P}_{\mathcal{N}} \cup \{P\}$  and  $\mathcal{P}_{\mathcal{N}''} = \mathcal{P}_{\mathcal{N}} \cup \{\neg P\}$  respectively.

The algorithm then iterates over these nodes and adds to the tree accordingly. After new nodes have been created for each of these nodes, the above steps are carried out again. The algorithm terminates once no more expansions can be applied. Leaf nodes either represent equivalence classes or are empty, i.e., no structure exists with the properties given in the node. The final classification theorem comprises the disjunction of the properties which label the leaf nodes.

In our example, the algorithm first generates a quasigroup of size 3, and then tries to show that the definition of the quasigroup represents an isomorphism class (case 1). This fails, as we have not yet specialised

the quasigroup axioms. In parallel (case 2), the algorithm constructs another quasigroup which is not isomorphic to the first, and then finds an invariant algebraic property,  $P$ , of the first quasigroup which is not true of the second, namely that  $\exists b$  s.t.  $b \circ b = b$ . The algorithm then adds two nodes to the classification tree, representing the quasigroups with and without this property. In the next iteration, the algorithm proves that the definition of quasigroups not having property  $P$  (i.e., having no idempotent elements) defines an isomorphism class for quasigroups of size 3, and hence this becomes a leaf node. As the algorithm iterates to its conclusion, it invents three more properties in order to find and prove definitions for the five isomorphic classes of size 3 quasigroups. Once it is no longer able to grow the classification tree, it checks that the five definitions cover all size 3 quasigroups, i.e., it proves that each size 3 quasigroup must satisfy one of these definitions.

The bootstrapping algorithm is a framework that combines a host of reasoning techniques which play their part in achieving the overall goal. While the general framework is implemented in a Lisp environment, it relies on third party systems to generate algebras and discriminants, and to verify the construction of the decision tree at each step. We use the following methodologies in the different steps of the algorithm:

*Generating Algebras* We use model generation to construct algebras. In the first step, the algorithm calls a model generator to return an algebra corresponding to the input axioms. Throughout the process, model generators are used to construct algebras which are not equivalent to a given algebra. For the experiments described in [7], we used the model generator Mace [17], but we have replaced this by Sem [31] and Finder [25], as they are more effective in our domain. While Sem is generally the more powerful of the two, it has weaknesses when dealing with function symbols of arity greater than 2, which can be introduced when we employ Skolemisation techniques (see [19] for details). Nevertheless, all three model generators remain integrated in the Lisp environment of the bootstrapping algorithm and can be used for other tasks, as described, for example, in section 4.2.

*Generating Discriminants* The approach to constructing discriminating properties varies from equivalence relation to equivalence relation. When dealing with the isomorphism relation, we treated the generation of a discriminant for a pair of algebras as a machine learning problem, and successfully applied automated theory formation [6] and inductive logic programming [8] to such problems.

*Verifying Properties* Throughout the bootstrapping procedure, all the results coming from third party systems are independently verified by first order automated theorem provers. Thus, for a given discriminant  $P$  and two algebras  $Q$  and  $Q'$ , we show that (1)  $P$  is a proper

discriminant for the equivalence relation  $E$  [which means that if  $Q$  and  $Q'$  differ with respect to the property, then they cannot be members of the same equivalence class], (2)  $P$  holds for  $Q$ , and (3)  $P$  does not hold for  $Q'$ . Proving these properties explicitly guarantees the overall correctness of the constructed decision tree. The proofs themselves are generally not very challenging, and we have experimented with several provers. We generally employ Spass [29] for these tasks.

*Verifying Equivalence Classes* The most difficult verification problems occurring during the classification process involve showing that a given node of the classification tree forms an equivalence class with respect to the equivalence relation under consideration. In other words, we need to verify that the tree cannot be expanded any further and that we do indeed have a leaf node. More formally, we need to prove that, for a particular set of properties of a node,  $P$ , all algebras of cardinality  $n$ , which satisfy  $P$ , are equivalent, and every member of the equivalence class satisfies  $P$ . These types of proof are necessary to fully verify the completeness of a decision tree. Although the theorems are essentially second order, because we work in a finite domain, they can be expressed as propositional logic problems by enumerating all possible equivalence mappings for structures of cardinality  $n$  and thus can be made accessible to ATP systems. In our original experiments, described in [7], we used Spass for these problems, as it was the only system which coped with the massive clause normalisations required (cf. [28]). For later experiments, we replaced Spass by state of the art SAT solvers and Satisfiability Modulo Theories (SMT) solvers, and developed a range of encoding techniques for a diverse range of systems (cf. [19]). Currently, we have integrated zChaff [20] which can handle pure Boolean SAT problems, the DPLLT [10] system, which can handle ground equations, and CVCLite [3], which can also deal with finite quantification. While using SAT solvers increases the power of our algorithm, if translated naively, many of the proof problems would still be beyond the capabilities of state of the art systems. To enable us to solve these problems, we implemented some computer algebra algorithms in GAP [11] that exploit some mathematical domain knowledge to reduce complexity. We expand upon this usage of GAP in section 5.1, as we make use of the computer algebra routines we implemented for isomorphism tasks in the new application to isotopic classification.

In our experiments with isomorphism as the equivalence relation, we mainly concentrated on the domain of quasigroups and loops. A *quasigroup* is a non-empty set  $G$  together with a binary operation  $\circ$  that satisfies the property  $\forall a, b \in G. (\exists x \in G. x \circ a = b) \wedge (\exists y \in G. a \circ y = b)$ . This property is often called the Latin Square property and has the

effect that every element of  $G$  appears exactly once in every row and every column of the multiplication table of  $\circ$ . A *loop* is a quasigroup that contains a unit element, i.e., an element  $e$  such that  $\forall x \in G, x \circ e = e \circ x = x$ . We generated novel isomorphism classification theorems for quasigroups of orders 3 to 5 and loops of order 4 to 6, a partial classification of loops of order 7, as well as some classification theorems for quasigroups of orders 6 and 7 with additional special properties, e.g., idempotency. The largest decision tree we have so far generated is the full isomorphism classification of quasigroups of size 5. This contains 2875 nodes and 1411 isomorphism classes, but is relatively shallow, with a maximum depth of 23. Its completion took more than four months of processing time. For more details see <http://www.cs.bham.ac.uk/~vxs/quasigroups/>.

#### 4. Generating Isotopy Invariants

The isotopy equivalence relation is of considerable importance in quasigroup theory, hence we concentrate on isotopy in this paper. It is defined as follows:

**DEFINITION 1.** *We say two quasigroups  $(G, \cdot)$  and  $(H, *)$  are isotopic to each other — or  $G$  is an isotope of  $H$  — if there are bijective mappings  $\alpha, \beta, \gamma$  from  $G$  to  $H$  such that for all  $x, y \in G$ ,  $\alpha(x) * \beta(y) = \gamma(x \cdot y)$  holds.*

Isotopy is an equivalence relation and the classes induced by it are called isotopism classes. It is a generalisation of isomorphism, since  $G$  and  $H$  are isomorphic if  $\alpha = \beta = \gamma$ . In other words, while two quasigroups can be isotopic but not necessarily isomorphic to each other, all members of an isomorphism class belong to the same isotopism class. Importantly, every quasigroup is isotopic to a loop, which we call its loop-isotope [21]. Thus, in certain respects, we can restrict the classification of quasigroups to just the classification of loops.

