Applying SAT Solving in Classification of Finite Algebras

Andreas Meier (ameier@dfki.de)
DFKI GmbH, Saarbrücken, Germany

Volker Sorge (v.sorge@cs.bham.ac.uk)
School of Computer Science, University of Birmingham, UK

Abstract. The classification of mathematical structures plays an important role for research in pure mathematics. It is, however, a meticulous task which can be aided by using automated techniques. Many automated methods concentrate on the quantitative side of classification, like counting isomorphism classes for certain structures with given cardinality. In contrast, we have devised a bootstrapping algorithm that performs qualitative classification by producing classification theorems that describe unique distinguishing properties for isomorphism classes. In order to fully verify the classification it is essential to prove a range of problems, which can become quite challenging for classical automated theorem provers even in the case of relatively small algebraic structures. But since the problems are in a finite domain, employing Boolean satisfiability solving is possible.

In this paper we present the application of satisfiability solvers to generate fully verified classification theorems in finite algebra. We explore diverse methods to efficiently encode the arising problems both for Boolean SAT solvers as well as for solvers with built-in equational theory. We give experimental evidence for their effectiveness, which leads to an improvement of the overall bootstrapping algorithm.

Keywords: Application of SAT, Finite Algebra, Mathematics

1. Introduction

The classification of finite algebraic structures is an important task in research in pure mathematics. Often, the first step towards full classification is to determine how many structures exist up to isomorphism for each cardinality. In particular, in domains where many structures have to be considered, this is an laborious task, which can be supported by automated techniques. For instance, isomorphism free enumeration techniques can be applied to count isomorphism classes for quasigroups and loops up to order 11 [15, 14]. While quantitative results of this type already give some insight into the size and complexity of an algebraic domain, classification theorems of a more qualitative nature are often more interesting. Their information can sometimes allow one to use properties of relatively small structures to help classify larger structures.

Automated techniques such as constraint solving and the Davis-Putnam method have been used extensively to determine the number of algebras of a given type and size, and this has answered many open questions. For instance, [6, 12, 20, 27] report on the use of model generation techniques and satisfiability (SAT) solvers to tackle quasigroup existence problems. J. Zhang was the first to use a general reasoning program to solve an open quasigroup existence problem. Later, Fujita, Slaney, Stickel, McCune, and H. Zhang used their systems to solve several open cases and reported very competitive results. More recently, completing partial quasigroups has been proposed as a structured benchmark domain for the study of constraint satisfaction methods [9]. In addition to classifying structures within an algebraic axiomatisation, automated theorem proving has been used to find new axiomatisations for algebras, thus enabling better intra-classification of algebras. In particular, new axiomatic representations of groups have been found [10, 11].

In [5], we have presented a bootstrapping algorithm that enables the fully verified qualitative classification of algebras up to isomorphism. The algorithm starts with only the basic axioms of a particular algebraic structure, successively computes properties to separate non-isomorphic structures, and returns a set of unique distinguishing properties for all isomorphism classes. As a simple example, given the axioms of group theory and the cardinality 6, our algorithm returns the following (paraphrased) result: "all groups of size 6 belong to either of two isomorphism classes where one contains all Abelian and the other all non-Abelian groups". The algorithm itself incorporates diverse reasoning techniques by employing state of the art systems; in particular it uses model generation, machine learning, computer algebra, and automated theorem proving to obtain its results. For instance, it incorporates the first-order prover Spass [24] to verify the results of the classification. One of the problems that needs to be proved is to show that a particular set of properties uniquely defines an isomorphism class for particular algebraic structures of a given cardinality. When conducting experiments with the algorithm, the theorem proving part turned out to be the main bottleneck. Although Spass was the only first-order prover that solved a significant number of problems for our domain, its application is still very limited. For instance, Spass failed to solve all necessary problems for the verification of the classification of relatively small structures such as quasigroups of cardinality 5. Since our classification is concerned with finite algebras, we considered SAT solvers as substitute proof engines. In this article we present a variety of approaches to an efficient encoding of quasigroup classification problems for several types of SAT solvers and their experimental com-

parison. The main contribution of our work is an extension of work presented in [26] on the encoding of quasigroup problems in propositional logic and the development of three alternative approaches to encode isomorphisms between quasigroups.

In detail, Sec. 2 presents an overview of our bootstrapping algorithm that has already been described in [5] concentrating mainly on the aspects that are important for this article.

In Sec. 3 we present a formalisation of our problems in propositional logic that builds on aspects of the work done by Zhang in [26]. In particular, we use Zhang's techniques for eliminating universal quantifications over finite domains and for encoding simple equations and inequations as propositional variables. We extend his work by two alternative approaches to deal with existential quantifications as well as by considering equations that contain nested operator applications. We furthermore adapt the clause normalisation procedure of Nonnengart and Weidenbach [17] to produce small clausal normal forms that are suitable both for pure SAT solvers as well as for solvers with built-in equational theories.

When constructing classification theorems for quasigroups the most challenging problems are concerned with showing that all structures with certain properties are isomorphic. In Sec. 4 we present three encodings for isomorphisms inside satisfiability problems we have developed: a naïve way of enumerating all isomorphisms, and two more refined approaches that take advantage of computer algebra computations to reduce the number of isomorphism by considering generating systems for the structures involved. These two approaches are particularly well suited for the domain of quasigroups.

We have tested our approaches by experimenting with three different SAT solvers: (1) zChaff [16], a Boolean SAT solver combining the Davis-Putnam procedure with Boolean constraint propagation; (2) CVClite [3], a validity checker that accepts full first order formulas with equality as input but that reasons on propositional problems with an efficient internal SAT solver; (3) DPLLT [7], a satisfiability solver with built-in decision procedures and equational theory that accepts ground clauses with equations[1]. The results of these experiments — given in detail in Sec. 5 — show not only that employing satisfiability checking instead of theorem proving can greatly improve the power of our classification algorithm but also that the more elaborate isomorphism encodings significantly increase the solvability horizon of the single solvers.

[1] Note that the DPLLT system we use in our experiments is just one instance of the general DPLLT engine based on the very first implementation of this principle.

2. Problem Domain

The problem domain for classification in finite algebra has been introduced in detail in [5]. Here we only briefly present the problem of generating classification theorems in finite algebra and illustrate it with a concrete example, to which we shall refer throughout this article. We then sketch the bootstrapping algorithm that we have designed to solve the problem and focus, in particular, on the proof problems that occur during the bootstrapping procedure.

2.1. Classification Problems in Finite Algebra

General classification problems in algebra can be defined with respect to any equivalence relation given on a class of algebras. In this article we restrict ourselves to the isomorphism relation on algebras. We define the classification problem as follows: let $\mathfrak{A}$ be a finite collection of algebraic structures and let $\cong$ be the isomorphism equivalence relation on $\mathfrak{A}$. Then $\cong$ induces a partition into equivalence classes $[A_1]_\cong, [A_2]_\cong, \ldots, [A_n]_\cong$, where $A_i \in \mathfrak{A}$ for $i = 1, \ldots, n$. We call the $[A_i]_\cong$ *isomorphism classes* and A_i a *representant* for the isomorphism class $[A_i]_\cong$. Thus an isomorphism class is a collection of algebras that are all isomorphic to each other.

Let P be a property which is invariant under isomorphism. Then P acts as a *discriminant* for any two structures A and B in $\mathfrak{A}$, in the sense that if $P(A)$ and $\neg P(B)$, then $A \ncong B$. If, in addition, P holds for every element of an isomorphism class $[A_i]_\cong$, but does not hold for any element in $\mathfrak{A} \setminus [A_i]_\cong$, then we call P a *classifying property* for $[A_i]_\cong$. A full set of classifying properties — with one property for each isomorphism class — comprises a classifying theorem stating that each element of $\mathfrak{A}$ exhibits exactly one of the classifying properties. The classification problem is therefore to find a full set of classifying properties.

Although our approach to solve the classification problem is general, in this article we restrict the class of algebras to quasigroups and loops. We call a non-empty set Q together with a closed binary operation $\circ : Q \times Q \to Q$ a quasigroup if, for each $(a, b) \in Q \times Q$, there is a unique $(x, y) \in Q \times Q$, so that $a \circ x = y \circ a = b$. In other words, for each pair of elements in Q there exist uniquely determined left and right divisors and therefore the property is sometimes also called *unique solvability* [18]. Moreover, for finite structures, the property guarantees that in the multiplication table of Q each element occurs exactly once in each row and each column. This property is also known as the *Latin square property*. We call a quasigroup L a loop if it contains a unit

Q_1	a	b	c
a	b	a	c
b	a	c	b
c	c	b	a

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

Q_3	a	b	c
a	a	b	c
b	b	c	a
c	c	a	b

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

Q_5	a	b	c
a	a	c	b
b	b	a	c
c	c	b	a

Figure 1. Isomorphism class representants of quasigroups of order 3.

element, i.e., there exists a $unit \in L$ such that for each $x \in L$ we have $x \circ unit = unit \circ x = x$. Note that in the general case neither quasigroups nor loops are associative. In case they are associative they are groups.

An example of quasigroups of order 3 is given in Fig. 1 in terms of multiplication tables for the respective operations $\circ$. In fact the five quasigroups Q_1 to Q_5 are representants of the five isomorphism classes for quasigroups of order 3. The classification problem is then to find five properties, one for each Q_1 to Q_5, that uniquely characterise each isomorphism class and moreover to show that these five quasigroups indeed represent all possible isomorphism classes.

2.2. Bootstrapping Algorithm

The bootstrapping algorithm to generate classification theorems takes a set of properties $\mathcal{P}$ and a cardinality n as input. It returns a decision tree that contains the classification theorem for the algebraic structures of order n that satisfy $\mathcal{P}$, as well as a set of representants for each isomorphism class.

The algorithm itself works as follows: Given a set of properties $\mathcal{P}$ and a cardinality n it initialises a decision tree with the root node $\mathcal{N}$ labelled with the properties $\mathcal{P}$. We denote the properties a node is labelled by $\mathcal{P}_\mathcal{N}$. The algorithm then constructs an example of an algebraic structure of order n satisfying $\mathcal{P}_\mathcal{N}$. In case no example can be produced, the algorithm will show that indeed no structure of size n with properties $\mathcal{P}_\mathcal{N}$ can exist. In case an example exists, the algorithm does either of the following two things: (1) It shows that the node represents an isomorphism class, i.e., it proves that all structures of order n that satisfy the properties $\mathcal{P}_\mathcal{N}$ are isomorphic to each other, or, (2) it constructs another algebraic structure satisfying $\mathcal{P}_\mathcal{N}$, which is not isomorphic to the first one.

