Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 125 (2005) 149–164
www.elsevier.com/locate/entcs

Reducing Symmetries to Generate Easier SAT Instances[1]

Jian Zhang, Zhuo Huang[2]

Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing 100080, China

Abstract

Finding countermodels is an effective way of disproving false conjectures. In first-order predicate logic, model finding is an undecidable problem. But if a finite model exists, it can be found by exhaustive search. The finite model generation problem in the first-order logic can also be translated to the satisfiability problem in the propositional logic. But a direct translation may not be very efficient. This paper discusses how to take the symmetries into account so as to make the resulting problem easier. A static method for adding constraints is presented, which can be thought of as an approximation of the least number heuristic (LNH). Also described is a dynamic method, which asks a model searcher like SEM to generate a set of partial models, and then gives each partial model to a propositional prover. The two methods are analyzed, and compared with each other.

Keywords: Finite model searching, SAT, symmetries, the least number heuristic

1 Introduction

Compared with theorem proving, the subject of disproving false conjectures has been less studied. But it is actually very important, since for open questions, you do not know whether the conjecture holds or not. If you give a false conjecture to a typical resolution-based theorem prover, the prover either runs forever or terminates without producing any useful information. When this

[1] Supported by the National Science Fund for Distinguished Young Scholars (No. 60125207) and K.C. Wong Education Foundation (Hong Kong).

[2] Email: zj@ios.ac.cn, hz@ios.ac.cn

doi:10.1016/j.entcs.2004.07.023

happens, you do not know whether it is because the conjecture is false or the inference rules are not enough or the prover is not so efficient.

An effective way of disproving such conjectures is to find a suitable countermodel, namely, a model of the axioms in which all the premises hold but the conjecture does not. However, for first-order predicate logic, determining the existence of models is an undecidable problem in general. Fortunately, in many cases, finite models exist for satisfiable formulas, and we can first try to find a finite model. If we succeed, the conjecture is disproved by the countermodel; but if we fail, in general, we can not say that the conjecture is false or true.

Currently, there are roughly two main approaches to finite model generation in the first-order logic. The first approach translates the problem into a satisfiability (SAT) problem in the propositional logic, and uses a SAT algorithm (e.g., the DPLL algorithm) to solve it. See for example, [6,8,1,5]. The second approach treats the problem as a constraint satisfaction problem, and uses backtracking search to find the interpretations of the functions/predicates directly. Tools like FINDER [11], FALCON [14], SEM [16] and Mace4 [9] are based on this approach. Gandalf [13] implements both approaches.

Each of the above two approaches has some benefits and weaknesses. For example, the translation approach may generate too many propositional formulas, and the constraint solving (or direct search) approach may not be so efficient on some problems. But using the first-order clauses directly leads to larger reasoning steps and also gives us opportunity to eliminate symmetrical subspaces.

We have been studying how to combine the two approaches. One way is to improve the direct search procedure by incorporating successful techniques developed in the SAT community [4]. Alternatively, we can also improve the translation approach by combining it with first-order model searchers [15]. This paper compares different ways of exploiting symmetries in the problem specification, so that the resulting SAT problem instances are easier. Some examples and experimental results will be given.

The paper is organized as follows. In the next section, we recall some basic concepts and notations, as well as two approaches to model finding. In Section 3, we give an example showing the importance of reducing symmetries. Then we present two approaches for adding constraints to the original problem so as to obtain easier propositional problem instance(s). The first one is static, which produces only one instance; and the second one is dynamic, which usually generates more than one instances. Examples are given to show their strengths and weaknesses.

2 Finite Model Searching

The finite model generation problem can be stated as follows. Given a set of first order formulas and a non-empty finite domain, find an interpretation of all the function symbols and predicate symbols appearing in the formulas such that every formula is true under this interpretation. Such an interpretation is called a *model*. Usually we also assume that the formulas are all clauses, and every variable in a clause is (implicitly) universally quantified.

We do not consider many-sorted formulas in this paper. Without loss of generality, an n-element domain is assumed to be $D_n = \{ 0, 1, \ldots, n-1 \}$. The Boolean domain is $\{$ FALSE, TRUE $\}$. If the arity of each function/predicate symbol is at most 2, a finite model can be conveniently represented by a set of multiplication tables, one for each function/predicate. For example, a 3-element model of the clause $f(x, x) = x$ is like the following:

f	0	1	2
0	0	1	0
1	1	1	0
2	0	1	2

Here f is a binary function symbol and its interpretation is given by the above 2-dimensional matrix. Each entry in the matrix is called a *cell*.