##### 4.1. OVERVIEW

As mentioned above, an important function of the bootstrapping algorithm is to generate invariant properties which enable us to discriminate between two algebras belonging to different equivalence classes. For instance, when dealing with the isomorphism equivalence relation, if one example of an algebra was commutative (i.e.,  $\forall x, y, (x * y = y * x)$ ) and another was not, then they could not belong to the same isomorphism class. Our machine learning approach was able to generate such properties, given background concepts including the multiplication operation.

Unfortunately, such simple properties do not enable us to distinguish between members of different isotopy classes. For example, commutativity is not an isotopy invariant, as the isotopism  $(\alpha, \iota, \iota)$ , where  $\iota$  is the identity mapping and  $\alpha \neq \iota$ , can map a commutative quasigroup to a non-commutative isotope. Due to the more complicated nature of isotopy invariants, initial experiments with the learning system failed to identify any suitable isotopy invariants.

In light of this failure, we developed bespoke methods for generating three types of isotopy invariants that are used while producing isotopic classifications of loops, as described in sections 4.2 and 4.3. We first describe the generation of universal identities (potential quasigroup isotopy invariants), which builds on a concept introduced by Falconer [9]. We describe how we obtain these invariants using an interplay of model generation and theorem proving. We also present other types of invariants that are derived by systematically examining sub-blocks of loops, and are constructed using computer algebra techniques. While to the best of our knowledge these particular invariants have not been reported in the literature on loops, it would not surprise us if they are already known to some experts in loop theory. In any case, they provide a sufficient basis for the isotopy classification results that we have obtained thus far.

The main difference between the two types of invariants is that universal identities enable us to express an isotopy discriminant as a single universally quantified equation, thus in a very concise way. Sub-block invariants on the other hand rely on an exhaustive counting argument, which makes for more complicated and lengthy discriminants. However, sub-block invariants are a reliable way of arriving at a discriminant, whereas we have to search blindly for an discriminating universal identity. Moreover, there is no guarantee that there always exists a universal identities that can discriminate two non-isotopic loops. Thus, we only allow for a reasonable number of attempts to find a discriminant based on a universal identity before resorting to the counting argument using sub-blocks.

## 4.2. UNIVERSAL IDENTITIES

To generate universal identities, we closely follow Falconer's construction in [9] that derives them from given loop identities. We first define two additional operations  $\backslash$  and  $/$  on a pair of elements such that:

1.  $x \cdot (x \backslash y) = y$  and  $x \backslash (x \cdot y) = y$
2.  $(y/x) \cdot x = y$  and  $(y \cdot x)/x = y$

Note that these operations are well defined because loops are quasi-groups. We then define a loop identity as a universally quantified equality between two words  $w_1$  and  $w_2$ , i.e.,  $w_1, w_2$  are combinations of variables representing loop elements with respect to the loop operation  $\cdot$ . Given a loop identity  $w_1 = w_2$ , we can obtain a *derived* or *universal identity*  $\bar{w}_1 = \bar{w}_2$  by recursively applying the following transformations:

1. if  $w = x$ , then  $\bar{w} = x$ ;
2. if  $w = u \cdot v$ , then  $\bar{w} = (\bar{u}/y) \cdot (z \setminus \bar{v})$

Here  $y$  and  $z$  are arbitrary elements of a loop, i.e., new universally quantified variables. As an example, consider the two loops below:

$L_4$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	0	5	3	4
2	2	0	1	4	5	3
3	3	5	4	1	0	2
4	4	3	5	0	2	1
5	5	4	3	2	1	0

$L_8$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	0	4	5	3
2	2	0	1	5	3	4
3	3	5	4	1	0	2
4	4	3	5	0	2	1
5	5	4	3	2	1	0

The following universal identity holds for  $L_4$  but does not hold for  $L_8$ :

$$\forall x \forall y_1 \forall y_2 \cdot (x/y_1) \cdot (((x/y_1) \cdot (y_2 \setminus x)) \cdot (y_2 \setminus x)) = ((x/y_1) \cdot (y_2 \setminus x)) \cdot ((x/y_1) \cdot (y_2 \setminus x)).$$

This universal identity was derived from the following loop identity:

$$\forall x \cdot x \cdot ((x \cdot x) \cdot x) = (x \cdot x) \cdot (x \cdot x)$$

Falconer's concept of universal identity depends on deriving identities from equations that hold for the free loop. However, in our scenario we can only generate finite models and a directed search for identities holding for the free loop is infeasible. Thus to generate a large number of isotopy invariants, we employ a process of interleaving model generation and first order theorem proving with intermediate transformations of the respective results. Figure 1 schematically depicts this process. We first systematically generate identities  $I$  for which we check whether they are loop identities, by trying to generate a loop of size  $\leq 8$  that satisfies  $I$  using the model generator Mace. We use Mace since it allows for easily generating models of different sizes in a single run, and as the problems are relatively easy, Mace is sufficiently powerful. All identities for which a loop exists are then transformed into universal identities  $U$  as described above. Each  $U$  is then passed to at least one first order theorem prover in order to show that it is an isotopy invariant. We employ

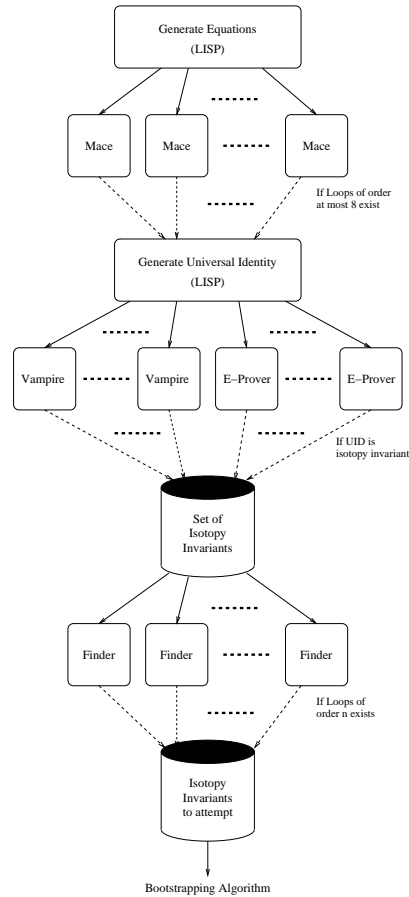


Figure 1. Generating isotopy invariants from universal identities.

both Vampire [22] and E [23] for this task. Combined, these show that around 80% of the universal identities are indeed isotopy invariants. Note that for each universal identity,  $U$ , we show that it is an invariant under isotopy independently of the size of a loop. We can therefore reuse these universal identities in different classifications. Consequently, we do not have to repeat this step in every classification, but rather perform it offline and collect universal identities in a pool of confirmed isotopy invariants. To date, we have generated 54,000 confirmed loop identities, which can be translated into universal identities. From these, we have attempted to prove 11,000 to be isotopy invariants and have succeeded to show this for 8,530 using both Vampire and E.