In case (2) the algorithm computes discriminating properties for the two structures. Either it computes one discriminating property P such

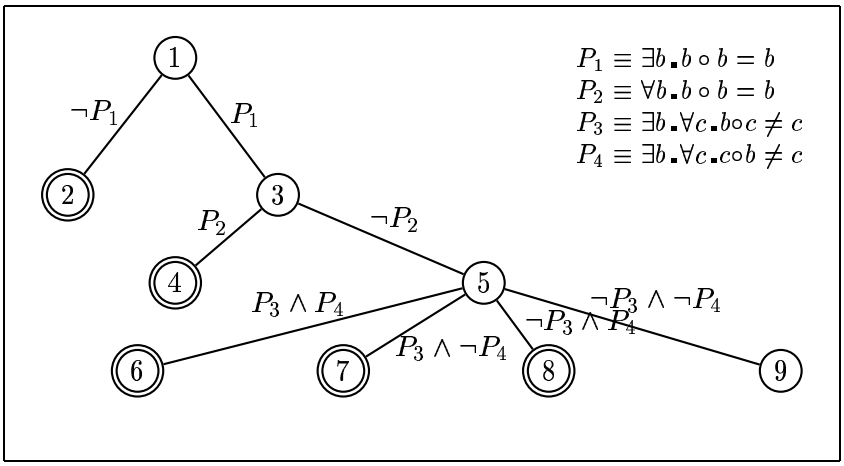

Figure 2. Decision tree for the classification problem of order 3 quasigroups.

that P holds for one structure and $\neg P$ holds for the other structure or it computes two discriminating properties P_1 and P_2, one for each structure. These properties are then used to further expand the decision tree: For one property P two new nodes $\mathcal{N}'$ and $\mathcal{N}''$ are added, 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. For two properties, P_1 and P_2, four new nodes have to be created: one for each of the possible combinations of discriminants, i.e., $P_1 \wedge P_2$, $\neg P_1 \wedge P_2$, $P_1 \wedge \neg P_2$, and $\neg P_1 \wedge \neg P_2$.

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. The leaf nodes then either represent isomorphism classes or are empty, i.e., no structure exists with the properties given in the node. We generally call the former *isoclass* nodes and the latter *dead-end* nodes and we shall refer to all non-leaf nodes as *branching* nodes. The final classification theorem corresponds then to the disjunction of the properties given as labels of the isoclass nodes.

An example of a fully constructed decision tree is given in Fig. 2. The tree represents the classification theorem for quasigroups of order 3. The leaves 2,4,6,7, and 8 are isoclass nodes, whereas leaf 9 is a dead-end node. The representants of the isoclass nodes correspond, from left to right to the quasigroups given in Fig. 1 (i.e., Q_1 is the representant of node 2, and so on). To preserve space, the properties have been denoted at the edges rather than at the vertices in the tree. Thus, the properties of a node correspond to the conjunction of the properties given on a path from the vertex to the root. In addition the basic properties of quasigroups have been omitted.

The bootstrapping algorithm itself only coordinates the constructions of the decision tree, while the occurring challenging deductive problems are outsourced to specialised systems:

- model generators (Finder [19], Sem [29], and Mace [13]) to generate example structures in each node,
- the machine learning program HR [4] to find discriminants, and
- automated theorem proving to solve the occurring proof problems.

2.3. Proof Problems

The decision tree constructed by the bootstrapping algorithm represents a classification theorem, and is thus the mathematical result of the classification process. For instance, the tree in Fig. 2 represents the classification theorem that there are 5 isomorphism classes of quasigroups of order 3, which can be uniquely described by the properties associated with the isoclass nodes. The proof of the overall classification theorem is done stepwise by showing the correctness of the decision tree in each step of its construction. The proof problems resulting from these correctness checks are therefore essentially artefacts of our bootstrapping approach. In this article we concern ourselves mainly with these proof problems and describe them in more detail in the remainder of this section. The two subsequent sections discuss their encoding in propositional logic.

The proof problems can be roughly divided into two categories:

1. Checking the correctness of computations in branching nodes.
2. Establishing properties of the final classification theorem.

Problems of type 1 are mainly concerned with verifying computations from systems external to the bootstrapping algorithm, i.e., model generation and machine learning. They are not strictly necessary for the construction of the decision tree, but are required if we want to generate a fully verified classification theorem.

Let A, A_1, A_2 be algebraic structures, let P, P_1, P_2 be properties, and let $\mathcal{P}$ be the algebraic properties given as input to the bootstrapping algorithm, then we can formulate the following theorems that need to be proved during the decision tree construction:

1. **Representant Theorem:** A satisfies the properties P and $\mathcal{P}$: we write formally $P(A) \wedge \mathcal{P}(A)$.

2. **Non-Isomorphic Theorem:** A_1 and A_2, both satisfying P and $\mathcal{P}$, are not isomorphic: $A_1 \ncong A_2$

3. **Discriminant Theorem:** P is a discriminant, i.e., if P holds for one algebra but does not hold for another algebra, then the two algebras are not isomorphic: $\forall A_1, A_2 . P(A_1) \wedge \neg P(A_2) \rightarrow A_1 \ncong A_2$.

4. **Isomorphism-Class Theorem:** All algebras of cardinality n that satisfy P and $\mathcal{P}$ are isomorphic: $\forall A_1, A_2 .[\mathcal{P}(A_1) \wedge \mathcal{P}(A_2) \wedge P(A_1) \wedge P(A_2)] \rightarrow A_1 \cong A_2$.

5. **Dead-End Theorem:** No algebra of cardinality n satisfies P and $\mathcal{P}$: $\neg \exists A . \mathcal{P}(A) \wedge P(A)$.

Theorems 1–3 belong to the first of the above categories, while theorems 4 and 5 belong to the second. In detail, the representant and non-isomorphic theorems verify the model generation: The former checks that a constructed model has indeed the desired properties. The latter verifies that two models are indeed non-isomorphic and thus guarantees against the construction of too many isomorphism classes. Theorem 3 verifies that the constructed property is indeed a valid discriminant.

Finally, the Isomorphism-Class and the Dead-End Theorem are used to establish that the algorithm has reached a leaf of the decision tree. The Isomorphism-Class Theorem verifies that all algebras satisfying the property given by the node's label form an isomorphism class. The Dead-End Theorem on the other hand establishes that a leaf node is indeed empty. Both theorems can also be regarded as a verification of failed model generation attempts, either to generate a non-isomorphic model or to construct a model at all.

As an example we examine the theorems proved in the first steps of the construction of the decision tree for quasigroups of order 3 given in Fig. 2. In node 1 the only property to consider is the unique solvability property for quasigroups. Q_1 from Fig. 1 is generated as representant that satisfies this property. This leads to the representant theorem:

$$[\forall a,b \in Q_1 . \exists x,y \in Q_1 . a \circ x = b \wedge y \circ a = b]. \tag{1}$$

Then Q_2 is generated as a second model, not isomorphic to Q_1, which is shown with the non-isomorphic theorem

$$Q_1 \ncong Q_2. \tag{2}$$

Given Q_1 and Q_2 the algorithm constructs the discriminant P_1, which is verified with the discriminant theorem and leads to the construction of two child nodes 2 and 3.

$$\forall A_1, A_2 .[\exists b \in A_1 . b \circ b = b] \wedge [\neg \exists b \in A_2 . b \circ b = b] \rightarrow A_1 \ncong A_2. \tag{3}$$

In node 2 no further non-isomorphic model can be constructed, but instead we can show that all structures with the properties $\mathcal{P}$ and $\neg P_1$ are isomorphic to each other. Since with Q_1 we already have a representant of the isomorphism class, for which we have shown that the properties hold, it suffices to prove that all structures satisfying these properties are isomorphic to Q_1:

$$\begin{aligned} \forall A. \quad & [[\forall a,b \in A. \exists x,y \in A. a \circ x = b \wedge y \circ a = b] \\ \wedge \quad & [\neg \exists b \in A. b \circ b = b]] \qquad \rightarrow Q_1 \cong A. \end{aligned} \tag{4}$$

Finally, in order to give also an example of a Dead-End Theorem we have to shift our attention to node 9 in the decision tree for quasigroups of order 3. The assertion to prove is the following lengthy theorem:

$$\begin{aligned} \neg \exists A. \quad & [\forall a,b \in A. \exists x,y \in A. a \circ x = b \wedge y \circ a = b] \\ \wedge \quad & [\exists b \in A. b \circ b = b] \wedge [\neg \forall b \in A. b \circ b = b] \\ \wedge \quad & [\neg \exists b \in A. \forall c \in A. b \circ c \neq c] \wedge [\neg \exists b \in A. \forall c \in A. c \circ b \neq c] \end{aligned} \tag{5}$$

Although the formulation of some of the above theorems is second order, all theorems can be formulated in propositional logic and passed to a SAT solver, since we work in a finite domain. We omit a transformation to propositional logic for discriminant theorems, since these theorems hold in general and not only for algebras of a certain size. Indeed in our experiments in [5] the first order theorem prover Spass proved those theorems generally without problems.

The representant and non-isomorphic theorems are relatively simple to prove, even for structures of large cardinality, since they make statements about concrete structures (e.g. Q_1 and Q_2 in our example). Spass was also successful on these theorems in the experiments described in [5] but struggled and quite often failed to show Isomorphism-Class Theorems and Dead-End Theorems for quasigroups of order greater than 4 and loops of order greater than 5.[2] We shall therefore concentrate on theorems 4 and 5 in the remainder of this article.

3. Specifying Theorems in Propositional Logic

In this section, we present an encoding of the theorems given in the previous section in propositional logic. While basic ideas for the encoding are taken from Zhang's article on the specification of Latin

[2] In fact, Spass was the only prover that managed to show any of the Isomorphism-Class and Dead-End Theorems for structures of order greater than 4, which is documented by the results of the 2004 CASC system competition [21], where these theorems were given as new entries for the TPTP [22].

square existence problems in propositional logic [26] we extended his work in order to deal with more general and complex properties. We first discuss Zhang's approach and then generalise it for our domain. Finally, we explain the generation of different input formats for the different SAT solvers under consideration.

3.1. An Encoding of Latin Square Problems

In [26], Zhang describes the encoding of Latin square existence problems in propositional logic. That is, his problems are concerned with the question of whether a Latin square A of a certain cardinality n exists that satisfies particular properties $P_1, \ldots, P_l$, where the properties P_i are of the form

$$\forall x_1, \ldots, x_n \in A . x_{1_1} \circ x_{1_2} = x_{1_3} \wedge \ldots \wedge x_{l_1} \circ x_{l_2} = x_{l_3} \rightarrow x_{r_1} \circ x_{r_2} = x_{r_3}$$

or

$$\forall x_1, \ldots, x_n \in A . x_{1_1} \circ x_{1_2} = x_{1_3} \wedge \ldots \wedge x_{l_1} \circ x_{l_2} = x_{l_3} \rightarrow x_{r_1} = x_{r_2}$$

with $x_{i_1}, x_{i_2}, x_{i_3} \in \{x_1, \ldots, x_n\}$ for $i = 1, \ldots, l, r$. Thus, the properties are restricted to the use of universal quantifiers only. They consist of simple equations without nested application of the operation $\circ$ and also always have only a simple element on the right hand side of the equations. In addition Zhang expresses the Latin square property in this form by

$$\forall x, y, u, w \in A . x \circ u = y \wedge x \circ w = y \rightarrow u = w$$
$$\forall x, y, u, w \in A . u \circ x = y \wedge w \circ x = y \rightarrow u = w.$$

This formulation is equivalent to our mathematical formalisation of the quasigroup property for finite cardinalities.