A finite model generation problem may be translated to a propositional satisfiability problem. A model can be represented by a set of assignments to propositional variables. Suppose there are m cells (c_0, c_1, $\ldots$, c_{m-1}) in the multiplication tables of the functions. We can introduce mn propositional variables: p_{ij} $(0 \leq i < m, 0 \leq j < n)$, where p_{ij} is true if and only if the i'th cell c_i has the value j. In addition, we also need one propositional variable for each cell in the predicates' multiplication tables. The first-order clauses can be translated into propositional clauses accordingly. For more details, see for example, [6,8].

Alternatively, we can also search for the values of the cells directly. A finite model generation problem may also be regarded as a constraint satisfaction problem (CSP), which has been studied by many researchers in Artificial Intelligence. The variables of the CSP are the *cell terms*, i.e., ground terms like $f(0, 0)$ and $f(0, 1)$. The domain of each variable is D_n (except for predicates, whose domain is the Boolean domain). The constraints are the set of ground instances of the input clauses. The goal is to find a set of assignments to the cells (e.g., $f(0, 1) = 2$) such that all the ground clauses hold.

Typically backtracking search is used to solve the above problem. The basic idea of the search procedure is roughly like the following: Repeatedly extend a partial model (denoted by *Pmod*) until it becomes a complete model (in which every cell gets a value). Initially *Pmod* is empty. *Pmod* is extended by selecting an unassigned cell and trying to find a value for it. Of course, when no value is appropriate for the cell, backtracking is needed and *Pmod* becomes smaller. Such a procedure may be depicted as a search tree. Each edge of the tree corresponds to choosing a value for some cell.

As mentioned in the Introduction, each of the translation approach and the direct search approach has benefits and weaknesses.

The propositional satisfiability (SAT) problem has been studied for more than 40 years. Many theoretical results have been obtained, and many efficient algorithms have been designed. In recent years, more and more highly efficient SAT solvers are being developed, such as zChaff [10] and BerkMin [3].

On the other hand, the direct search approach works on first-order clauses, and may employ some structural information to speed up the search process. One technique that has been proved to be very useful is the so-called *Least Number Heuristic* (LNH in short) [14,16]. It is based on the observation that in typical benchmark problems, most of the domain elements are "equivalent" when search begins. So we need only choose a few representative values to assign to the cells, and many branches of the search tree can be pruned. The LNH is more effective at the first few levels of the search tree. On many problems, it can reduce the search space significantly, and yet the overhead is negligible. In contrast, few good methods are known to discover and use symmetries in propositional clauses.

It is certainly desirable to combine the benefits of the two approaches, so that more problems can be easily solved.

3 A Motivating Example

Let us look at an example, i.e., finding ortholattices [7]. The axioms (and lemmas) are as follows:

```
m(x,y) = m(y,x).
j(x,y) = j(y,x).
j(j(x,y),z) = j(x,j(y,z)).
c(c(x)) = x.
j(x,m(x,y)) = x.
m(x,y) = c(j(c(x),c(y))).
m(x,x) = x.
j(x,x) = x.
```

```
j(c(x),x) = 1.
m(c(x),x) = 0.
j(1,x) = 1.
j(x,1) = 1.
m(1,x) = x.
m(x,1) = x.
m(0,x) = 0.
m(x,0) = 0.
j(0,x) = x.
j(x,0) = x.
```

When asked to find a 13-element model of the above formulas, MACE 2.2 [8] takes 9.34 seconds to conclude that such a model does not exist. Most of the time is spent on SAT solving rather than obtaining the propositional clauses (DPLL time: 9.09 seconds). If we add the following two clauses to the input:

```
c(0) = 1.    c(2) = 3.
```

the execution time will be 1.14 seconds (DPLL time: 0.89 seconds). The reduction is significant. The experiments were carried out on a Dell desktop computer (Optiplex GX270, Pentium 4, 2.8 GHz, 2G memory).

The above two clauses represent the initial two steps taken by SEM [16]. Note that in the first step, there is only one branch, i.e., SEM decides that only the value `1` can be assigned to `c(0)`. Similarly, in the second step, there is also one choice. So adding the two clauses does not change the satisfiability of the original problem.