During the classification of loops of a particular size,  $n$ , we draw on this pool by first filtering them again by using the model generator Finder to generate loops of size  $n$  that satisfy the invariant. We have

chosen Finder over Mace for this task as it can deal more reliably with the multiple operations of our loop structures. We then extract those invariants for which at least one loop of order  $n$  exists, and we use only these as potential discriminants. Note that the filter discards those invariants which cannot solve any discrimination problem, as no loop of size  $n$  satisfies the invariant property. Then, when we need to discriminate between two loops, we test whether one of the invariants holds for one and not for the other, using DPLLT or CVCLite. While the model generation and theorem proving stages are run in parallel by distributing the problems on a large cluster of processors for both the generation and testing of invariants, finding a discriminant can still take a very long time. Moreover, universal identities are not necessarily sufficient to discriminate between two non-isotopic quasigroups, as there is no theoretical result guaranteeing this. We therefore explore only a limited number of randomly selected invariants, and if this is not successful, we employ two different methods for generating invariants, which are based on the sub-blocks of quasigroups, as described in section 4.3. A more goal-directed approach for finding discriminating universal identities would be desirable, and this is the subject of future work.

### 4.3. SUB-BLOCK INVARIANTS

To complement the previous method of universal identities, we have developed a more reliable method to find isotopy invariants that are based on exhaustive counting arguments. Invariants are derived by systematically examining sub-blocks of loops, and are constructed using computer algebra techniques. The required theoretical tools have been developed by us for this purpose and, to the best of our knowledge, have never been presented in that form before. The algorithms computing the sub-block invariants are implemented as Lisp functions in the same environment in which the overall bootstrapping algorithm is implemented.

#### 4.3.1. Sub-Blocks

Let  $(G, \cdot)$  be a quasigroup, and let  $A$  and  $B$  be non-empty subsets of  $G$ . We adopt the usual notation for the set  $A \cdot B$ , namely,  $A \cdot B = \{a \cdot b : a \in A \wedge b \in B\}$ .

**LEMMA 2.** *Let  $(G, \cdot)$  be a quasigroup and let  $(H, *)$  be a quasigroup that is isotopic to  $(G, \cdot)$  under the bijections  $(\alpha, \beta, \gamma)$ . Then, for any non-empty subsets  $A$  and  $B$  of  $G$ , we have  $|A \cdot B| = |\alpha(A) * \beta(B)|$ .*

*Proof.* Observe that since  $\gamma$  is a bijection, then  $|\gamma(A \cdot B)| = |A \cdot B|$ . It suffices then to show that  $\gamma(A \cdot B) = \alpha(A) * \beta(B)$ . But this follows

immediately from the fact that for all  $a \in A$  and  $b \in B$ , we have  $\gamma(a \cdot b) = \alpha(a) * \beta(b)$ .

When  $G$  is finite, one can interpret the elements of  $A$  (resp.,  $B$ ) as designating a subset of rows (resp., columns) in the multiplication table of  $G$ . The set  $A \cdot B$  then consists of the elements where these rows and columns meet. The above result thus suggests the following notation:

NOTATION 3. Let  $(G, \cdot)$  be a quasigroup of order  $n$ , and let  $i, j, k$  each be integers such that  $1 \leq i, j, k \leq n$ . Let  $G(i, j, k)$  denote the set:

$$G(i, j, k) = \{(A, B) : A, B \subseteq G, |A| = i, |B| = j, |A \cdot B| = k\}$$

THEOREM 4. Let  $(G, \cdot)$  and  $(H, *)$  be isotopic quasigroups of order  $n$ , and let  $i, j, k$  each be integers such that  $1 \leq i, j, k \leq n$ . Then  $|G(i, j, k)| = |H(i, j, k)|$ .

*Proof.* Note that the one-to-one correspondence between the collection of ordered pairs  $(A, B)$  such that  $A, B \subseteq G, |A| = i, |B| = j$ , and the corresponding collection of ordered pairs of subsets of  $H$ , is preserved under isotopy. The result now follows easily from Lemma 2.

EXAMPLE 5. As an example for sub-block invariants, consider the following two loops:

$L_{38}$	0	1	2	3	4	5	
	0	1	2	3	4	5	
	1	1	2	5	0	3	4
	2	2	5	0	4	1	3
	3	3	4	1	5	2	0
	4	4	0	3	1	5	2
	5	5	3	4	2	0	1

$L_{20}$	0	1	2	3	4	5	
	0	0	1	2	3	4	5
	1	1	2	0	5	3	4
	2	2	0	1	4	5	3
	3	3	4	5	1	2	0
	4	4	5	3	0	1	2
	5	5	3	4	2	0	1

Loop  $L_{38}$  contains 4 different  $2 \times 2$  sub-blocks that contain exactly 2 distinct elements, i.e., we have  $|L_{38}(2, 2, 2)| = 4$ . In detail, the single sub-blocks are the following:

	0	2
0	0	2
	2	0

	2	4	
1	5	3	
	4	3	5

	1	3	
2	5	4	
	3	4	5

	3	5	
4	1	2	
	5	2	1

For loop  $L_{20}$  on the other hand, we have  $|L_{20}(2, 2, 2)| = 0$  since it does not contain a single  $2 \times 2$  sub-block with two distinct elements. Thus we can discriminate the two loops using the invariant given as property  $P_9$

below, which expresses the fact that there exists a  $2 \times 2$  sub-block that contains exactly 2 distinct elements.

$$P_9: \exists r_1, r_2 \bullet \exists c_1, c_2 \bullet \exists! v_1, v_2 \bullet r_1 \neq r_2 \wedge c_1 \neq c_2 \wedge v_1 \neq v_2 \wedge \bigwedge_{k=1}^2 \bigwedge_{j=1}^2 \bigvee_{k=1}^2 (r_i \cdot c_j = v_k)$$

Note the use of the unique existence quantifier for variables  $v_1$  and  $v_2$ .

The above results form the basis for two more isotopy-invariants, which we now present: frequency tuples and patterns.

#### 4.3.2. Frequency Tuples

Continuing with the notation above, fix an element  $(A, B) \in G(i, j, k)$ , and for each  $g_h \in A \cdot B$ , with  $1 \leq h \leq k$ , let

$$f(g_h) = |\{(a, b) \in A \times B : a \cdot b = g_h\}|$$

In other words,  $f(g_h)$  is the number of times that  $g_h$  appears in the block formed by  $A$  and  $B$ , henceforth referred to as the  $A \cdot B$  block. We let  $F(A, B) = (f(g_1), \dots, f(g_k))$ , and call this the (un-ordered) *frequency-tuple* of  $(A, B)$ . If two such frequency-tuples  $F$  and  $F'$  are the same (up to order), then we write  $F \approx F'$ . (We thank the anonymous referee for his/her suggestion that frequency tuples could have been treated as multisets.)

LEMMA 6. *Let  $(G, \cdot)$  and  $(H, *)$  be isotopic quasigroups (under the bijections  $(\alpha, \beta, \gamma)$ ) of order  $n$ , and let  $i, j, k$  each be integers such that  $1 \leq i, j, k \leq n$ . If  $(A, B) \in G(i, j, k)$ , then  $F(A, B) \approx F(\alpha(A), \beta(B))$ .*

*Proof.* In light of Theorem 4, it suffices to prove that, for every  $g \in A \cdot B$ ,  $f(g) = f(\gamma(g))$ . But this equality follows immediately from the fact that if  $a \cdot b = g$ , then  $\alpha(a) * \beta(b) = \gamma(g)$ .