To specify a Latin square existence problem of the above type in propositional logic Zhang uses the following transformation steps:

1. The second-order quantifier over a structure A is replaced by using an arbitrary structure of cardinality n, $A = \{e_1, \ldots, e_n\}$ together with a binary operation $\circ$, where the e_i are new constants. In order to make sure that the structure is indeed of order n the n elements are explicitly stated to be distinct by adding the assumption

   $$e_1 \neq e_2 \wedge e_1 \neq e_3 \wedge \ldots \wedge e_{n-1} \neq e_n.$$

   Moreover, for each constant an instance of the reflexivity axiom is added: $e_1 = e_1 \wedge \ldots \wedge e_n = e_n$.

2. To the list of required properties $P_1, \ldots, P_l$, which include the formalisation of the Latin square property, the following properties are explicitly added:

$$\forall x, y, u, w \in A . x \circ y = u \wedge x \circ y = w \rightarrow u = w$$
$$\forall x, y \in A . x \circ y = e_1 \vee \ldots \vee x \circ y = e_n.$$

The former property is called the unique image property whereas the latter one is called the closure property. Note that these additional properties also have only universal quantifiers and consist of simple equations without nested $\circ$ operations.

3. All ground instances of the properties $P_1, \ldots, P_l$ and the additional properties with the constants $e_1, \ldots, e_n$ of the instantiation of the algebra A are constructed.

This creates clauses with ground equality, which can already be passed to a satisfiability solver with a built-in equational theory. To obtain a purely propositional logic formula the following additional step is necessary:

4. Replace each ground equation $e_i \circ e_j = e_k$ by a Boolean variable p_{ijk} with $i, j, k = 1 \ldots n$ and replace each occurrence of $e_i = e_k$ by a Boolean variable q_{ij} with $i, j = 1 \ldots n$.

This results in a Boolean satisfiability problem containing altogether $n^3 + n^2$ Boolean variables.

In contrast to the properties and formulas considered by Zhang the properties constructed during our classification procedure can be of a more complex nature. In particular, they can contain

- (nested) universal and existential quantifiers,
- terms containing nested $\circ$ operations,
- complex terms, containing applications of $\circ$, on the right hand sides of equations, and
- further interpreted symbols such as a special unit element *unit*.

While we can directly adopt the first two steps of Zhang's transformations for our properties, we need to extend steps three and four in order to cope with the above complications. Moreover, as opposed to the properties considered by Zhang, our properties do not necessarily result immediately in clausal normal form after transformation to propositional logic. This makes it necessary to also explicitly consider efficient methods of clause normalisation.

3.2. Elimination of Quantifiers

In our properties, the first order quantifiers ranging over elements of the instantiated algebra $A = \{e_1, \ldots, e_n\}$ can be both universal and existential. Moreover, the quantifiers can be arbitrarily nested. In order to eliminate the quantifiers to obtain ground instances we have the choice between two different procedures, which differ wrt. the handling of the existential quantifiers.

The first approach replaces existentially quantified formulas by disjunctions over the finite set of elements. Universally quantified formulas are replaced by conjunctions over the finite set of elements. That is, for a given property all quantified sub-formulas in P are processed recursively: An existentially quantified formula $\exists x \in A . F[x]$ results in a disjunction of the instantiations, $F[e_1] \vee \ldots \vee F[e_n]$, whereas a universally quantified formula $\forall x \in A . F[x]$ results in a conjunction of the instantiations, $F[e_1] \wedge \ldots \wedge F[e_n]$.

We illustrate the extended quantifier elimination procedure by its application to property $P_3 = \exists b \in A . \forall c \in A . b \circ c \neq c$ from the decision tree in Fig. 2, where $A = \{e_1, e_2, e_3\}$. The procedure first tackles the outside existential quantifier of P_3, which results in the disjunction:

$$\forall c \in A . e_1 \circ c \neq c \vee \forall c \in A . e_2 \circ c \neq c \vee \forall c \in A . e_3 \circ c \neq c.$$

Afterwards, the three resulting universal quantifiers are tackled, yielding the following fully grounded formula:

$$\begin{aligned} & (e_1 \circ e_1 \neq e_1 \wedge e_1 \circ e_2 \neq e_2 \wedge e_1 \circ e_3 \neq e_3) \\ \vee\ & (e_2 \circ e_1 \neq e_1 \wedge e_2 \circ e_2 \neq e_2 \wedge e_2 \circ e_3 \neq e_3) \\ \vee\ & (e_3 \circ e_1 \neq e_1 \wedge e_3 \circ e_2 \neq e_2 \wedge e_3 \circ e_3 \neq e_3). \end{aligned}$$

A second, alternative approach to deal with existential quantifiers is Skolemisation. The Skolemisation transformation consists of three steps: (1) push all negations to the literals, (2) replace existential quantifiers by Skolem-functions, and (3) add formulas expressing the closure of the introduced Skolem-functions. The result is a formula, which contains only universal quantifiers, which can be replaced by conjunctions as described above.[3]

In the example of property P_3, Skolemisation yields

$$[\forall c \in A . sk_b \circ c \neq c] \wedge [sk_b = e_1 \vee sk_b = e_2 \vee sk_b = e_3]$$

[3] The described procedure introduces Skolem-functions whose arity depends on the number of universal quantifiers in whose range the original existential quantifier was. An alternative is the recursive intertwined elimination of universal quantifiers and Skolemisation such that only Skolem-constants, i.e., 0-arity functions, have to be introduced.

where sk_b is the Skolem-function introduced for the variable b. The elimination of the universal quantifier then yields:

$$\begin{aligned} & [sk_b \circ e_1 \neq e_1 \wedge sk_b \circ e_2 \neq e_2 \wedge sk_b \circ e_3 \neq e_3] \\ \wedge\ & [sk_b = e_1 \vee sk_b = e_2 \vee sk_b = e_3]. \end{aligned}$$

The use of Skolemisation instead of a disjunctive treatment of existential quantifiers generally results in formulas with fewer and less-nested disjunctions and conjunctions. This is beneficial for the clause normalisation described in Sec. 3.5 and is indeed helpful for SAT solvers with equational theory such as CVClite (see the results of our experiments in Sec. 5). However, Skolemisation is counterproductive for purely Boolean satisfiability solvers such as zChaff for reasons we will explain in Sec. 3.4.

3.3. Flattening

Equations containing terms with nested applications of the operation $\circ$ as well as occurrences of $\circ$ operations on right hand sides of equations can be simplified by a process, which we call *flattening*. The idea of flattening is to recursively replace subexpressions like $x \circ y$ by new existentially quantified variables r with $x \circ y = r$ until all occurring equations are of the form $x \circ y = z$.

We demonstrate the flattening procedure with the property of associativity $\forall a,b,c \in A.(a \circ (b \circ c)) = ((a \circ b) \circ c)$. The following step-wise transformation fully flattens the occurring equations:

$$\begin{aligned} & \forall a,b,c \in A.(a \circ (b \circ c))=((a \circ b) \circ c) \\ \hookrightarrow\ & \forall a,b,c \in A.\exists z_1 \in A.(b \circ c=z_1)\wedge(a \circ z_1)=((a \circ b) \circ c) \\ \hookrightarrow\ & \forall a,b,c \in A.\exists z_1,z_2 \in A.(b \circ c=z_1)\wedge(a \circ b=z_2)\wedge(a \circ z_1)=(z_2 \circ c) \\ \hookrightarrow\ & \forall a,b,c \in A.\exists z_1,z_2,z_3 \in A.(b \circ c=z_1)\wedge(a \circ b=z_2)\wedge(z_2 \circ c=z_3)\wedge(a \circ z_1=z_3) \end{aligned}$$

Quantifiers introduced by flattening can then be eliminated as described in Sec. 3.2.

3.4. Axiomatisation of Equational Theory

For systems, such as zChaff, that provide no built-in equational theory, the necessary equational theory has to be stated explicitly. Zhang's approach already contains the encoding of a basic equational theory: step 1 adds the reflexivity of the new constants e_i as well as the condition that they are pairwise distinct. Step 2 introduces, furthermore, the unique image property. These axioms suffice in the general case, which only deals with fully quantified formulas without interpreted symbols. In the case of interpreted symbols, however, we have to enrich

the formalisation by axioms explicitly specifying further properties of equality with respect to the given symbol.

For instance, if we are dealing with loops we have to axiomatise equality for the unit element *unit* explicitly, by adding the following axioms to our problem formalisation:

$$\forall x,y,z \in A . (x \circ y = z) \wedge z = unit \rightarrow (x \circ y = unit) \quad \text{(Substitution 1)}$$
$$\forall x,y,z \in A . (x \circ y = z) \wedge y = unit \rightarrow (x \circ unit = z) \quad \text{(Substitution 2)}$$
$$\forall x,y,z \in A . (x \circ y = z) \wedge x = unit \rightarrow (unit \circ y = z) \quad \text{(Substitution 3)}$$
$$\forall x,y,z \in A . (x \circ y = unit) \wedge z = unit \rightarrow (x \circ y = z) \quad \text{(Substitution 4)}$$
$$\forall x,y,z \in A . (x \circ unit = z) \wedge y = unit \rightarrow (x \circ y = z) \quad \text{(Substitution 5)}$$
$$\forall x,y,z \in A . (unit \circ y = z) \wedge x = unit \rightarrow (x \circ y = z) \quad \text{(Substitution 6)}$$
$$unit = unit \quad \text{(Reflexivity)}$$
$$\forall x \in A . unit = x \Leftrightarrow x = unit \quad \text{(Symmetry)}$$
$$\forall x,y \in A . unit = x \wedge unit = y \rightarrow x = y \quad \text{(Transitivity)}$$

With the introduction of the *unit* symbol, ground equations of the form $e_i \circ e_j = unit$, $e_i \circ unit = e_j$, $unit \circ e_i = e_j$ for $i,j = 1 \ldots n$ and $unit = e_i$ and $e_i = unit$ for $i = 1 \ldots n$ and $unit = unit$ can be created. Hence, to replace the equations by Boolean variables additional corresponding variables are necessary. Altogether these are $3n^2+2n+1$ additional Boolean variables.

In consequence, introducing interpreted symbols adds considerably to the complexity of the theorem. This increase in complexity is the reason why the Skolemisation approach described in Sec. 3.2 is not feasible to construct Boolean satisfiability problems since each new Skolem-function would have to be fully axiomatised as above. On the other hand, since introduced interpreted symbols can be used to construct discriminants, they can improve the bootstrapping algorithm by allowing for further discriminants (see Sec. 5 for an example of a discriminant with *unit*). However, since the discriminants computed in the bootstrapping algorithm never introduce new interpreted symbols, we restrict the algorithm to the symbols explicitly given in the domain theory, which in our case is only *unit*.

3.5. Constructing Clausal Normal Forms

The properties we are dealing with can contain an arbitrary composition of logical connectives and quantifiers. This means that the quantifier elimination of nested universal and existential quantifiers can result in lengthy and nested disjunctions and conjunctions of equational literals. In other words our formulas — contrary to the restricted properties considered in [26] — are generally not in clausal normal form

after quantifier elimination. Hence, when working with systems that require clausal normal form as input, such as zChaff or DPLLT, we have to explicitly perform clause normalisation. Because of the nested disjunctions and conjunctions of equational literals, a naïve clause normalisation approach would suffer from a combinatorial explosion of the number and the length of the resulting clauses. For our implementation, we adopted clause normalisation techniques from [17] that aim to create *small clausal normal forms*. The basic technique is the introduction of additional Boolean variables to suitably *break formulas*. This avoids combinatorial explosion but extends the theorem formalisation by additional propositional variables.