4 Adding Formulas to Eliminate Symmetries

When solving the quasigroup problems, Fujita *et al.* [2,12] add a few clauses which eliminate quite many symmetrical subspaces. This greatly reduces the search time. But the additional constraints are domain-specific, namely, they can only be applied to quasigroup problems and other similar problems.

MACE [8] has an option (‘`-c`’) which allows the user to impose the constraint that the constants are different from each other. It is quite helpful when finding counterexamples, because the negation of the conjecture usually contains Skolem constants. It is also useful when, for instance, finding non-commutative groups. But that option may miss some solutions, in which two constants are assigned the same domain element. For example, when this option is used, MACE fails to find a 10-element countermodel which shows that some equation (i.e., the equation E1 in [7]) does not hold for ortholattices. MACE has another option (‘`-z`’) which adds isomorphism constraints to the generated propositional formula. But it applies only to constants.

The LNH is a more general method. We can simulate it by adding certain constraints. This kind of static symmetry reduction is implemented in Paradox. See Section 6 of [1]. A similar method is adopted in SAGE [5]. For simplicity, we assume that no domain elements appear in the input and that there is only one binary function symbol `f`. SAGE adds the following constraints (denoted by *Cf*):

```
f(0,0)=0  |  f(0,0)=1.
f(0,1)=0  |  f(0,1)=1  |  f(0,1)=2.
f(1,0)=0  |  f(1,0)=1  |  f(1,0)=2  |  f(1,0)=3.
...
```

It can prune the search tree greatly, since we now need to examine only 2 (instead of `n`) values for `f(0,0)`, only 3 (instead of `n`) values for `f(0,1)`, ...

To get an understanding of its effectiveness, let us look at the QG5 problem. It has only one binary function symbol '`f`' whose multiplication table should be a quasigroup. In addition to this property, it has the following axioms:

```
f(x,x) = x.
f(f(f(y,x),y),y) = x.
f(y,f(f(x,y),y)) = x.
f(f(y,f(x,y)),y) = x.
```

Suppose we try to find all of its models. If we do not use any method for eliminating isomorphism, there are 120 models of size 7, and 720 models of size 8. If we add the above three *Cf* formulas to the input, there are 24 models of size 7, and 24 models of size 8. But when we use the LNH, there is only one model of size 7, and one model of size 8.

From this simple example, we can see that the method is helpful, but it is not good enough.

4.1 More Accurate Simulation

The *Cf* constraints are only an approximation to the LNH. Some combinations actually need not be considered. For example, when `f(0,0) = 0` and `f(0,1) = 1`, we need not consider the case `f(1,0) = 3`.

Recently, we implemented a new strategy which adds more constraints to the generated SAT instance. There are two kinds of constraints, denoted by *C1* and *C2*. We add them while "visiting" the cells in some fixed order: c_0, c_1, c_2, ... For unary function g, we visit the cells in this order:

$$g(0),\ g(1),\ g(2),\ \ldots$$

```
proc addCons(int $i$)
   /* processing cell $c_i$ */
{
   Update the sets $S_1$ and $S_2$;
   Compute the sets $S_4$ and $S_5$;
   $S'_4 \;=\; S_4$;
   if ($S_5$ has more than one elements) {
      /* The elements are symmetric */
      Let $v$ be the smallest element in $S_5$;
      Add the constraints $C1(i)$;
      $S_3 \;=\; S_3 \cup \{v\}$;
      $S'_4 \;=\; S'_4 \cup \{v\}$;
   }
   if ($S'_4$ has more than one elements) {
      Add the constraints $C2(i)$;
   }
}
```

Fig. 1. Adding Constraints to Reduce the Search Space

For binary function f, we visit the cells in this order:

$$f(0,0),\ f(0,1),\ f(1,0),\ f(1,1),\ f(0,2),\ \ldots$$

Suppose currently we are visiting cell c_i. We use the procedure in Fig. 1 to add constraints. It involves the following sets of domain elements:

- S_0: the set of domain elements appearing in the input formulas.
- $S_1 = \{$ the arguments of $c_k \mid 0 \leq k \leq i$ $\}$.
- $S_2 = S_0 \cup S_1$. It contains the elements which are distinguished (i.e., not symmetrical to other elements).
- S_3: the set of domain elements which are chosen as representatives of "unused" elements.
- $S_4 = S_3 - S_2$. We also use a closely related set S'_4.
- $S_5 = D_n - (S_2 \cup S_3)$.