Given this latest result, we adopt the following notation:

NOTATION 7. *Let  $(G, \cdot)$  be a quasigroup of order  $n$ , let  $i, j, k$  be integers such that  $1 \leq i, j, k \leq n$ , and let  $F$  be a frequency-tuple for some  $(C, D) \in G(i, j, k)$ . Then, let  $G(i, j, k, F)$  denote the set:*

$$G(i, j, k, F) = \{(A, B) \in G(i, j, k) : F(A, B) \approx F\}$$

THEOREM 8. *Let  $(G, \cdot)$  be a quasigroup of order  $n$ , let  $i, j, k$  be integers such that  $1 \leq i, j, k \leq n$ , and let  $F$  be a frequency-tuple for some  $(C, D) \in G(i, j, k)$ . Furthermore, let  $(H, *)$  be a quasigroup isotopic to  $(G, \cdot)$ . Then  $|G(i, j, k, F)| = |H(i, j, k, F)|$ .*

*Proof.* This is an immediate consequence of Lemma 6 and Theorem 4.

EXAMPLE 9. To illustrate the idea of frequencies, consider again the loop  $L_{38}$  together with the two  $3 \times 3$  sub-blocks  $S_1$  and  $S_2$  given on the right:

$L_{38}$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	5	0	3	4
2	2	5	0	4	1	3
3	3	4	1	5	2	0
4	4	0	3	1	5	2
5	5	3	4	2	0	1

$S_1$	3	4	5
0	3	4	5
1	0	3	4
2	4	1	3

$S_2$	0	4	5
1	1	3	4
2	2	1	3
3	3	2	0

Both sub-blocks contain the same number of distinct elements, namely 5. However,  $S_1$  contains three elements  $(0, 1, 5)$  once and two elements  $(3, 4)$  three times, whereas  $S_2$  contains two elements  $(0, 4)$  once, two elements  $(1, 2)$  twice, and one element  $(3)$  three times. Thus  $S_1$  has the frequency tuple  $(1, 1, 1, 3, 3)$  and  $S_2$  has the frequency tuple  $(1, 1, 2, 2, 3)$ . Overall for  $L_{38}$  we have invariants  $|L_{38}(2, 2, 2, (1, 1, 1, 3, 3))| = 4$  and  $|L_{38}(2, 2, 2, (1, 1, 2, 2, 3))| = 52$ .

To generate isotopy invariants based on Theorems 4 and 8, we implemented an algorithm that compares the number of elements in sub-blocks for two quasigroups  $(G, \cdot)$  and  $(H, *)$ . This works iteratively, as follows: for  $i = 2, \dots, n-1$ ,  $j = 2, \dots, n-1$ , and  $k = \max(i, j), \dots, n$ , if  $|G(i, j, k)| \neq |H(i, j, k)|$  then return the invariant, otherwise continue. If all the possible sub-blocks are exhausted without yielding an invariant, we perform a frequency analysis for all the  $G(i, j, k)$  and  $H(i, j, k)$  until we find a pair  $G(i, j, k, F)$  and  $H(i, j, k, F')$ , such that  $F \neq F'$ . To keep the formulas resulting from these invariants small, we always prefer an existence argument over the actual comparison of numbers of sub-blocks. That is, we give a preference to invariants, such that either  $|G(i, j, k)| = 0$  or  $|H(i, j, k)| = 0$ .

#### 4.3.3. Patterns

Given non-empty subsets  $A$  and  $B$  of a quasigroup  $(G, \cdot)$ , we look for patterns amongst the numbers of distinct elements within the respective sub-blocks. By this, we mean the following: Let  $|A| = i$ ,  $|B| = j$ , and choose  $i', j'$  such that  $1 \leq i' \leq i$  and  $1 \leq j' \leq j$ . Now for each  $k$ ,  $1 \leq k \leq n$ , we let  $AB(i', j', k) = \{(A', B') : A' \subseteq A, B' \subseteq B, |A'| = i', |B'| = j', |A' \cdot B'| = k\}$ . Furthermore, let  $p_k = |AB(i', j', k)|$ . In other words,  $p_k$  is the number of  $i' \times j'$  sub-blocks of the  $A \cdot B$  block, that have precisely  $k$  distinct entries. We now let  $\mathfrak{P}_{i', j'}(A, B) = (p_1, \dots, p_n)$ , and we call  $\mathfrak{P}_{i', j'}(A, B)$  the  $i' \times j'$  pattern-tuple of  $(A, B)$ .

LEMMA 10. Let  $(G, \cdot)$ ,  $(H, *)$ ,  $(\alpha, \beta, \gamma)$ ,  $i, j, k, n$  be as in Lemma 6, and let  $i', j'$  be integers such that  $1 \leq i' \leq i$  and  $1 \leq j' \leq j$ . If  $A, B \subseteq G$  such that  $|A| = i$  and  $|B| = j$ , then  $\mathfrak{P}_{i',j'}(A, B) = \mathfrak{P}_{i',j'}(\alpha(A), \beta(B))$ .

*Proof.* Note that for each  $k$ ,  $1 \leq k \leq n$ , and for each  $(A', B') \in AB(i', j', k)$ , we have  $|A' \cdot B'| = |\alpha(A') * \beta(B')|$ , by Lemma 2. Now since  $\alpha$  and  $\beta$  are bijections, then  $(A', B') \in AB(i', j', k)$  if and only if  $(\alpha(A'), \beta(B')) \in \alpha(A)\beta(B)(i', j', k)$ . The result follows.

Following similar lines as previously, we introduce the following notation:

NOTATION 11. Let  $(G, \cdot)$  be a quasigroup of order  $n$ , and let  $\mathfrak{P}_{i',j'}$  be an  $i' \times j'$  pattern-tuple of  $(C, D)$  for some  $C, D \subseteq G$  such that  $|C| = i$  and  $|D| = j$  ( $1 \leq i, j \leq n$ ), where integers  $i', j'$  are such that  $1 \leq i' \leq i$  and  $1 \leq j' \leq j$ . We let  $G(i, j, \mathfrak{P}_{i',j'})$  denote the set:

$$G(i, j, \mathfrak{P}_{i',j'}) = \{(A, B) : A, B \subseteq G, |A| = i, |B| = j, \mathfrak{P}_{i',j'}(A, B) = \mathfrak{P}_{i',j'}\}$$

THEOREM 12. Let  $(G, \cdot)$  be a quasigroup of order  $n$ , and let  $\mathfrak{P}_{i',j'}$  be an  $i' \times j'$  pattern-tuple of  $(C, D)$  for some  $C, D \subseteq G$  such that  $|C| = i$  and  $|D| = j$  ( $1 \leq i, j \leq n$ ), where integers  $i', j'$  are such that  $1 \leq i' \leq i$  and  $1 \leq j' \leq j$ . Furthermore, let  $(H, *)$  be a quasigroup isotopic to  $(G, \cdot)$ . Then  $|G(i, j, \mathfrak{P}_{i',j'})| = |H(i, j, \mathfrak{P}_{i',j'})|$ .

*Proof.* This follows immediately from Lemma 10.

EXAMPLE 13. We illustrate patterns with the example of loop  $L_{25}$  below in which we are interested in  $2 \times 2$  pattern tuples within  $4 \times 4$  sub-blocks. The particular sub-block  $S$  below contains exactly one  $2 \times 2$  sub-block with exactly two distinct elements.