For instance, consider the formula $F_1 \vee F_2$. A naïve clause normalisation would compute the clause set $\mathcal{C}_1$ for F_1 and the clause set $\mathcal{C}_2$ for F_2. The clause set of $F_1 \vee F_2$ is then the Cartesian product $\mathcal{C}_1 \times \mathcal{C}_2$. The following introduction of the new Boolean variable p, $(F_1 \vee p) \wedge (\neg p \vee F_2)$, is satisfiability preserving but results in a different set of clauses: If $\mathcal{C}_1'$ and $\mathcal{C}_2'$ are the clause sets of $F_1 \vee p$ and $\neg p \vee F_2$, respectively, then the clause set of $(F_1 \vee p) \wedge (\neg p \vee F_2)$ is the union $\mathcal{C}_1' \cup \mathcal{C}_2'$.

We can further simplify clause normalisation by using Skolemisation as discussed in Sec. 3.2, which reduces the complexity of the formulas resulting from the quantifier elimination. However, as discussed in the previous subsection, Skolemisation is inappropriate for use with purely Boolean satisfiability solvers.

3.6. Generating Different Input Formats

Since the aim of our formalisation is to produce input for different types of SAT solvers we conclude the section with a brief description of the transformations required to produce the different input formats we need. The three systems we are concerned with — CVClite, DPLLT, and zChaff — accept ground first-order formulas with equality, ground first-order clauses with equality, and purely Boolean formulas in clausal normal form, respectively. Therefore, all three systems require the transformation step 1 from Sec. 3.1, which replaces the quantifiers over algebras A by arbitrary instances of the required cardinality. All three systems also require the explicit statement of the cardinality of A, i.e., that all elements of the instance of A are distinct, which corresponds to the first part of step 2 in Sec. 3.1. From there on, however, CVClite and DPLLT strongly differ from zChaff with respect to which further transformations are necessary.

Both CVClite and DPLLT have built-in equational theory and accept ground first order terms as input. Thus they neither require flattening, nor axiomatisation of the equational theory, nor the replace-

ment of ground equations by Boolean variables. They only require quantifier elimination applied to the properties. For the elimination of existentially quantified variables, both alternative treatments are possible since the equational theory for Skolem-functions does not have to be explicitly axiomatised. While DPLLT needs its input in clausal form, and we therefore have to perform clause normalisation, this step can be omitted for CVClite, as it accepts full formulas as input.

zChaff accepts as input Boolean satisfiability problems in the DIMACS format, which requires clauses. Hence, to create input for zChaff the full set of transformations has to be applied in the following order: flattening, axiomatisation of equational theory for additional symbols, quantifier elimination (without the option of Skolemisation), replacement of ground equations by Boolean variables, and clause normalisation.

4. Dealing with Isomorphisms

The formalisation discussed so far can essentially deal with all properties on quasigroups that are potentially constructed during the classification process. However, it is not yet sufficient to fully deal with the most challenging problems of our domain, the Isomorphism-Class Theorems. As an example of such a problem, consider again the theorem for node 2 in the decision tree in Fig. 2 already given in formula (4):

$$\forall A.[[\forall a,b \in A.\exists x,y \in A.a \circ x = b \wedge y \circ a = b] \wedge [\neg\exists b \in A.b \circ b = b]] \rightarrow Q_1 \cong A$$

The theorem states that all structures of order 3 satisfying the quasigroup property and the property $\neg P_1$ given in Fig. 2 are isomorphic to the structure Q_1 (see Fig. 1).

Following the transformations introduced in the preceding section we can start rewriting the theorem by eliminating the universal quantifier on A with an arbitrary set $A = \{e_1, e_2, e_3\}$ and adding an explicit encoding of the representant structure Q_1 as additional assumption. This results in the assumptions

$e_1 \neq e_2 \wedge e_1 \neq e_2 \wedge e_2 \neq e_3$	(Cardinality of A)
$\forall a,b \in A.\exists x,y \in A.a \circ x = b \wedge y \circ a = b$	(Quasigroup property)
$\neg\exists b \in A.b \circ b = b$	(Classifying property $\neg P_1$)
$e'_1 \circ' e'_1 = e'_2 \wedge e'_1 \circ' e'_2 = e'_1 \wedge e'_1 \circ' e'_3 = e'_3$	
$\wedge\ e'_2 \circ' e'_1 = e'_1 \wedge e'_2 \circ' e'_2 = e'_3 \wedge e'_2 \circ' e'_3 = e'_2$	(Representant Q_1)
$\wedge\ e'_3 \circ' e'_1 = e'_3 \wedge e'_3 \circ' e'_2 = e'_2 \wedge e'_3 \circ' e'_3 = e'_1$	

and a conclusion of the form

$$\exists h \in \{Q_1 \mapsto A\}.bijective(h) \wedge homomorphism(h).$$

Observe that, unlike in Fig. 2, we have denoted the elements of Q_1 as $\{e'_1, e'_2, e'_3\}$ to avoid confusion with the elements introduced for A. In a similar manner, $\circ'$ denotes the operation on Q_1, in order to distinguish it from $\circ$, the operation on A. Note also, that the conclusion now states explicitly that there has to exist a mapping h from the representant Q_1 to A that is a bijective homomorphism.

While we can transform all the assumptions into propositional logic using the techniques discussed in Sec. 3, translating the conclusion is not that straight forward. We have developed three encodings for isomorphisms inside satisfiability problems that we describe in this section. We first present the naïve approach of enumerating all possible isomorphisms, followed by two more refined approaches that take advantage of computer algebra computations to reduce the number of isomorphisms.

4.1. Naïve Approach

The conclusion of the Isomorphism-Class Theorem existentially quantifies over a mapping h from Q_1 to A. Since the structures involved are finite we can eliminate the quantifier by considering all possible mappings between the two structures. In the general case of structures with cardinality n, there are n^n possible mappings. We can, however, reduce this number immediately by taking into account that we only have to consider bijective mappings in the first place, which leaves $n!$ possible mappings. For our example theorem there are 6 possible bijections $h_1, \ldots, h_6$ from Q_1 to A:

$$\begin{array}{ll} h_1(e'_1) = e_1, h_1(e'_2) = e_2, h_1(e'_3) = e_3 & h_2(e'_1) = e_1, h_2(e'_2) = e_3, h_2(e'_3) = e_2 \\ h_3(e'_1) = e_2, h_3(e'_2) = e_1, h_3(e'_3) = e_3 & h_4(e'_1) = e_2, h_4(e'_2) = e_3, h_4(e'_3) = e_1 \\ h_5(e'_1) = e_3, h_5(e'_2) = e_1, h_5(e'_3) = e_2 & h_6(e'_1) = e_3, h_6(e'_2) = e_2, h_6(e'_3) = e_1. \end{array}$$

Given these 6 functions the original conclusion can be replaced by

$$\begin{aligned} & \mathit{homomorphism}(h_1) \vee \mathit{homomorphism}(h_2) \vee \mathit{homomorphism}(h_3) \\ \vee\ & \mathit{homomorphism}(h_4) \vee \mathit{homomorphism}(h_5) \vee \mathit{homomorphism}(h_6). \end{aligned}$$

Here $\mathit{homomorphism}(h_i)$ for $i = 1, \ldots, 6$ is an abbreviation for the homomorphism property $\forall x,y \in Q_1 . h_i(x \circ' y) = h_i(x) \circ h_i(y)$. Note that we can omit the bijective property for each h_i, since they are bijective by construction. The quantified variables x and y ranging over Q_1 can be eliminated as explained in Sec. 3.2, which results in the following conjunction of equations:

$$\begin{aligned} & h_i(e'_1 \circ' e'_1) = h_i(e'_1) \circ h_i(e'_1) \wedge h_i(e'_1 \circ' e'_2) = h_i(e'_1) \circ h_i(e'_2) \wedge h_i(e'_1 \circ' e'_3) = h_i(e'_1) \circ h_i(e'_3) \\ \wedge\ & h_i(e'_2 \circ' e'_1) = h_i(e'_2) \circ h_i(e'_1) \wedge h_i(e'_2 \circ' e'_2) = h_i(e'_2) \circ h_i(e'_2) \wedge h_i(e'_2 \circ' e'_3) = h_i(e'_2) \circ h_i(e'_3) \\ \wedge\ & h_i(e'_3 \circ' e'_1) = h_i(e'_3) \circ h_i(e'_1) \wedge h_i(e'_3 \circ' e'_2) = h_i(e'_3) \circ h_i(e'_2) \wedge h_i(e'_3 \circ' e'_3) = h_i(e'_3) \circ h_i(e'_3). \end{aligned}$$

Since the results of expressions such as $e'_j \circ' e'_k$ are given by the multiplication table of Q_1 we can simplify the left hand sides of the above equations to

$$\begin{aligned} & h_i(e'_2){=}h_i(e'_1) \circ h_i(e'_1) \wedge h_i(e'_1){=}h_i(e'_1) \circ h_i(e'_2) \wedge h_i(e'_3){=}h_i(e'_1) \circ h_i(e'_3) \\ \wedge\ & h_i(e'_1){=}h_i(e'_2) \circ h_i(e'_1) \wedge h_i(e'_3){=}h_i(e'_2) \circ h_i(e'_2) \wedge h_i(e'_2){=}h_i(e'_2) \circ h_i(e'_3) \\ \wedge\ & h_i(e'_3){=}h_i(e'_3) \circ h_i(e'_1) \wedge h_i(e'_2){=}h_i(e'_3) \circ h_i(e'_2) \wedge h_i(e'_1){=}h_i(e'_3) \circ h_i(e'_3). \end{aligned}$$

Finally, for each h_i the expressions $h_i(e'_j)$ can be replaced by its image. For instance, in the case of h_1 this yields:

$$\begin{aligned} & e_2 = e_1 \circ e_1 \wedge e_1 = e_1 \circ e_2 \wedge e_3 = e_1 \circ e_3 \\ \wedge\ & e_1 = e_2 \circ e_1 \wedge e_3 = e_2 \circ e_2 \wedge e_2 = e_2 \circ e_3 \\ \wedge\ & e_3 = e_3 \circ e_1 \wedge e_2 = e_3 \circ e_2 \wedge e_1 = e_3 \circ e_3. \end{aligned}$$

Note that in the above formula there is no reference to the homomorphisms h_i anymore. And indeed the h_i were intermediate concepts only and the final conclusion no longer contains them anymore. Similarly, while the final conclusion describes the structures that are isomorphic to Q_1 there is no mention of the actual elements of Q_1 anymore. We can therefore also remove the encoding for the representant Q_1 from the assumptions.

Since the resulting formula contains only flat equations, it can be directly translated into a Boolean satisfiability problem. Thus the naïve approach is suitable for all the input formats we are interested in generating. It suffers, however, from combinatorial explosion. For structures of cardinality n there are $n!$ possible bijective mappings and each mapping finally results in a conjunction of n^2 equations $e_i \circ e_j = e_k$ with $i, j, k = 1 \ldots n$. Hence, altogether the conclusion results in $n!\, n^2$ equation literals.