Except for S_0, the other sets are changing dynamically.

Initially $i = 0$, and the set S_3 is empty. $C1(i)$ consists of the following constraints:

$$\bigvee_{j \in S_2 \cup S_3 \cup \{v\}} (c_i = j);$$

$$c_i \neq k, \text{ for each } k \in S_5 - \{v\}.$$

$C2(i)$ consists of the following constraints $C2(i)(a,b)$:

$$\bigwedge_{j<i}(c_j \neq a \wedge c_j \neq b) \;\rightarrow\; c_i \neq b$$

for each pair $a, b \in S'_4$ $(a < b)$.

During the process, the set S_5 is decreasing. When S_5 has only one element or it is empty, we do not add the $C1$ constraints. Similarly, when S'_4 has no more than 1 element, we do not add the *C2* constraints. When both these happen, the process can be terminated.

In the Appendix, we shall prove that the method is correct.

4.2 Example

Now we look at an example, i.e., finding a 6-element group. The axioms are the following clauses:

```
f(x,0) = x
f(0,x) = x
f(x,g(x)) = 0
f(g(x),x) = 0
f(x,f(y,z)) = f(f(x,y),z)
```

where `0` is a special domain element (i.e., the identity element of a group).

To add constraints to eliminate symmetrical subspaces, SAGE visits the cells in the following order:

```
f(1,1),  f(1,2),  f(2,1),  f(2,2),
f(1,3),  f(3,1),  f(2,3),  f(3,2),  f(3,3),
...
```

Note that the first row and the first column in the multiplication table of `f` have already been fixed (because of the first two axioms).

The following are the $C1$ constraints:

$$f(1,1)=0 \vee f(1,1)=1 \vee f(1,1)=2;\quad f(1,1)\neq 3;\quad f(1,1)\neq 4;\quad f(1,1)\neq 5;$$
$$f(1,2)=0 \vee f(1,2)=1 \vee f(1,2)=2 \vee f(1,2)=3;\quad f(1,2)\neq 4;\quad f(1,2)\neq 5;$$
$$f(2,1)=0 \vee f(2,1)=1 \vee f(2,1)=2 \vee f(2,1)=3 \vee f(2,1)=4;\quad f(2,1)\neq 5;$$
$$f(2,2)=0 \vee f(2,2)=1 \vee f(2,2)=2 \vee f(2,2)=3 \vee f(2,2)=4 \vee f(2,2)=5.$$

The following are the *C2* constraints:

$$f(1,2) \neq 3 \ \wedge \ f(1,2) \neq 4 \ \rightarrow \ f(2,1) \neq 4$$
$$f(1,2) \neq 3 \ \wedge \ f(1,2) \neq 4 \ \wedge \ f(2,1) \neq 3 \ \wedge \ f(2,1) \neq 4 \ \rightarrow \ f(2,2) \neq 4$$
$$f(1,2) \neq 3 \ \wedge \ f(1,2) \neq 5 \ \wedge \ f(2,1) \neq 3 \ \wedge \ f(2,1) \neq 5 \ \rightarrow \ f(2,2) \neq 5$$
$$f(2,1) \neq 4 \ \wedge \ f(2,1) \neq 5 \ \rightarrow \ f(2,2) \neq 5$$
$$f(2,1) \neq 4 \ \wedge \ f(2,1) \neq 5 \ \wedge \ f(2,2) \neq 4 \ \wedge \ f(2,2) \neq 5 \ \rightarrow \ f(1,3) \neq 5$$
$$f(2,1) \neq 4 \ \wedge \ f(2,1) \neq 5 \ \wedge \ f(2,2) \neq 4 \ \wedge \ f(2,2) \neq 5 \\ \wedge \ f(1,3) \neq 4 \ \wedge \ f(1,3) \neq 5 \ \rightarrow \ f(3,1) \neq 5$$
$$f(2,1) \neq 4 \ \wedge \ f(2,1) \neq 5 \ \wedge \ f(2,2) \neq 4 \ \wedge \ f(2,2) \neq 5 \\ \wedge \ f(1,3) \neq 4 \ \wedge \ f(1,3) \neq 5 \ \wedge \ f(3,1) \neq 4 \ \wedge \ f(3,1) \neq 5 \ \rightarrow \ f(2,3) \neq 5$$
$$f(2,1) \neq 4 \ \wedge \ f(2,1) \neq 5 \ \wedge \ f(2,2) \neq 4 \ \wedge \ f(2,2) \neq 5 \\ \wedge \ f(1,3) \neq 4 \ \wedge \ f(1,3) \neq 5 \ \wedge \ f(3,1) \neq 4 \ \wedge \ f(3,1) \neq 5 \\ \wedge \ f(2,3) \neq 4 \ \wedge \ f(2,3) \neq 5 \ \rightarrow \ f(3,2) \neq 5$$
$$f(2,1) \neq 4 \ \wedge \ f(2,1) \neq 5 \ \wedge \ f(2,2) \neq 4 \ \wedge \ f(2,2) \neq 5 \\ \wedge \ f(1,3) \neq 4 \ \wedge \ f(1,3) \neq 5 \ \wedge \ f(3,1) \neq 4 \ \wedge \ f(3,1) \neq 5 \\ \wedge \ f(2,3) \neq 4 \ \wedge \ f(2,3) \neq 5 \ \wedge \ f(3,2) \neq 4 \ \wedge \ f(3,2) \neq 5 \ \rightarrow \ f(3,3) \neq 5$$