$L_{25}$	0	1	2	3	4	5	$S$	1	2	3	4	$S'$	2	4
0	0	1	2	3	4	5	2	0	1	5	3	2	1	3
1	1	2	0	4	5	3	3	5	4	1	0	5	3	1
2	2	0	1	5	3	4	4	3	5	0	2	5	3	1
3	3	5	4	1	0	2	5	4	3	2	1			
4	4	3	5	0	2	1								
5	5	4	3	2	1	0								

The overall pattern tuple for  $S'$  is  $\mathfrak{P}_{2,2} = (0, 1, 12, 23, 0, 0)$ . The invariant for  $L_{25}$  counting the number of  $4 \times 4$  sub-blocks with a pattern-tuple  $\mathfrak{P}_{2,2} = (0, 1, 12, 23, 0, 0)$  is  $|L_{25}(4, 4, \mathfrak{P}_{2,2})| = 18$ .

In order to generate additional invariants for a given pair of quasigroups  $(G, \cdot)$  and  $(H, *)$ , we employ Theorem 12 in a similar fashion to that used in section 4.3. That is, we successively compare the number of patterns of the same size and the same number of distinct elements.

Observe that, rather than considering the entire  $i' \times j'$  pattern-tuple of  $(A, B)$ , we could instead, for instance, focus on only one component at a time, which simplifies the resulting invariant properties. With this in mind, we let  $\mathfrak{P}_{i',j'}(A, B)_{(k)}$  denote the  $k$ -th component of the ordered  $n$ -tuple  $\mathfrak{P}_{i',j'}(A, B)$ . It is obvious then that, in the context of Lemma 10, we have  $\mathfrak{P}_{i',j'}(A, B)_{(k)} = \mathfrak{P}_{i',j'}(\alpha(A), \beta(B))_{(k)}$ . This leads to some further notation:

NOTATION 14. *Let  $(G, \cdot)$  be a quasigroup of order  $n$ , and let  $i, j, i', j', k, p_k$  be integers such that  $1 \leq i, j, k \leq n$ ,  $1 \leq i' \leq i$ ,  $1 \leq j' \leq j$  and  $p_k \geq 0$ . We let  $(G, i, j, i', j', k, p_k)$  denote the set:*

$$(G, i, j, i', j', k, p_k) = \{(A, B) : A, B \subseteq G, |A| = i, |B| = j \text{ and } \mathfrak{P}_{i',j'}(A, B)_{(k)} = p_k\}$$

Note that we have effectively proved the following corollary:

COROLLARY 15. *Let  $(G, \cdot), (H, *)$ ,  $i, j, i', j', k, n$  be as in Lemma 10, and let  $p_k$  be a non-negative integer. Then*

$$|(G, i, j, i', j', k, p_k)| = |(H, i, j, i', j', k, p_k)|.$$

We now combine the notions of patterns and frequencies, by first expanding the notation  $AB(i', j', k)$ , described at the beginning of this section. The idea is that, for a given  $(A, B) \in G(i, j, \mathfrak{P}_{i',j'})$ , we want to see how the frequency-tuples  $F(A', B')$  are distributed, for each  $(A', B') \in AB(i', j', k)$  ( $1 \leq k \leq n$ ). To this end, for a frequency-tuple  $F$  of size  $k$ , we let  $AB(i', j', k, F) = \{(A', B') \in AB(i', j', k) : F(A', B') \approx F\}$ . This effectively partitions the collection of  $i' \times j'$  sub-blocks of the  $A \cdot B$  block, according to both their number of distinct elements and their frequency-tuples. We refer to this partition as the  $i' \times j'$  frequency distribution  $\mathfrak{F}_{i',j'}(A, B)$  of the  $A \cdot B$  block.

Now since each of these properties – number of distinct elements, frequency-tuples, and pattern-tuples – is preserved under isotopism, then it follows that the number of  $i \times j$  blocks with both a given pattern-tuple  $\mathfrak{P}_{i',j'}$  and a given frequency distribution  $\mathfrak{F}_{i',j'}$ , is likewise preserved under isotopism. We could, of course, extend these properties recursively, by looking at sub-blocks of sub-blocks, and so on. It is an interesting question whether such a recursive extension would suffice to completely classify all (finite) loops. Regardless of the answer to this question, one would hope to eventually find other quite different types of classifying properties, and perhaps better still, to find ways to automate their discovery.

## 5. Making Isotopy Class Theorems Tractable

The most difficult verification problems which occur during the classification process involve showing that a given node forms an isotopy class, i.e., we need to prove that a particular set of properties is indeed classifying with respect to isotopy. The opposite problem is to take a given node that doesn't form an isotopy class, and construct a representant that is not isotopic to the existing representant of the node. To solve these two problems with automated theorem provers and model generators requires considerable effort, as their naive formalisations yield problems of an intractable size that are not solvable using state of the art techniques, such as satisfiability solving. We therefore employ several computational methods, implemented in the computer algebra system GAP [11], for reducing the complexity of the problems to be solved.

The methods we developed for isotopism problems make use of those we developed for the corresponding isomorphism problems. The verification problem for isomorphism classes, as well as its opposite model generation problem, can become, when approached naively, computationally intractable even for very small structures. Unfortunately, the computational techniques we have developed to reduce the complexity of these problems cannot be directly transferred to the corresponding isotopism case. We therefore exploit some well-known results from the isotopy theory of quasigroups in order to adapt the techniques developed for the isomorphism case to the isotopism case. Hence, we first give a brief account of how we handled isomorphism problems (for further details see [19]), and then present their adaptation for isotopism, using theorems presented in [21].

### 5.1. HANDLING OF ISOMORPHISM PROBLEMS

In general, to prove that a particular set of properties associated with a node on the decision tree is classifying with respect to isomorphism is a higher-order problem, as it requires proving that all structures satisfying the properties are isomorphic. However, since we are concerned with finite structures, the proof problem can be reduced to first-order and even propositional logic by enumerating all (finitely many) possible isomorphic structures of the representant,  $Q$ , of the node. With all isomorphic structures available, it suffices to prove that each structure that satisfies the properties of the node equals one of the isomorphic structures. For the construction of a non-isomorphic structure, it suffices to generate a structure that satisfies the properties of the node but differs from each of the isomorphic structures.

A naive approach to enumerate all isomorphic structures would consider all  $n!$  possible bijective mappings for structures of cardinality  $n$ . From each of these bijective mappings, a structure can be constructed that is isomorphic to  $Q$ , and these  $n!$  are all possible isomorphic structures of  $Q$ . This naive approach can be simplified by the usage of generating systems. A structure  $Q$  with binary operation  $\circ$  is said to be *generated* by a set of elements  $\{q_1, \dots, q_m\} \subseteq Q$  if every element of  $Q$  can be expressed as a combination — usually called a factorisation or word — of the  $q_i$  under the operation  $\circ$ . We call a set of generators together with the corresponding factorisations a *generating system*. Given a generating system, we can exploit the fact that each isomorphism is uniquely determined by the images of the generators, to reduce the total number of isomorphisms that we need to consider. If  $n$  is the cardinality of the structures and  $m$  is the number of generators, then, instead of  $n!$ , there are only  $\frac{n!}{(n-m)!}$  possible mappings and possible resulting structures to consider, since only the  $m$  generators have to be mapped explicitly.