4.2. Representant Generating Systems

In order to reduce the complexity of the conclusion for the Isomorphism-Class Theorems we have developed two alternative encodings, which generally result in smaller encodings. Both are based on the computation of sets of generators and factorisations to decrease the number of potential isomorphism mappings. A structure A with binary operation $\circ$ is said to be *generated* by a set of elements $\{a_1, \ldots, a_m\} \subseteq A$ if every element of A can be expressed as a combination — usually called a factorisation or word — of the a_i under the operation $\circ$. For example, Q_1 in Fig. 1, can be generated by element $e'_1 \in Q_1$, as both $e'_2 = e'_1 \circ' e'_1$ and $e'_3 = (e'_1 \circ' e'_1) \circ' (e'_1 \circ' e'_1)$ can be expressed as factorisations in e'_1. 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 in order to reduce the total number of isomorphisms we need to consider. If we again consider our example theorem and the generating system for Q_1 we have only three potential mappings for the generator e'_1 to the elements of $A = \{e_1, e_2, e_3\}$, namely $h_1(e'_1) = e_1$, $h_2(e'_1) = e_2$, $h_3(e'_1) = e_3$. Taking the factorisations for e'_2 and e'_3 together with the homomorphism property the mappings can be completed as follows:

$$\begin{aligned} h_1\colon\quad h_1(e'_2) &= h_1(e'_1 \circ' e'_1) = h_1(e'_1) \circ h_1(e'_1) = e_1 \circ e_1 \\ h_1(e'_3) &= h_1((e'_1 \circ' e'_1) \circ' (e'_1 \circ' e'_1)) = (h_1(e'_1) \circ h_1(e'_1)) \circ (h_1(e'_1) \circ h_1(e'_1)) \\ &= (e_1 \circ e_1) \circ (e_1 \circ e_1). \end{aligned}$$

For h_2 and h_3 we get the analogous result, where e_1 is replaced by e_2 and e_3, respectively. Taking these 3 potential mappings the original conclusion of the Isomorphism-Class Theorem can be replaced by the disjunction:

$$\begin{aligned} & (homomorphism(h_1) \wedge bijective(h_1)) \\ \vee\ & (homomorphism(h_2) \wedge bijective(h_2)) \\ \vee\ & (homomorphism(h_3) \wedge bijective(h_3)). \end{aligned}$$

Here it is necessary to show bijectivity for each mapping, since it is no longer guaranteed by the construction. Naturally, it suffices to prove injectivity as the mapping is between finite structures. In other words we have to add that $h_i(e'_j) \neq h_i(e'_k)$ for all $j \neq k$, or in the concrete case of h_1 we add: $h_1(e'_1) \neq h_1(e'_2) \wedge h_1(e'_1) \neq h_1(e'_3) \wedge h_1(e'_2) \neq h_1(e'_3)$. These inequalities together with the grounded homomorphism properties can then be simplified analogously to the naïve approach in Sec. 3.2, i.e., we simplify the left hand sides of the equations and replace all occurrences of $h_1(e'_j)$ by their respective images. This eventually yields the following lengthy conjunction for h_1, which has already been simplified by removing redundant conjuncts:

$$\begin{aligned} & e_1 = e_1 \circ (e_1 \circ e_1) \wedge (e_1 \circ e_1) \circ (e_1 \circ e_1) = e_1 \circ (e_1 \circ e_1) \circ (e_1 \circ e_1) \\ \wedge\ & e_1 = (e_1 \circ e_1) \circ e_1 \wedge (e_1 \circ e_1) = (e_1 \circ e_1) \circ ((e_1 \circ e_1) \circ (e_1 \circ e_1)) \\ \wedge\ & (e_1 \circ e_1) \circ (e_1 \circ e_1) = (e_1 \circ e_1) \circ (e_1 \circ e_1) \circ e_1 \\ \wedge\ & e_1 = ((e_1 \circ e_1) \circ (e_1 \circ e_1)) \circ ((e_1 \circ e_1) \circ (e_1 \circ e_1)) \\ \wedge\ & e_1 \neq (e_1 \circ e_1) \wedge e_1 \neq (e_1 \circ e_1) \circ (e_1 \circ e_1) \wedge (e_1 \circ e_1) \neq (e_1 \circ e_1) \circ (e_1 \circ e_1). \end{aligned}$$

We have implemented an algorithm to compute a minimal generating system for a given structure in the computer algebra system Gap [8] (see [5] for more details on the algorithm). Calls to the algorithm are integrated into the overall bootstrapping algorithm, which employs it to compute generating systems for the representants of potential

isomorphism classes. Once a generating system is computed, the bootstrapping algorithm verifies its correctness by showing an additional theorem that simply checks that the representant in question actually complies with the generating system. We call this theorem *Representant Gensys-Verification Theorem*, but since it is fairly easy to check we will not go into details here.

Employing the verified generating system of the representant can reduce the number of mappings that are candidates for isomorphisms. 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, since only the m generators have to be mapped explicitly. However, this reduction is only effective when we produce input for solvers such as CVClite and DPLLT which can deal with the complex terms on both sides of the ground equations. For the generation of a purely Boolean encoding for a SAT solver like zChaff, flattening of the equations (see Sec. 3.3) is required. This, however, introduces new quantifiers, which have to be eliminated again later on. For instance, to flatten the conjunct above we need two additional quantified variables x_1 and x_2, which replace $x_1 = e_1 \circ e_1$ and $x_2 = x_1 \circ x_1 = (e_1 \circ e_1) \circ (e_1 \circ e_1)$ and whose scope is the complete conjunct. Their subsequent elimination would result in a disjunction with 9 parts, which are the different instantiations of the variables x_1 and x_2 in the conjunct. Indeed, we found that the factor by which the encoding is enlarged is related to the number m of generators in the computed generating system. When there are m generators, then there are $n - m$ factorised elements. These $n - m$ factorised elements result in $n - m$ different terms in the ground equations, which require $n - m$ variables for flattening. The elimination of these $n - m$ variables leads to a disjunction with $(n-m)^n$ parts, i.e., flattening and quantifier elimination enlarges the encoding by a factor of $(n - m)^n$. This factor clearly outweighs the benefits of the reduction of isomorphisms and indeed experiments confirmed that zChaff's performance was worse for this encoding as opposed to the naïve approach. Thus, employing representant generating systems is not suitable when creating Boolean satisfiability problems.

The use of generating systems is particularly suitable for structures like quasigroups. In our experiments the algorithm could generally come up with generating systems of at most 2 generators, even for quasigroup structures of cardinality 7. This subsequently led to at most $n(n-1)$ possible mappings as opposed to the $n!$ mappings created by the naïve approach. For algebraic structures that tend to have large generating systems, this approach might be counterproductive. As an example, consider a semi-group A whose operation maps every pair of inputs x, y to one element c only. Its only generating system is $A - \{c\}$, which

does not decrease the number of possible mappings to consider, but introduces the additional burden of proving the *Representant Gensys-Verification Theorem*. Thus, in theory the computation and usage of generating systems is generally applicable, however, its practical impact is limited depending on the type of structures considered.

4.3. General Generating Systems

We will now generalise our notion of generating systems to further simplify Isomorphism-Class Theorems. The idea is based on the observation that generating systems are invariants under isomorphism. That is, isomorphic structures have similar generating systems.

We can exploit this fact in our context as follows: In order to verify that a node in the classification tree represents an isomorphism class, we first show that every structure satisfying the properties of the node also has a generating system similar to the one for the representant of the node.[4] We call this the *General Gensys-Verification Theorem* and, having successfully proved it, we can express the Isomorphism-Class Theorem using the general generating system.

Let's consider again our representant Q_1 together with its generating system consisting of e'_1 as generator and factorisations $e'_2 = e'_1 \circ' e'_1$ and $e'_3 = (e'_1 \circ' e'_1) \circ (e'_1 \circ' e'_1)$. From the proof of the General Gensys-Verification Theorem we know that all structures of the form $A = \{e_1, e_2, e_3\}$ with operation $\circ$ that exhibit the properties given by the node 2 in the decision tree contain a similar generating system. Without loss of generality we can fix it as follows: Let e_1 be the generator and let $e_2 = e_1 \circ e_1$ and $e_3 = (e_1 \circ e_1) \circ (e_1 \circ e_1)$. Since an isomorphism is determined by its actions on the generators, we only have to consider the 3 possible mappings of the single generator. However, as opposed to the two preceding approaches, this time we consider the possible mappings from A to the representant Q_1:

$$\begin{array}{lll} h_1(e_1) = e'_1 & h_1(e_2) = e'_1 \circ' e'_1 & h_1(e_3) = (e'_1 \circ' e'_1) \circ' (e'_1 \circ' e'_1) \\ h_2(e_1) = e'_2 & h_2(e_2) = e'_2 \circ' e'_2 & h_2(e_3) = (e'_2 \circ' e'_2) \circ' (e'_2 \circ' e'_2) \\ h_3(e_1) = e'_3 & h_3(e_2) = e'_3 \circ' e'_3 & h_3(e_3) = (e'_3 \circ' e'_3) \circ' (e'_3 \circ' e'_3). \end{array}$$

We can now replace the right hand sides of the equations by the values determined by the multiplication table for Q_1, resulting in:

$$\begin{array}{lll} h_1(e_1) = e'_1 & h_1(e_2) = e'_2 & h_1(e_3) = e'_3 \\ h_2(e_1) = e'_2 & h_2(e_2) = e'_3 & h_2(e_3) = e'_1 \\ h_3(e_1) = e'_3 & h_3(e_2) = e'_1 & h_3(e_3) = e'_2. \end{array}$$

[4] Clearly, if the node does not represent an isomorphism class, this does not have to be the case.

We then remove all mappings that are not bijective. While in our example all mappings are bijective, when dealing with structures of larger cardinality this step often reduces the number of mappings considerably. For the remaining mappings it suffices to show that one of them is a homomorphism to assure that there is indeed an isomorphism between A and Q_1. In other words the Isomorphism-Class Theorem is replaced by the disjunction:

$$homomorphism(h_1) \vee homomorphism(h_2) \vee homomorphism(h_3)$$

As in the previous two approaches we now replace each occurrence of $homomorphism(h_i)$ for $i = 1 \ldots 3$ with the actual homomorphism property, eliminate the quantifiers and simplify as far as possible. In the case of the mapping h_1 this yields the following conjunction of equations:

$$\begin{aligned} & h_1(e_1 \circ e_1) = e'_2 \wedge h_1(e_1 \circ e_2) = e'_1 \wedge h_1(e_1 \circ e_3) = e'_3 \\ \wedge\ & h_1(e_2 \circ e_1) = e'_1 \wedge h_1(e_2 \circ e_2) = e'_3 \wedge h_1(e_2 \circ e_3) = e'_2 \\ \wedge\ & h_1(e_3 \circ e_1) = e'_3 \wedge h_1(e_3 \circ e_2) = e'_2 \wedge h_1(e_3 \circ e_3) = e'_1. \end{aligned}$$

Since the resulting equations are of the form $h_1(e_i \circ e_j) = e'_k$, the left hand sides $e_i \circ e_j$ cannot be simplified. However, we can exploit the fact that h_1 is bijective and that the pre-image for each e'_k is uniquely determined by $h_1(e_l) = e'_k$. Hence, we can replace the right hand sides of the equations and then drop the function application of h_1:

$$\begin{aligned} & e_1 \circ e_1 = e_2 \wedge e_1 \circ e_2 = e_1 \wedge e_1 \circ e_3 = e_3 \\ \wedge\ & e_2 \circ e_1 = e_1 \wedge e_2 \circ e_2 = e_3 \wedge e_2 \circ e_3 = e_2 \\ \wedge\ & e_3 \circ e_1 = e_3 \wedge e_3 \circ e_2 = e_2 \wedge e_3 \circ e_3 = e_1. \end{aligned}$$

This final formalisation of the isomorphism only contains expressions of the form $e_i \circ e_j = e_k$, which are *a priori* flat and are therefore suitable for the translation into Boolean SAT problems as well. And, as there are at most $\frac{n!}{(n-m)!}$ — in most cases even fewer — possible mappings to consider, where m is the number of generators, the complexity is generally better than in either of the previous approaches. However, some of the complexity of general generating systems is actually hidden in the proof of the General Gensys-Verification Theorem, which we have ignored so far.