Note that some optimizations have been done. For example, $f(1,1) \neq 3$ does not appear in the first formula, since the $C1$ constraints restrict the values of $f(1,1)$ to be in the set { 0, 1, 2 }.

If we use no additional constraints to eliminate symmetries, we will find 80 models. If we add the *C1* constraints when generating the propositional clauses, 16 models will be found. If we add both *C1* and *C2* constraints, then only 9 models are found.

Here we give two models of group theory which are generated when the *C1* constraints are used as additional constraints, but not generated when both *C1* and *C2* constraints are used. The first one is defined as follows:

f	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	0	3	2	5	4
2	2	3	5	4	0	1
3	3	2	4	5	1	0
4	4	5	0	1	3	2
5	5	4	1	0	2	3

	0	1	2	3	4	5
g	0	1	4	5	2	3

This model is eliminated because *C2* contains the constraint:

$$f(2,1) \neq 4 \ \wedge \ f(2,1) \neq 5 \ \rightarrow \ f(2,2) \neq 5.$$

The second one is defined as follows:

f	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	0	1	2
4	4	5	3	2	0	1
5	5	3	4	1	2	0

	0	1	2	3	4	5
g	0	2	1	3	4	5

The reason is that *C2* contains the constraint:

$$f(2,1) \neq 4 \ \wedge \ f(2,1) \neq 5 \ \wedge \ f(2,2) \neq 4 \ \wedge \ f(2,2) \neq 5 \ \rightarrow \ f(1,3) \neq 5.$$

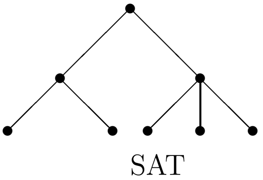
Now we increase the size of groups to 10. If we use the *C1* constraints only, there are nearly one thousand models; but if we use both *C1* and *C2* constraints, the number of models is reduced to 11.

5 Generating Multiple SAT Instances Dynamically

As mentioned in Section 2, a backtracking search method works by extending partial models. It is also mentioned that the LNH is more effective at the first few levels of the search tree. Below certain levels, the domain elements are no longer "equivalenet" and the heuristic is not effective. On the other hand, for many problems, propositional reasoning is more efficient at most nodes of the search tree.

Naturally one may think of combining first-order model searching with SAT solving, as demonstrated by the example in Section 3.

In [15], we propose a scheme for combining the two approaches, and report some experiences with SEM and MACE. The scheme looks like the following:



At the first few levels of the search, we use SEM with the LNH. Below certain levels (e.g., when every domain element is distinct), the search is transfered to a SAT solver. The borderline can be decided by the user.

Table 1
Number of Partial Models for Various Problems

size	7	8	9	10	11	12	13
QG5	1	1	5	12	26	70	217
OL	23	54	849	6501	>10000	>10000	>10000
NCG	7	16	31	57	79	223	210

Let us look at the QG5 problem again. Its axioms are given in the previous section. For this problem, the first 3 steps of SEM's search tree is like the following:

```
f(0,1) = 2;    f(2,0) = 3;     f(2,1) = 4.
```

At each step, SEM concludes – using the LNH and through various kinds of reasoning – that there is only one value that can be assigned to the corresponding cell. If we add these three equations to the input, and ask SEM to find all the models, without using the LNH, SEM will find that there are 2 models of size 7, 6 models of size 8. This is not too far away from the optimal numbers (1 model of size 7 and 1 model of size 8).