In theory, the worst case complexity resulting from the simplification with generating systems is still  $n!$ , because there is no guarantee that a generating system with a small number of generators exists. In our experiments, however, the number of generators was typically only 2, so the complexity for isomorphism problems was reduced from  $n!$  to  $n^2$ .

This approach requires additional effort in order to compute a generating system for  $Q$ , as well as to prove that a computed system is indeed a generating system for  $Q$ . The former task is clearly a purely computational problem that is most efficiently solved using a computer algebra approach. We have therefore implemented an algorithm in GAP that given a quasigroup  $Q$  computes a minimal generating system for  $Q$  by generating and comparing the traces of all elements in  $Q$ . The latter task is then to certify that the result of the algorithm is indeed a generating system for  $Q$ . This is a proof problem, which is trivial even for large  $n$ , and is easily solved by a theorem proving system.

## 5.2. HANDLING OF ISOTOPISM PROBLEMS

Unfortunately, we cannot directly employ the trick of using generating systems since we now have three, instead of one, bijective mappings to consider, so the images of the generators no longer uniquely determine isotopisms. This means a naive approach to performing the isotopism proof or model construction would have to consider all  $(n!)^3$  possible triples of bijective mappings to be applied to a representant  $Q$  of a node in the classification tree. As with the case of isomorphism, from

the  $(n!)^3$  possible triples of bijective mappings, all  $(n!)^3$  structures that are isotopic to  $Q$  can be computed. With all the isotopes of  $Q$  available, an automated theorem prover can be used to show that each structure that satisfies the properties of the node equals one of the isotopes. Moreover, a model generator can be asked to compute a structure that satisfies the properties of the node but differs from each of the isotopes. However, we have found that this naive approach exceeds the abilities of state of the art SAT and SMT solvers, even for  $n = 4$ .

In order to simplify our problems, particularly by making use of the reduction offered by generating systems, we exploit the following known results from the isotopy theory of quasigroups (see [21] for details).

**DEFINITION 16.** *A quasigroup  $(G, \cdot)$  is a principal isotope of the quasigroup  $(G, *)$  if there are permutations  $\alpha, \beta$  on  $G$  such that for all  $x, y \in G, \alpha(x) * \beta(y) = x \cdot y$ .*

In other words, the third permutation  $\gamma$  of the isotopy definition 1 is replaced by the identity permutation in definition 16.

**THEOREM 17.** *If  $(G, \cdot)$  and  $(H, *)$  are isotopic quasigroups, then  $(H, *)$  is isomorphic to some principal isotope of  $(G, \cdot)$ .*

**DEFINITION 18.** *Let  $(G, \cdot)$  be a quasigroup. Let  $f$  and  $g$  be fixed elements of  $G$ . Then the isotope  $(G, *)$  such that  $(x \cdot g) * (f \cdot y) = x \cdot y$  for all  $x, y \in G$  is called the  $fg$ -isotope of  $(G, \cdot)$ .*

**THEOREM 19.** *Let  $(G, \cdot)$  and  $(H, *)$  be quasigroups.  $H$  is isotopic to  $G$  if and only if it is isomorphic to an  $fg$ -isotope of  $G$ .*

Moreover, it is easy to show that every  $fg$ -isotope is, in fact, a loop with unit element  $fg$ . Consequently, every quasigroup has a loop isotope, and hence, rather than compute all  $(n!)^3$  isotopes for a structure  $Q$ , it suffices to:

1. compute the set of all  $fg$ -isotopes of  $Q$ ,  $\mathcal{FG}(Q)$ ,
2. remove structures from  $\mathcal{FG}(Q)$  that are isomorphic to other structures in  $\mathcal{FG}(Q)$  until the resulting set  $\mathcal{FG}'(Q)$  contains no pair of isomorphic structures,
3. compute the set of all structures that are isomorphic to a structure in  $\mathcal{FG}'(Q)$ . We denote this set:  $\mathcal{I}_{fg}(Q)$ .

For a given representant,  $Q$ , there are  $n^2$   $fg$ -isotopes and for each  $fg$ -isotope there are  $n!$  isomorphic structures. Hence, our simplification

reduces the complexity from  $(n!)^3$  structures in the naive approach to  $n!n^2$  structures. While this already gives a considerable improvement, the resulting complexity still exceeds the capabilities of state of the art SAT and SMT solvers, even for small  $n$ . We now observe that step 3 discusses isomorphisms only, which consequently allows us to exploit the generating systems simplification developed for the isomorphism problems as described in section 5.1. Hence, instead of step 3 above, we can perform the following alternative steps:

- 3a. compute a generating system for each structure in  $\mathcal{FG}'(Q)$ , respectively,
- 3b. for each pair of a structure in  $\mathcal{FG}'(Q)$  and its generating system, compute the set of isomorphic structures with respect to the generating system, respectively. Then, compute the union  $\mathcal{G}_{fg}(Q)$  of all resulting structures.

In theory, the worst case complexity of  $\mathcal{G}_{fg}(Q)$  is still  $n!n^2$ . However, in our experiments, we found that the complexity of  $\mathcal{FG}'(Q)$  is typically  $n$  rather than  $n^2$ , and the complexity resulting from the isomorphisms typically comes to  $n^2$  instead of  $n!$  when the generating system simplification is exploited. Hence, in practise, the simplified isotopism proof and model generation formalisations typically require the consideration of just  $n^3$  structures.

The single steps of this procedure are realised as follows: For step 1 we have designed another bespoke computer algebra algorithm that systematically computes all  $fg$ -isotopes of the structure  $Q$  and retains only those that are indeed loops. While this computation could have theoretically been done using a model generator, a dedicated algorithm is both more efficient and reliable. In step 2, we then sift the set  $\mathcal{FG}(Q)$  to retain only the elements that are pairwise non-isomorphic by performing a pairwise comparison between its elements. This is a relatively simple proof problem and we handle it with a theorem prover. For step 3a, we re-use the computer algebra algorithm and its certification we had designed and implemented in GAP for the isomorphism classification as described in section 5.1. Finally, in step 3b we can again re-use the machinery already implemented for the isomorphism classification in order to compute for each structure in  $\mathcal{FG}'(Q)$  the possible isomorphism mappings, finishing by simply disjoining the result.