In the case of our example the theorem states that any structure A satisfying the quasigroup property and $\neg P_1$ has the same generating system as the representant Q_1. The conclusion of the theorem is therefore

$$\exists x_1,x_2,x_3 \in A.(x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3) \wedge (x_2 = x_1 \circ x_1) \wedge (x_3 = (x_1 \circ x_1) \circ (x_1 \circ x_1)).$$

We can now rearrange and shrink the equations by using the fact that $x_2 = x_1 \circ x_1$ holds, and therefore $(x_1 \circ x_1)$ can be replaced by x_2 in the equation for x_3. The formula then has the form

$$\exists x_1,x_2,x_3 \in A.(x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3) \wedge (x_1 \circ x_1 = x_2 \wedge x_2 \circ x_2 = x_3).$$

For the expansion of the quantifiers we can exploit the information from the inequalities to immediately eliminate inconsistent instantiations, which gives us the following 6 cases:

$$\begin{aligned} & [e_1 \circ e_1 = e_2 \wedge e_2 \circ e_2 = e_3] \vee [e_1 \circ e_1 = e_3 \wedge e_3 \circ e_3 = e_2] \\ \vee\ & [e_2 \circ e_2 = e_1 \wedge e_1 \circ e_1 = e_3] \vee [e_2 \circ e_2 = e_1 \wedge e_1 \circ e_1 = e_3] \\ \vee\ & [e_3 \circ e_3 = e_1 \wedge e_1 \circ e_1 = e_2] \vee [e_3 \circ e_3 = e_2 \wedge e_2 \circ e_2 = e_1]. \end{aligned}$$

Our computer algebra algorithm always returns generating systems that can be shrunk such that we always have a fully flat formalisation. This makes formalisation of the General Gensys-Verification Theorem well suited for purely Boolean SAT solvers. Both formalisation of the General Gensys-Verification Theorem as well as the naïve isoclass transformation consist of $n!$ cases. However, the complexity of the former is generally better since a case consists of a conjunction of $n-m$ equations of the form $e_i \circ e_j = e_k$, where $n - m$ is the number of factorisations. On the contrary the cases for the naïve transformation consist of n^2 equations of that form. While this means that, when using general generating systems, the General Gensys-Verification Theorem is the bottleneck, in practice the approach still behaves better than the naïve approach, as described in the next section.

5. Experiments and Results

In order to test the usefulness of the different encodings we have developed, we have conducted a number of experiments. We were particularly interested in the following three main questions:

How do the different systems compare? We are interested in comparing the performance of CVClite, DPLLT, and zChaff in order to see if the additional effort to transform our theorems from full formulas to clausal normal form, and further to Boolean satisfiability problems is justified. And since our original motivation to develop encodings for SAT solvers was the limitation of first order automated theorem provers in our domain we are also interested in comparing the performance of the SAT approach to the first order theorem prover Spass.

How do the different encodings compare? Here we want to test how useful the elaborate encodings for Isomorphism-Class Theorems including the computation of generating systems as opposed to the

naïve encoding are. Moreover, we want to test if the Skolemisation of existential variables has an advantage over the disjunctive existential quantifier elimination for the systems CVClite and DPLLT.

How does our approach scale up? This question can be investigated along two dimensions: On the one hand the problem size increases with increasing cardinality of the structures. On the other hand, within the same domain, problems become increasingly difficult as more and more properties are added during the classification.

In the remainder of this section we describe the general experimental setup and then discuss the results. The tables containing the actual results are given in the Appendix.

5.1. Experimental Setup

In our experiments we used the solvers DPLLT, CVClite, and zChaff. The input was prepared in the format of the respective systems as described in full in Sec. 3.6. For a comparison with our experiments in [5] we also used Spass, where Spass received the same specification as input as CVClite.

We applied the systems to satisfiability problems from the following classification trees:

1. Quasigroups of order 5 (abbreviated as Q5) containing 1283 isoclass nodes, 66 dead-end nodes and 1327 branching nodes.

2. Loops of order 6 (Loops6): 109 isoclass nodes, 18 dead-end nodes, and 106 branching nodes.

3. Quasigroups of order 6 (Q6+) with additional property $\exists x . \forall y . (y \circ x) \circ (x \circ y) = x$: 13 isoclass nodes, 2 dead-end nodes and 17 branching nodes.[5]

4. Quasigroups of order 7 (Q7-qg9) with the additional property QG9 $\forall x . \forall y . (((y \circ x) \circ x) \circ x) = y$: 7 isoclass and 55 branching nodes.

Except for Loops6, the above classification trees are intermediate decision trees, since the classifications were still running when we started the experiments with the SAT solvers. Hence, the number of isomorphism classes used in the experiments in this article is smaller than the actual number. Meanwhile, we have completed the classification of Q5 and Q6+. While we have also completed other classifications, for instance, the QG3 – QG8 quasigroups of order 6 and 7, the resulting decision trees were too small to conduct meaningful experiments. The

[5] The property is a generalised form of the QG3 property $\forall x . \forall y . (y \circ x) \circ (x \circ y) = x$.

classifications above, on the other hand, provide a large number of challenging problems.

To answer our three main questions we conducted a number of experiments with different settings. The experiments can roughly be divided into (1) *main experiments* applying the systems to the problems of all four classification trees wrt. a common encoding and (2) *additional experiments* applying the systems wrt. alternative encodings to problems from selected classification trees.

(1) In the main experiments we applied zChaff, DPLLT, and CVClite to all problems of the four classification trees and Spass to the Loops6 and Q6+ problems. As common encoding we took one that is suitable for all systems, namely the non-Skolemised version of the Isomorphism-Class Theorem formalisation with general generating systems from Sec. 4.3. The results of these experiments are given in the Tables III–V in the Appendix.

For the nodes in the classification trees, we generated and checked the following problems: for dead-end nodes the Dead-End Theorems (abbreviated by Deadend-Th in the result tables), for isoclass nodes the general Gensys-Verification Theorems (Gensys-Th) and the Isomorphism-Class Theorems (Iso-Th). For branching nodes either the General Gensys-Verification Theorem or the Isomorphism-Class Theorem does not hold. Hence, there are both General Gensys-Verification Theorems (BrGsys-Th) and non-theorems (BrGsys-NTh) as well as Isomorphism-Class Theorems (BrIso-Th) and non-theorems (BrIso-NTh).

(2) For the comparison of different encodings we applied DPLLT and CVClite to the Skolemised problems from Q6+ and Q7-qg9. The results are given in Table VI and Table VII. Moreover, we also applied DPLLT, zChaff, and CVClite to problems from Q6+ and Q7-qg9 using different formalisations for the Isomorphism-Class Theorems; these results are given in the Tables VIII–X. In these experiments, DPLLT and CVClite were applied to both the formalisation using no generating system, i.e., using the naïve isomorphism encoding described in Sec. 4.1 (indicated by Withoutgensys in the result tables) and the formalisation with representant generating system as introduced in Sec. 4.2 (Withrepgensys). zChaff was applied to the Withoutgensys problems only. For DPLLT and CVClite we also compared the Skolemised and non-Skolemised versions of these problems.

All experiments were conducted on a cluster of 140 identical Pentium IV machines, each with 1 GB of main memory, using SUN GridEngine to distribute the experiments. We ran the systems in a mode that would not record proof objects or traces in a file. For each single problem in each problem suite the systems got a time limit of 5 days pure CPU time and a 512MB memory limit. While this seems to make for a very

long overall time for the experiments considering the large number of problems and the relatively high failure rate in the experiments, this can be relativised by the fact that the majority of failed runs were when experimenting with CVClite, which generally failed due to reaching the memory limit after just a few hours.

The Tables III–X in the appendix detail the results of the experiments. They are structured as follows: The first three columns state the problem domain and the name and number of theorems or non-theorems considered. The subsequent four columns give timing information in seconds; the minimum and maximum time needed to successfully solve a problem for the considered problem suite, the average run time taken only over successful runs in the problem suite and finally the median run time taken over all runs in the problem suite, including the failed runs that received the full five day runtime (i.e., 432000 seconds) as penalty. Finally the last column gives the number of problems a system failed to solve for the considered problem set.

5.2. Results

Comparison of Systems: When comparing the systems's performances in the main experiments (Tables III–V) we can observe that zChaff was, on average, the fastest system. DPLLT occasionally outperformed zChaff in minimum run-time (i.e., the fastest solution to a problem of a given category) but was slower overall. Finally CVClite and Spass clearly performed worse, in particular considering that they failed to solve a substantial number of problems. Thus, the results indicate that zChaff shows the best performance in our domain, followed, with some distance, by DPLLT. Hence, the encoding of Boolean satisfiability problems for zChaff pays off to push the solvability horizon in our domain. Moreover, the SAT solvers zChaff and DPLLT clearly outperform the first-order theorem prover Spass.

Comparison of Encodings: When looking at different formalisations of the Isomorphism-Class Theorems we have to first consider which theorems we have to compare. For the formalisation with general generating systems we are required to prove the Isomorphism-Class Theorems as well as the corresponding general Gensys-Verification Theorems, where the latter are clearly more difficult. However, for the other two formalisations, the naïve and the representant generating system formalisation, the Isomorphism-Class Theorem is shown independently and essentially contains the complexity of the problem. Hence we have to compare the performance of the systems on the Iso-Th problems for these two formalisations with the Gensys-Th problems of the main experiments.

For the naïve formalisation, DPLLT and CVClite performed worse than for the other two formalisations: they failed for more problems and for the problems they solved they needed considerably more time. This is only partially true for zChaff. For the Q6+ problems zChaff's performance was on average exactly as good for the naïve formalisation as for the formalisation with general generating systems. For the Q7-qg9 problems it performs on average worse for the naïve formalisation. This indicates that the disadvantages of the naïve formalisation will have more impact for higher cardinalities.

For DPLLT the formalisation with representant generating systems clearly outperforms the formalisation with general generating system wrt. to runtime. However, for Q7-qg9, DPLLT failed for more Iso-Th problems of this formalisation. Since CVClite fails for almost all Iso-Th and Gensys-Th problems of Q6+ and Q7-qg9, a substantiated analysis of its performance is not possible.

Overall these results indicate that the elaborate Isomorphism-Class Theorem formalisations based on the computation of generating systems outperform the naïve formalisation. A comparison of the formalisation with representant generating systems as opposed to the formalisation with general generating systems, which is possible for DPLLT and CVClite only, shows no clear result.