Thus adding the above constraints is quite helpful for eliminating isomorphic subspaces. Of course, we can ask SEM to go beyond the 3 steps and more subspaces can be eliminated. In general, more than one SAT instances are generated using this approach.

Is there any overhead? Yes. The main overhead will be the translation time (the time for obtaining propositional clauses from first-order ones), and perhaps reading/writing files. That will be significant if many SAT instances are generated. On the other hand, if the problem is difficult, the nodes of SEM's search tree are not so many, and the majority of work will be done by a SAT solver. The translation time is not so much, if compared with the searching time. In that case, the combination will be very helpful.

We have slightly modified SEM so that it backtracks when the LNH is no longer effective (i.e., when no domain elements are "equivalent"). We ask the modified program to count how many partial models it generates. Table 1 gives the number of partial models generated by SEM, on several problems. In addition to QG5 and ortholattice (OL) mentioned earlier, we have tested the program on the non-commutative group (NCG) problem. The first line of the table gives the size of the model, while the other lines give the corresponding numbers of partial models. We can see that, for NCG and QG5, there are not too many partial models. But OL has many partial models. This is probably because that the problem is too easy or has too many solutions.

We have also tried several other problems. Some of them are too easy, and some are too hard. For example, the combinatory logic problem `cl_BM` is already quite difficult for SEM, when the size of the model is 6. So the results are not included in the above table. When the size is 6, 1599 partial models are generated; and when the size is 7, 49438 partial models are generated.

One way to reduce the number of partial models is to ask SEM to backtrack earlier (e.g., when there are still some "equivalent" domain elements). But then the symmetries are not exploited fully. Another way is to ask SEM to solve the easier subproblems, in which many cells are assigned values.

If we are just looking for one model, it is not necessary to generate all the partial models. In this case, it will be beneficial to combine SEM with a SAT solver more closely, as done in [15]. Then as soon as one partial model leads to a full model, SEM can be terminated and no more partial models need to be generated.

6 Concluding Remarks

As many highly efficient SAT solvers are being developed in recent years, it becomes more interesting to use them to find finite models and counterexamples in the first-order logic. If we take symmetries into account when generating propositional clauses, easier SAT instances may be obtained.

In this paper, we discussed two different approaches. The first one is static, which adds some formulas to the input and then gets a set of propositional clauses in the conventional way. Only one SAT instance is generated. We presented a procedure for adding the formulas, which is an approximation to the least number heuristic. It can usually eliminate many isomorphic models, and makes the resulting SAT instance easier.

The second approach is dynamic, which uses a first-order model searcher to derive some partial models, and then gets a number of SAT instances (each corresponding to a partial model). Comparatively speaking, this approach can usually eliminate more isomorphic models. But for some problems, there may be too many partial models.

Some examples are given, and experimental results are reported, using existing tools (or variations of them). We believe that the combination of first-order model searching and SAT solving is very promising for finding large models and counterexamples.

References

[1] K. Claessen and N. Sörensson, New techniques that improve MACE-style finite model finding. In: *Model Computation – Principles, Algorithms, Applications*, CADE-19 Workshop W4, Miami, Florida, USA, 2003.

[2] M. Fujita, J. Slaney and F. Bennett, Automatic generation of some results in finite algebra, *Proc. 13th Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, 52–57, 1993.

[3] E. Goldberg and Y. Novikov, BerkMin: A fast and robust SAT solver, *Design, Automation, and Test in Europe (DATE'02)*, 142–149, 2002.

[4] Z. Huang, H. Zhang and J. Zhang, Improving first-order model searching by propositional reasoning and lemma learning, *Proc. 7th Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2004.

[5] Z. Huang and J. Zhang, Generating SAT instances from first-order formulas, *J. of Software*, to appear.

[6] S. Kim and H. Zhang, ModGen: Theorem proving by model generation, *Proc. 12th AAAI*, 162–167, 1994.

[7] W. McCune, Automatic proofs and counterexamples for some ortholattice identities, *Information Processing Letters*, 65(6): 285–291, 1998.

[8] W. McCune, MACE 2.0 reference manual and guide, Technical Memorandum ANL/MCS-TM-249, Argonne National Laboratory, Argonne, IL, USA, 2001.