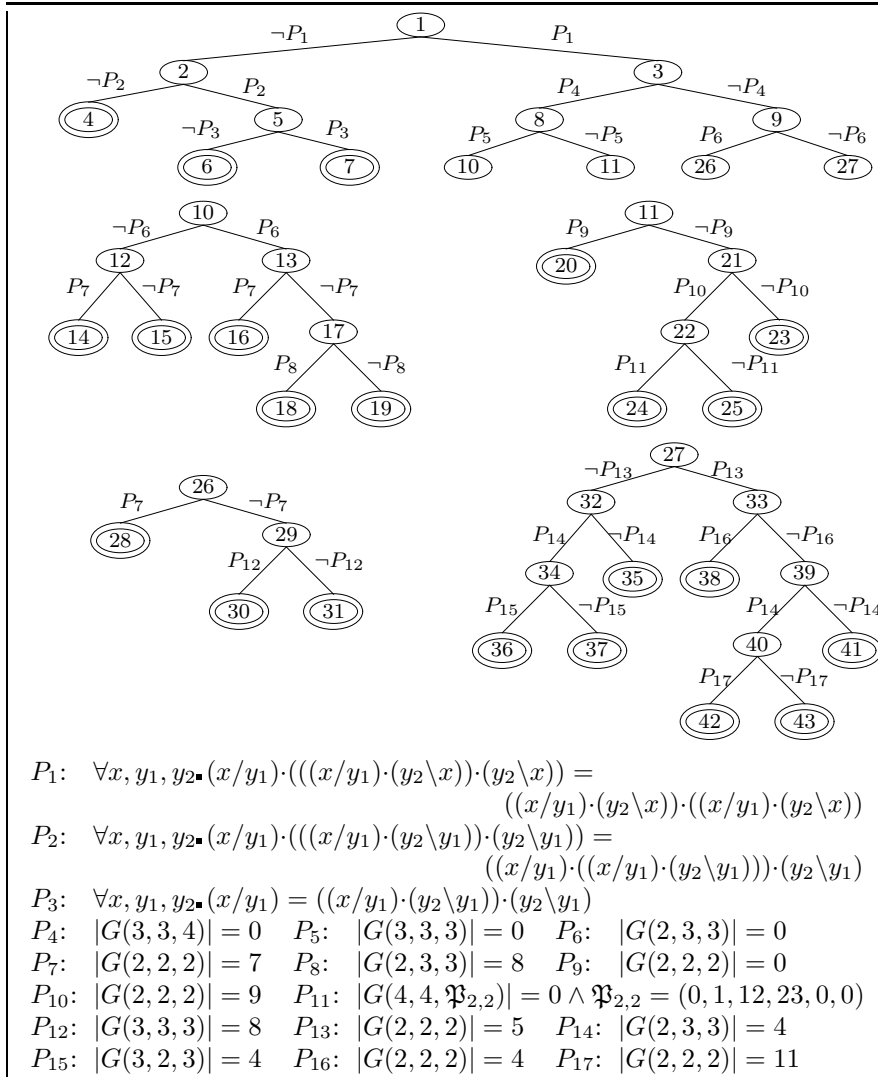


Figure 2. Decision tree and discriminating properties for the loops 6 classification.

## 6. Results

Our primary results are new qualitative isotopy classification theorems for loops of order 6 and 7, in the sense that they provide full sets of classifiers and corresponding representants for the already known number of isotopism classes, 22 and 564, respectively. Due to the rather large size of the order 7 theorem, we only discuss the order 6 results in

detail, but summarise the results for the order 7 theorem at the end of this section.

The decision tree for the order 6 theorem is given in figure 2 (for display purposes, we have broken it up into five parts). The entire tree consists of 43 nodes, where the doubly circled leaf nodes represent isotopy classes. The respective classifying properties correspond to the conjunction of discriminants given along the path from the leaves to the root. The discriminants themselves are given below the tree in figure 2.

Note that the top three discriminants are universal identities, while the remainder are sub-block invariants, with the exception of  $P_{11}$ , which is a pattern invariant. The pattern invariant and four of the sub-block invariants ( $P_4, P_5, P_6, P_9$ ) use an existence argument. The reason for the relatively low number of universal identities as discriminants is that in each step we only test 100 invariants randomly selected out of the overall pool of known invariants before testing for discriminating sub-block properties. This restriction is necessary as in our current implementation, testing universal identities with respect to discrimination still consumes a lot of search time. However, we hope that with more advanced encoding techniques and better pre-selection criteria for universal identity invariants, we will be able to replace more sub-block discriminants in the decision tree by universal identities.

For the construction of the models in the tree, we used the Sem, Finder, and DPLLT systems. Apart from the proof problems concerning universal identities, which were carried out by Vampire or E, the majority of the verification proofs have been done with CVClite or DPLLT. A translation into a purely propositional problem without equality, was prohibitive, however, as both the number of variables and nesting depth of the quantifications led to intractably large Boolean satisfiability problems of several hundred thousand Boolean variables. While most properties of the tree are fully automatically verified, we still had difficulty proving that the leaf nodes in the decision tree that contain discriminating properties  $P_8, P_{11}, P_{12}$  are indeed isotopy classes. However, we hope to finalise this by shifting to the new version of DPLLT in the future. Moreover, given that the number of isotopy classes of order 6 is indeed 22 as known from the literature [18, 21] and given the proved discriminating nature of the classifying properties, the decision tree does indeed constitute a classification theorem.

We have also finished the classification theorem for loops of order 7 with respect to sub-block invariants. During the construction of the decision tree, in many cases, the model generation fails to produce any results, so we re-use results from our isomorphism classification of loops of order 7, as well as enumeration results by McKay and Myrwold [18], to search for non-isotopic models. That is, we start with

a set of isomorphism class representants of order 7 loops and search amongst those for one that is non-isotopic to a given loop. This has the effect that we can construct an optimal tree with altogether 1127 nodes, 564 leaves and a total of 563 discriminants, which are of the form:

- 439 sub-blocks invariants,
- 7 frequency invariants,
- 116 pattern invariants,
- 1 invariant that relies on extending the concept of patterns by a recursion depth, i.e., we additionally count patterns inside patterns.

We are currently endeavouring to find more discriminants based on universal identities to replace some of the sub-block invariants.

## 7. Conclusions and Future Work

We have shown how the bootstrapping algorithm developed in [7] has been extended to produce isotopy classification theorems in loop theory. This involved solving two technical problems, namely the generation of isotopic invariants, and verifying that nodes in the classification tree define isotopy classes. To solve the first problem, we used the notion of universal identities to generate a pool of thousands of invariants, and we also developed new techniques for using concepts based on sub-blocks as invariants. To solve the second problem, we combined results from algebraic quasigroup theory in a novel computational way to extend techniques we developed in the context of isomorphism classification to isotopy problems. This not only significantly simplified the verification tasks but also made the related task of generating non-isotopic models feasible.

It has become clear that it will be difficult to exceed loops of size 7 or 8 with our existing techniques. In particular, the computational techniques to generate sub-block invariants need to be improved in order to scale up to larger sizes. This problem can be partially overcome by using more universal identities as discriminants. To this end, it is necessary to cut down on search time by using more intelligent pre-selection of universal identity invariants, as well as producing better reformulations of problems involving universal identities to make them accessible to efficient SAT solving techniques. It is also not clear that our current set of invariants will suffice for all orders. Indeed, we have already investigated another kind of invariant – which extends the

concept of frequency to patterns – but which has not been used in the context of the results presented in this paper. Finding new types of isotopy invariants for loops is certainly interesting from a mathematical viewpoint.

We plan to extract details of the bespoke invariant generation techniques developed for isotopy into more extensive background knowledge and improved production rules for the machine learning approach. We believe that it may be possible to re-instate the learning approach, thus restoring a more generic approach to invariant discovery, and possibly discovering new invariants. We also plan to undertake a detailed analysis of the decision trees, as well as a comparison of classification theorems of different sizes for particular structures. From a purely technical point of view, an efficient re-implementation of the entire bootstrapping algorithm would be desirable to overcome some limitations of the Lisp implementation and choice of data structures. We also intend to revisit the discrimination tree for order 7 loops described above, and simplify it by searching for discriminating universal identities. Given the complexity of the bootstrapping process, and in particular the fact that it uses many different reasoning systems, we are also considering enabling the process to produce proof objects, which could be independently verified by a proof checker.