In order to test the impact of Skolemisation, CVClite and DPLLT were applied to Skolemised problems of Q6+ and Q7-qg9. The results are quite different for the two systems, see Table VII and VI. On the one hand, CVClite can solve considerably more problems with Skolemisation than without and is also faster. On the other hand, DPLLT fails on more Skolemised problems and it needs more time for the problems it solves. The same behaviour can also be observed for Skolemised and non-Skolemised versions of different Isomorphism-Class Theorem formalisations. Hence, there is no clear result for the impact of Skolemisation for the employed SAT solvers in our domain.

Scalability: The tables in the Appendix do not give a clear scalability result wrt. to run-times and solved problems such as "the higher the cardinality of the classification tree the more difficult the problems, i.e., the longer the SAT solvers take and the less problems can be solved". Indeed, none of the three SAT solvers shows a definite increase in maximal, average, and median run-times or in the number or percentage of failed attempts with increasing cardinality of the tackled problems.

Instead we can observe a generally very high variance within the set of problems of a classification tree. For instance, consider the minimal runtime of less than one second as opposed to the maximal runtime 266359 seconds for DPLLT on Q7-qg9 Gensys-Th problems (see Table III). There are essentially two reasons for this large variance:

Table I. Characteristic numbers for Gensys-Th problems in zChaff input.

Problem Class	Cardinality	depth of node	#variables	#clauses	time
Q5	5	1	150	2055	< 1
Q5	5	23	352	4531	132
Q6+	6	1	257	5925	286
Q6+	6	7	405	6975	4877
Q7-qg9	7	1	2401	20594	< 1
Q7-qg9	7	8	4044	24907	6345

(1) Some properties introduced during the classification are particularly well suited for the search procedures of the systems, for instance, the idempotency property $\forall x . x \circ x = x$. This effect can also be observed by a comparison of the average results of Loops6 and Q6+. Although both classifications are concerned with quasigroups of order 6 the performance of the systems on the problems of these two classes varies considerably. Whereas the unit property prunes the search rather well, the special property of Q6+ turned out to be particularly difficult.

(2) The deeper a node is in the classification tree, the more properties it is associated with. For instance, in the Q5 tree there are isoclass nodes at depth 5 associated with 5 additional properties and at depth 23 with 23 additional properties. And indeed showing theorems for the latter takes considerably longer than for the former. Moreover, the properties can become more complex than in our example classification in Fig. 2, adding to the complexity of the resulting problems. As examples consider the following properties from the Q5 and Loops6 trees:

$$\exists b . \exists c . (c \circ c = b) \wedge (b \circ b = c) \wedge (b \circ c = unit) \wedge (c \circ b = unit) \wedge b \neq unit$$
$$\exists b . \exists c . \exists d . (b \circ c = b) \wedge (b \circ b \neq c) \wedge (d \circ d = b)$$
$$\forall b . \exists c . \exists e . (c \circ b = e \circ e) \wedge (c \circ (e \circ e) = b)$$

To illustrate the impact of the properties on the complexity of the resulting satisfiability problems, consider the figures characterising Gensys-Th problems in zChaff input in Table I. Both the number of Boolean variables and the number of clauses of the satisfiability problems increase for larger cardinalities *and* for nodes deeper in the decision tree. With the increase in complexity the time necessary to solve the problems also increases. However, we can again observe that there is not necessarily an increase in time needed wrt. to the cardinality of the structures involved.

6. Conclusion

We have presented the application of satisfiability solving in the challenging problem of classification in finite algebra. We have extended existing approaches to encode quasigroup existence problems for SAT solvers in order to deal with the more complex properties of our domain. Our developed techniques are not restricted to our problem domain but are applicable in the general case of transforming equality problems over finite domains to Boolean satisfiability problems.

The most challenging problem for our particular domain was to efficiently encode isomorphism problems by reducing the number of possible isomorphisms that have to be considered. We solved this by developing two formalisations employing the concept of generating systems that significantly improve over a naïve encoding and that are particularly effective in our domain of quasigroups. This enables us to substitute the first-order theorem provers so far used in our bootstrapping algorithm for constructing classification theorems by SAT solvers. The developed encodings are not geared towards only one particular type of solver but can be used to produce several input formats, which enables us to employ and experiment with diverse systems, such as zChaff, DPLLT, and CVClite.

The results of our experiments lead us to three conclusions. (1) SAT solvers can successfully extend the solvability horizon of our bootstrapping algorithm, i.e., they clearly outperformed the first-order theorem prover Spass. Indeed, employing SAT solvers instead of Spass has led to new mathematical classification results such as a full classification theorem for quasigroups of order 5. (2) Moreover, the developed elaborate formalisations of isomorphism problems also help to push the solvability horizon of the bootstrapping algorithm even further, since for the classification of quasigroups the SAT solvers clearly perform better for the elaborate formalisations than for a naïve formalisation. (3) Overall, the results indicate that zChaff shows the best performance, followed, with some distance, by DPLLT. Hence, the encoding in the less intuitive input format of zChaff pays off in our domain. Another advantage of zChaff as opposed to DPLLT is that it creates a proof trace output for unsatisfiable problems, which can be checked as a resolution proof by independent proof checkers. This is an important issue considering that we are interested in fully verifiable classification theorems.

Future work could include investigating the use of further solvers, for instance one with integrated computations for specialised mathematical domains [2]. Besides classification wrt. isomorphism it is also worthwhile to consider other equivalence relations. For instance, in terms

of quantitative classifications for quasigroups and loops representatives for every isomorphism and isotopy class have been generated up to order 10 [14]. Investigating isotopism classes is even more interesting from a mathematical viewpoint than isomorphism classes, however, it might present an even greater challenge from the automated reasoning side; firstly, because finding appropriate properties strong enough to discriminate structures with respect to isotopy presents a hard problem for HR, and secondly, an easy transfer of our techniques to reduce the number of mappings between structures to the case of showing isotopy class theorems is not obvious.

Acknowledgments

We would like to thank Simon Colton and Roy McCasland for their cooperation on the classification of quasigroups, and Albert Oliveras Llunell, Zhaohui Fu, Clark Barrett, and Thomas Hillenbrand for their support in using the systems DPLLT, zChaff, CVClite, and Spass, respectively. We also wish to thank the anonymous reviewers for their detailed and extremely helpful comments.

References

1. Alur, R. and D. Peled (eds.): 2004, 'Proc. of Computer Aided Verification, 16^{th} International Conference, CAV 2004', Vol. 3114 of *LNCS*. Springer.
2. Audemard, G., P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani: 2002, 'Integrating Boolean and Mathematical Solving: Foundations, Basic Algorithms and Requirements'. In: *Proc. of CALCULEMUS-2002*, Vol. 2385 of *LNAI*.
3. Barrett, C. and S. Berezin: 2004, 'CVC Lite: A New Implementation of the Cooperating Validity Checker'. in [1], pp. 515–518.
4. Colton, S.: 2002, 'The HR program for theorem generation'. in [23].
5. Colton, S., A. Meier, V. Sorge, and R. McCasland: 2004, 'Automatic Generation of Classification Theorems for Finite Algebras'. In: *Proc. of IJCAR 2004*, Vol. 3097 of *LNAI*. pp. 400–414. Springer.
6. Fujita, M., J. Slaney, and F. Bennett: 1993, 'Automatic Generation of Some Results in Finite Algebra'. In: *Proc. IJCAI-13*. pp. 52–57.
7. Ganzinger, H., G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli: 2004, 'DPLL(T): Fast Decision Procedures'. in [1], pp. 175–188.
8. Gap: 2000, *GAP Reference Manual*. The GAP Group, School of Mathematical and Computational Sciences, University of St. Andrews.
9. Gomes, C. P., B. Selman, N. Crato, and H. Kautz: 2000, 'Heavy-tailed Phenomena in Satisfiability and Constraint Satisfaction Problems'. *Journal of Automated Reasoning* **24**, 67–100.
10. Kunen, K.: 1992, 'Single Axioms for Groups'. *Journal of Automated Reasoning* **9**(3), 291–308.
11. McCune, W.: 1993, 'Single Axioms for Groups and Abelian Groups with Various Operations'. *Journal of Automated Reasoning* **10(1)**, 1–13.

12. McCune, W.: 1994, 'A Davis-Putnam program and its application to finite first-order model search: quasigroup existence problems'. Technical report, Argonne National Laboratory, Division of MSC.
13. McCune, W.: 2003, 'Mace4 Reference Manual and Guide'. Argonne National Laboratory. ANL/MCS-TM-264.
14. McKay, B. D., A. Meinert, and W. Myrvold: 2002, 'Counting Small Latin Squares'. In: *European Women in Mathematics Int. Workshop on Groups and Graphs*. pp. 67–72.
15. McKay, B. D. and I. M. Wanless, 'The number of Latin squares of order eleven'. Submitted for publication. Available at `http://cs.anu.edu.au/~bdm/papers/ls11.pdf`.
16. Moskewicz, M., C. Madigan, Y. Zhao, L. Zhang, and S. Malik: 2001, 'Chaff: Engineering an efficient SAT Solver'. In: *Proc. of the Design Automation Conference*. pp. 530–535.
17. Nonnengart, A. and C. Weidenbach: 2001, 'Computing Small Clause Normal Forms'. In: *Handbook of Automated Reasoning*. Elsevier.
18. Pflugfelder, H. O.: 1990, *Quasigroups and Loops: Introduction*, Vol. 7 of *Sigma Series in Pure Mathematics*. Heldermann Verlag.
19. Slaney, J.: 1995, 'FINDER, Notes and Guide'. Center for Information Science Research, Australian National University.
20. Slaney, J., M. Fujita, and M. E. Stickel: 1995, 'Automated Reasoning and Exhaustive Search: Quasigroup Existence Problems'. *Computers and Mathematics with Applications* **29**, 115–132.
21. Sutcliffe, G.: 2005, 'The IJCAR-2004 Automated Theorem Proving Competition'. *AI Communications* **18**(1), 33–40.
22. Sutcliffe, G. and C. Suttner: 1998, 'The TPTP Problem Library: CNF Release v1.2.1'. *Journal of Automated Reasoning* **21**(2), 177–203.
23. Voronkov, A. (ed.): 2002, 'Proc. of the 18th International Conference on Automated Deduction (CADE–18)', Vol. 2392 of *LNAI*. Springer.
24. Weidenbach, C., U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic: 2002, 'SPASS Version 2.0'. in [23], pp. 275–279.
25. Zhang, H.: 1997a, 'SATO: An Efficient Propositional Prover'. In: *Proc. of CADE–14*, Vol. 1249 of *LNAI*. pp. 272–275.
26. Zhang, H.: 1997b, 'Specifying Latin squares in propositional logic'. In: *Automated Reasoning and Its Applications, Essays in honor of Larry Wos*. MIT Press.
27. Zhang, H., M. P. Bonacina, and J. Hsiang: 1996, 'PSATO: a Distributed Propositional Prover and its Application to Quasigroup Problems'. *Journal of Symbolic Computation* **21**, 543–560.
28. Zhang, H. and J. Hsiang: 1994, 'Solving Open Quasigroup Problems by Propositional Reasoning'. In: *Proc. of Int. Computer Symposium*. Hsinchu, Taiwan.
29. Zhang, J. and H. Zhang: 2001, 'SEM User's Guide'. Department of Computer Science, University of Iowa.