[9] W. McCune, Mace4 reference manual and guide, Technical Memorandum No. 264, Argonne National Laboratory, Argonne, IL, USA, 2003.

[10] M. Moskewicz *et al*, Chaff: Engineering an efficient SAT solver, *Proc. 38th Design Automation Conference*, 530–535, 2001.

[11] J. Slaney, FINDER: Finite domain enumerator – system description, *Proc. CADE-12*, 798–801, 1994.

[12] J. Slaney, M. Fujita and M. Stickel, Automated reasoning and exhaustive search: Quasigroup existence problems, *Computers and Mathematics with Applications* 29(2): 115–132, 1995.

[13] T. Tammet, Finite model building: improvements and comparisons. In: *Model Computation – Principles, Algorithms, Applications*, CADE-19 Workshop W4, Miami, Florida, USA, 2003.

[14] J. Zhang, Constructing finite algebras with FALCON, *J. Automated Reasoning* 17(1): 1–22, 1996.

[15] J. Zhang, Automatic symmetry breaking method combined with SAT, *Proc. ACM Symp. on Applied Computing*, 17–21, 2001.

[16] J. Zhang and H. Zhang, SEM: A system for enumerating models, *Proc. 14th IJCAI*, 298–303, 1995.

Appendix. Correctness of the Static Method

We shall prove that adding the *C1* and *C2* constraints does not change satisfiability. Let

$$\begin{aligned} SC1(m) &= C1(0) \cup \ldots \cup C1(m) \\ SC2(m) &= C2(0) \cup \ldots \cup C2(m) \end{aligned}$$

We denote the original problem by P, and try to prove the following lemmas:

(a) P is satisfiable iff $P \wedge SC1(0) \wedge SC2(0)$ is satisfiable.

(b) $P \wedge SC1(m) \wedge SC2(m)$ is satisfiable iff $P \wedge SC1(m+1) \wedge SC2(m)$ is satisfiable.

(c) $P \wedge SC1(m+1) \wedge SC2(m)$ is satisfiable iff $P \wedge SC1(m+1) \wedge SC2(m+1)$ is satisfiable.

For each lemma, we only need to prove one direction (i.e., the "only if" part). We assume that the set $C1(i)$ is not empty, for every i.

Proof of (a): Suppose P is satisfiable and has a model M. Let v_0 be the value of the cell c_0 in M. When visiting c_0, both S_3 and S_4 are empty. So is $C2(0)$. Thus we only need to consider $C1(0)$. Suppose the procedure `addCons(0)` chooses the value v for c_0. If $v_0 \in S_2 \cup S_3$ or $v_0 = v$, M will satisfy $C1(0)$. Otherwise, we can apply the permutation $\sigma = \{\langle v, v_0 \rangle\}$ to M, and get $\sigma(M)$. Since neither v nor v_0 appears in the input specification of P, $\sigma(M)$ is also a model of M. In this model, the value of c_0 is v, and $C1(0)$ is satisfied. In summary, $P \wedge SC1(0) \wedge SC2(0)$ is satisfiable.

Proof of (b): Suppose $P \wedge SC1(m) \wedge SC2(m)$ is satisfiable and has a model M. Let v_{m+1} be the value of the cell c_{m+1} in M. Assume that the procedure `addCons(m+1)` chooses the value v for c_{m+1}. If $v_{m+1} \in S_2 \cup S_3$ or $v_{m+1} = v$, $C1(m+1)$ will be satisfied, and M will be a model of $P \wedge SC1(m+1) \wedge SC2(m)$. Otherwise, we can apply the permutation $\sigma = \{\langle v, v_{m+1} \rangle\}$ on M, and get $\sigma(M)$. Since neither v nor v_{m+1} appears in $P \wedge SC1(m) \wedge SC2(m)$, $\sigma(M)$ is also a model of $P \wedge SC1(m) \wedge SC2(m)$. In $\sigma(M)$, the value of c_{m+1} is v. So $\sigma(M)$ is model of $P \wedge SC1(m+1) \wedge SC2(m)$. In summary, $P \wedge SC1(m+1) \wedge SC2(m)$ is satisfiable.