Historically, classification projects have been undertaken for algebras with relatively few classes for small orders (e.g., there are only 5 groups of size 8 up to isomorphism). However, classification of other algebras are no less valid projects, albeit ones which are possibly more suited to an automated approach, due to the large number of classes of small order (e.g., there are 106,228,849 loops of size 8 up to isomorphism and 1,676,267 up to isotopism). From a mathematical discovery point of view, our next step is to attempt to identify global invariants of loops which can be used in their classifications. In particular, we aim to discover *families* of loops, parameterised by a pair  $\langle P, f(n) \rangle$ , where  $P$  is an algebraic property and  $f(n)$  is a Boolean function which determines whether  $P$  is a classifying invariant for loops of size  $n$ . For instance, in group theory,  $P$  could be the property of being dihedral, and  $f(n)$  would return true for all even numbers. Such descriptions of families are the building blocks of full classification theorems, and their discovery would attract much attention from mainstream mathematics. However, proving the correctness of such family descriptions would be a significant challenge to automated reasoning, not least because they are higher order propositions, and the mathematical proofs of similar results – in group theory for example – tend to be fairly involved.

We have shown that it is possible to make progress on large classification projects, but that this requires the combination of different

reasoning methods, including theorem proving, model generation, machine learning, satisfiability solving and computer algebra techniques. We advocate not just the improvement of particular automated reasoning techniques, but also the combination of reasoning techniques so that the whole is more than the sum of the parts (as also advocated in [4]). We believe that such integrated approaches will lead the way in the application of automated reasoning to complex mathematical discovery tasks.

### Acknowledgements

We would like to thank the anonymous reviewers for their very helpful comments. We would also like to thank the journal special issue organisers and the organisers of the IJCAR'06 conference for all their efforts.

### References

1. Alur, R. and D. Peled (eds.): 2004, 'Computer Aided Verification, 16<sup>th</sup> International Conference, CAV 2004', Vol. 3114 of *LNCS*. Boston, MA, USA.; Springer Verlag.
2. Aschbacher, M.: 2004, 'The Status of the Classification of Finite Simple Groups'. *Notices of the AMS* **51**(7), 736–740.
3. Barrett, C. and S. Berezin: 2004, 'CVC Lite: A New Implementation of the Cooperating Validity Checker'. in [1], pp. 515–518.
4. Bundy, A.: 2007, 'Cooperating Reasoning Processes: More than just the sum of their parts'. In: *Twentieth International Joint Conference on Artificial Intelligence*. pp. 2–11.
5. Chou, S.: 1985, 'Proving and Discovering Geometry Theorems using Wu's Method'. Technical Report 49, Computing Science, University of Austin at Texas.
6. Colton, S.: 2002, *Automated Theory Formation in Pure Mathematics*. Springer.
7. Colton, S., A. Meier, V. Sorge, and R. McCasland: 2004, 'Automatic Generation of Classification Theorems for Finite Algebras'. In: D. Basin and M. Rusinowitch (eds.): *Automated Reasoning — 2nd International Joint Conference, IJCAR 2004*, Vol. 3097 of *LNAI*. Cork, Ireland, pp. 400–414.
8. Colton, S. and S. Muggleton: 2006, 'Mathematical Applications of Inductive Logic Programming'. *Machine Learning* **64**, 25–64.
9. Falconer, E.: 1970, 'Isotopy Invariants in Quasigroups'. *Transactions of the American Mathematical Society* **151**(2), 511–526.
10. Ganzinger, H., G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli: 2004, 'DPLL(T): Fast Decision Procedures'. in [1], pp. 175–188.
11. GAP: 2002, 'GAP – Groups, Algorithms, and Programming, Version 4.3'. The GAP Group. <http://www.gap-system.org>.

12. Kronecker, L.: 1870, ‘Auseinandersetzung einiger Eigenschaften der Klassenanzahl idealer komplexer Zahlen’. *Monatsbericht der Berliner Akademie* pp. 881–889.
13. Kunen, K.: 1992, ‘Single Axioms for Groups’. *Journal of Automated Reasoning* **9**(3), 291–308.
14. Manna, Z. and C. Zarba: 2003, ‘Combining decision procedures’. In: *Formal Methods at the Cross Roads: From Panacea to Foundational Support*, Lecture Notes in Computer Science. Springer.
15. McCune, W.: 1992, ‘Automated discovery of new axiomatizations of the left group and right group calculi’. *Journal of Automated Reasoning* **9**(1), 1–24.
16. McCune, W.: 1993, ‘Single Axioms for Groups and Abelian Groups with Various Operations’. *Journal of Automated Reasoning* **10**(1), 1–13.
17. McCune, W.: 2003, ‘Mace4 Reference Manual and Guide’. Argonne National Laboratory. ANL/MCS-TM-264.
18. McKay, B. D., A. Meynert, and W. Myrvold: 2002, ‘Counting Small Latin Squares’. In: *European Women in Mathematics International Workshop on Groups and Graphs*. Varna, Bulgaria, pp. 67–72.
19. Meier, A. and V. Sorge: 2005, ‘Applying SAT Solving in Classification of Finite Algebras’. *Journal of Automated Reasoning* **35**(1–3), 201–235.
20. Moskewicz, M., C. Madigan, Y. Zhao, L. Zhang, and S. Malik: 2001, ‘Chaff: Engineering an efficient SAT Solver’. In: *Proc. of the 39<sup>th</sup> Design Automation Conference (DAC 2001)*. Las Vegas, USA, pp. 530–535.
21. Pflugfelder, H. O.: 1990, *Quasigroups and Loops: Introduction*, Vol. 7 of *Sigma Series in Pure Mathematics*. Berlin, Germany: Heldermann Verlag.
22. Riazanov, A. and A. Voronkov: 2001, ‘Vampire 1.1’. In: R. Goré, A. Leitsch, and T. Nipkow (eds.): *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, Vol. 2083 of *LNAI*. Siena, Italy, pp. 376–380.
23. Schulz, S.: 2002, ‘E: A Brainiac theorem prover’. *Journal of AI Communication* **15**(2–3), 111–126.
24. Schwenk, J.: 1995, ‘A classification of Abelian quasigroups’. *Rendiconti di Matematica, Serie VII* **15**, 161–172.
25. Slaney, J.: 1995, ‘FINDER, Notes and Guide’. Center for Information Science Research, Australian National University.
26. Slaney, J., M. Fujita, and M. Stickel: 1995, ‘Automated Reasoning and Exhaustive Search: Quasigroup existence problems’. *Computers and Mathematics with Applications* **29**, 115–132.
27. Sorge, V., A. Meier, R. McCasland, and S. Colton: 2006, ‘The Automatic Construction of Isotopy Invariants’. In: *Third International Joint Conference on Automated Reasoning*.
28. Sutcliffe, G.: 2005, ‘The IJCAR-2004 Automated Theorem Proving Competition’. *AI Communications* **18**(1), 33–40.
29. Weidenbach, C., U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic: 2002, ‘SPASS Version 2.0’. In: A. Voronkov (ed.): *Proc. of the 18th International Conference on Automated Deduction (CADE-18)*, Vol. 2392 of *LNAI*. pp. 275–279.
30. Wu, W.: 1984, ‘Basic principles of mechanical theorem proving in geometries’. *Journal of System Sciences and Mathematical Sciences* **4**(3), 207–235.
31. Zhang, J. and H. Zhang: 2001, ‘SEM User’s Guide’. Department of Computer Science, University of Iowa.