Appendix

Table II. Main experiments Spass.

Domain	Problem	#	Min	Max	Avg.	Median	Fail
Q6+	BrGsys-NTh	14	14	135450	20515	14251	0
	BrGsys-Th	3	–	–	–	432000	3
	BrIso-NTh	13	8	814	319	359	0
	BrIso-Th	4	534	22960	6732	1717	0
	Deadend-Th	2	–	–	–	432000	2
	Gensys-Th	13	96436	96436	96436	432000	12
	Iso-Th	13	22	190746	20315	475	1
Loops6	BrGsys-NTh	81	< 1	271606	3692	1	2
	BrGsys-Th	25	3	63898	10129	432000	17
	BrIso-NTh	77	< 1	186248	3732	2	6
	BrIso-Th	29	< 1	166906	10426	5	6
	Deadend-Th	18	27	94713	10058	5500	6
	Gensys-Th	109	< 1	100611	5904	432000	56
	Iso-Th	109	< 1	363181	26907	699	27

Table III. Main experiments DPLLT.

Domain	Problem	#	Min	Max	Avg.	Median	Fail
Q5	BrGsys-NTh	1079	< 1	45	1	< 1	0
	BrGsys-Th	248	< 1	310	29	15	0
	BrIso-NTh	872	< 1	< 1	< 1	< 1	0
	BrIso-Th	455	< 1	< 1	< 1	< 1	0
	Deadend-Th	66	< 1	216	25	14	0
	Gensys-Th	1283	< 1	1692	32	17	0
	Iso-Th	1283	< 1	1	< 1	< 1	0
Q6+	BrGsys-NTh	14	1	176	27	15	0
	BrGsys-Th	3	520	3924	2319	2515	0
	BrIso-NTh	13	< 1	3	1	< 1	0
	BrIso-Th	4	< 1	1	< 1	< 1	0
	Deadend-Th	2	578	1736	1157	1157	0
	Gensys-Th	13	219	5738	3345	3542	0
	Iso-Th	13	< 1	3	1	< 1	0
Loops6	BrGsys-NTh	81	< 1	21	1	< 1	0
	BrGsys-Th	25	< 1	27	7	5	0
	BrIso-NTh	79	< 1	1	< 1	< 1	0
	BrIso-Th	27	< 1	1	< 1	< 1	0
	Deadend-Th	18	< 1	14	3	1	0
	Gensys-Th	109	< 1	25	5	3	0
	Iso-Th	109	< 1	10	< 1	< 1	0
Q7-qg9	BrGsys-NTh	55	< 1	83	4	1	0
	BrIso-NTh	26	< 1	1	< 1	< 1	0
	BrIso-Th	29	< 1	2	< 1	< 1	0
	Gensys-Th	7	< 1	266359	44519	170	1
	Iso-Th	7	< 1	< 1	< 1	< 1	0

Table IV. Main experiments CVClite.

Domain	Problem	#	Min	Max	Avg.	Median	Fail
Q5	BrGsys-NTh	1079	< 1	20184	261	169	98
	BrGsys-Th	248	6	17634	1788	432000	128
	BrIso-NTh	872	< 1	393	11	4	2
	BrIso-Th	455	< 1	211	18	8	2
	Deadend-Th	66	< 1	392345	13907	34560	27
	Gensys-Th	1283	< 1	20462	670	432000	777
	Iso-Th	1283	< 1	806	20	8	0
Q6+	BrGsys-NTh	14	951	951	951	432000	13
	BrGsys-Th	3	–	–	–	432000	3
	BrIso-NTh	13	114	6136	2178	3712	4
	BrIso-Th	4	3256	3256	3256	432000	3
	Deadend-Th	2	–	–	–	432000	2
	Gensys-Th	13	–	–	–	432000	13
	Iso-Th	13	266	3464	1082	432000	7
Loops6	BrGsys-NTh	81	1	5131	378	604	29
	BrGsys-Th	25	15	13866	1974	432000	13
	BrIso-NTh	79	1	31156	701	102	4
	BrIso-Th	27	2	1711	330	127	0
	Deadend-Th	18	10	87957	21221	432000	10
	Gensys-Th	109	< 1	20776	1671	4022	45
	Iso-Th	109	8	3515	333	812	29
Q7-qg9	BrGsys-NTh	55	30	7831	785	913	19
	BrIso-NTh	26	12	22263	1412	58	0
	BrIso-Th	29	3	3292	216	107	6
	Gensys-Th	7	12	2485	1026	432000	4
	Iso-Th	7	2	64	23	16	0

Table V. Main experiments zChaff.

Domain	Problem	#	Min	Max	Avg.	Median	Fail
Q5	BrGsys-NTh	1079	< 1	4	< 1	< 1	0
	BrGsys-Th	248	< 1	97	12	6	0
	BrIso-NTh	872	< 1	< 1	< 1	< 1	0
	BrIso-Th	455	< 1	< 1	< 1	< 1	0
	Deadend-Th	66	< 1	75	11	5	0
	Gensys-Th	1283	< 1	132	13	7	0
	Iso-Th	1283	< 1	< 1	< 1	< 1	0
Q6+	BrGsys-NTh	14	< 1	7	2	2	0
	BrGsys-Th	3	1590	2966	2392	2619	0
	BrIso-NTh	13	< 1	5	2	2	0
	BrIso-Th	4	3	6	4	4	0
	Deadend-Th	2	1685	2182	1933	1933	0
	Gensys-Th	13	286	4877	2350	2387	0
	Iso-Th	13	< 1	12	5	5	0
Loops6	BrGsys-NTh	81	< 1	1	< 1	< 1	0
	BrGsys-Th	25	< 1	13	3	3	0
	BrIso-NTh	77	< 1	< 1	< 1	< 1	0
	BrIso-Th	29	< 1	< 1	< 1	< 1	0
	Deadend-Th	18	< 1	8	1	1	0
	Gensys-Th	109	< 1	11	2	2	0
	Iso-Th	109	< 1	< 1	< 1	< 1	0
Q7-qg9	BrGsys-NTh	55	< 1	1029	300	255	0
	BrIso-NTh	26	< 1	< 1	< 1	< 1	0
	BrIso-Th	29	< 1	2	< 1	< 1	0
	Gensys-Th	7	< 1	6345	2846	2604	0
	Iso-Th	7	< 1	< 1	< 1	< 1	0

Table VI. Skolemisation experiments DPLLT.

Domain	Problem	#	Min	Max	Avg.	Median	Fail
Q6+ Skolem	BrGsys-NTh	14	< 1	109	17	5	0
	BrGsys-Th	3	46692	82812	68019	74554	0
	BrIso-NTh	13	< 1	< 1	< 1	< 1	0
	BrIso-Th	4	< 1	< 1	< 1	< 1	0
	Deadend-Th	2	716	716	716	216358	1
	Gensys-Th	13	273	185391	94458	166908	4
	Iso-Th	13	< 1	4	1	< 1	0
Q7-qg9 Skolem	BrGsys-NTh	55	< 1	230	18	2	0
	BrIso-NTh	26	< 1	< 1	< 1	< 1	0
	BrIso-Th	29	< 1	3	< 1	< 1	0
	Gensys-Th	7	< 1	33254	11129	432000	4
	Iso-Th	7	< 1	< 1	< 1	< 1	0

Table VII. Skolemisation experiments CVClite.

Domain	Problem	#	Min	Max	Avg.	Median	Fail
Q6+ Skolem	BrGsys-NTh	14	9	414	104	62	0
	BrGsys-Th	3	–	–	–	432000	3
	BrIso-NTh	13	9	110	34	23	0
	BrIso-Th	4	21	51	31	26	0
	Deadend-Th	2	–	–	–	432000	2
	Gensys-Th	13	2828	5981	3764	5981	6
	Iso-Th	13	22	111	57	61	0
Q7-qg9 Skolem	BrGsys-NTh	55	49	4227	500	265	1
	BrIso-NTh	26	11	83	40	34	0
	BrIso-Th	29	19	1116	156	58	0
	Gensys-Th	7	13	624	318	432000	5
	Iso-Th	7	< 1	94	32	29	0

Table VIII. Different isomorphism formalisation experiments DPLLT.

Domain	Special	Problem	#	Min	Max	Avg.	Median	Fail
Q6+	Withoutgensys	BrIso-NTh	17	< 1	155	29	7	0
		Iso-Th	13	259	10712	3526	3412	1
	Withoutgensys +Skolem	BrIso-NTh	17	< 1	135	16	4	0
		Iso-Th	13	457	142264	89217	114334	3
	Withrepgensys	BrIso-NTh	17	1	386	145	139	0
		Iso-Th	13	1	308	103	2448	0
	Withrepgensys +Skolem	BrIso-NTh	17	1	308	174	143	1
		Iso-Th	13	131	84204	27069	19256	2
Q7-qg9	Withoutgensys	BrIso-NTh	20	–	–	–	432000	20
		Iso-Th	7	< 1	< 1	< 1	432000	5
	Withoutgensys +Skolem	BrIso-NTh	20	–	–	–	432000	20
		Iso-Th	7	–	–	–	432000	7
	Withrepgensys	BrIso-NTh	55	< 1	318	28	6	0
		Iso-Th	7	33	663	313	663	3
	Withrepgensys +Skolem	BrIso-NTh	55	< 1	204	32	9	0
		Iso-Th	7	183	211297	105740	432000	5

Table IX. Different isomorphism formalisation experiments CVClite.

Domain	Special	Problem	#	Min	Max	Avg.	Median	Fail
Q6+	Withoutgensys	BrIso-NTh	17	526	526	526	432000	16
		Iso-Th	13	–	–	–	432000	13
	Withoutgensys +Skolem	BrIso-NTh	17	13	506	87	42	0
		Iso-Th	13	2798	7172	3988	432000	8
	Withrepgensys	BrIso-NTh	17	767	767	767	432000	16
		Iso-Th	13	–	–	–	432000	13
	Withrepgensys +Skolem	BrIso-NTh	17	8	105	33	27	0
		Iso-Th	13	1639	6374	3883	432000	8
Q7-qg9	Withoutgensys	BrIso-NTh	20	58	805	232	291	6
		Iso-Th	7	17	1307	455	432000	4
	Withoutgensys +Skolem	BrIso-NTh	20	56	2700	437	187	0
		Iso-Th	7	45	610	327	432000	5
	Withrepgensys	BrIso-NTh	55	14	5726	623	471	20
		Iso-Th	7	42	2749	1395	432000	5
	Withrepgensys +Skolem	BrIso-NTh	55	21	2373	374	163	0
		Iso-Th	7	317	8900	3582	432000	4

Table X. Different isomorphism formalisation experiments zChaff.

Domain	Problem	#	Min	Max	Avg.	Median	Fail
Q6+ Withoutgensys	BrIso-NTh	17	< 1	4	1	< 1	0
	Iso-Th	13	476	3068	2092	2105	0
Q7-qg9 Withoutgensys	BrIso-NTh	55	< 1	14	2	1	0
	Iso-Th	7	341	44092	11264	6194	2