Proof of (c): Suppose $P \wedge SC1(m+1) \wedge SC2(m)$ is satisfiable. We need to prove that adding the constraints $C2(m+1)$ does not change satisfiability. We sort the constraints $C2(m+1)(a,b)$ in ascending order of $\langle a, b \rangle$. Denote them by cl_0, cl_1, …, cl_t. Let $VC = \{c_j \mid 0 \leq j \leq m\}$, i.e., the cells that already have been processed. We try to prove the following:

(c1) If $P \wedge SC1(m+1) \wedge SC2(m)$ is satisfiable, then $P \wedge SC1(m+1) \wedge SC2(m) \wedge cl_0$ is satisfiable.

(c2) For any $i \geq 0$, if $P \wedge SC1(m+1) \wedge SC2(m) \wedge cl_0 \wedge \ldots \wedge cl_i$ is satisfiable, then $P \wedge SC1(m+1) \wedge SC2(m) \wedge cl_0 \wedge \ldots \wedge cl_i \wedge cl_{i+1}$ is satisfiable.

Proof of (c1). Suppose that $P \wedge SC1(m+1) \wedge SC2(m)$ is satisfiable, and it has a model M. Let S_6 denote the set of domain elements that have been assigned to some cell in VC, i.e., $\{$ the value of c_j in $M \mid 0 \leq j \leq m$ $\}$. Suppose cl_0 is $C2(m+1)(a,b)$:

$$\bigwedge_{c_j \in VC} (c_j \neq a \wedge c_j \neq b) \ \rightarrow\ c_{m+1} \neq b,$$

where $a, b \in S'_4$ and $a < b$.

- If $a \in S_6$ or $b \in S_6$, which means that a cell in VC has been assigned the value a or b. So cl_0 holds in M.
- If c_{m+1} is not assigned the value b, cl_0 will also hold in M.
- Neither of the above is true. Then we can apply the permutation $\sigma = \{\langle a, b\rangle\}$ to M, and get $\sigma(M)$. The permutation does not change the value of any cell in VC. Moreover, a and b do not appear in the input specification of P, so $\sigma(M)$ is a model of $P \wedge SC1(m+1) \wedge SC2(m)$. In this model, the value of c_{m+1} is a, which is different from b. So cl_0 holds. Thus $P \wedge SC1(m+1)$ $\wedge SC2(m) \wedge cl_0$ is satisfiable.

Proof of (c2). Suppose that $P \wedge SC1(m+1) \wedge SC2(m) \wedge cl_0 \wedge \ldots \wedge cl_i$ is satisfiable, and it has a model M. Let S_6 denote the set of domain elements that have been assigned to some cell in VC, i.e., $\{$ the value of c_j in $M \mid 0 \leq j \leq m$ $\}$. Consider the constraint cl_{i+1}:

$$\bigwedge_{c_j \in VC} (c_j \neq a \wedge c_j \neq b) \ \rightarrow\ c_{m+1} \neq b,$$

where $a, b \in S'_4$ and $a < b$.

- If $a \in S_6$ or $b \in S_6$, which means that a cell in VC has been assigned the value a or b. So cl_{i+1} holds in M.
- If c_{m+1} is not assigned the value b, cl_{i+1} will also hold in M.
- Neither of the above is true. Then we can apply the permutation $\sigma = \{\langle a, b\rangle\}$ to M, and get $\sigma(M)$. The permutation does not change the value of any cell in VC. Moreover, a and b do not appear in the input specification of P, so $\sigma(M)$ is a model of $P \wedge SC1(m+1) \wedge SC2(m)$. In this model, the value of c_{m+1} is a, which is different from b. So cl_{i+1} holds.

  Since $a < b$ and the constraints $cl_0, cl_1, \ldots, cl_i$ hold in M, they also hold in $\sigma(M)$. Otherwise, suppose that for some k $(0 \leq k \leq i)$, cl_k:

$$\bigwedge_{c_j \in VC} (c_j \neq a' \wedge c_j \neq b') \ \rightarrow\ c_{m+1} \neq b'$$

does not hold in $\sigma(M)$. Then we have $b' = a$, and $a' < a$. If [$\bigwedge_{c_j \in VC}(c_j \neq a' \wedge c_j \neq a) \;\rightarrow\; c_{m+1} \neq a$] does not hold in $\sigma(M)$, then [$\bigwedge_{c_j \in VC}(c_j \neq a' \wedge c_j \neq b) \;\rightarrow\; c_{m+1} \neq b$] does not hold in M. This contradicts with our assumption, since $a' < a$.

Thus $P \wedge SC1(m+1) \wedge SC2(m) \wedge cl_0 \wedge \ldots \wedge cl_i \wedge cl_{i+1}$ is satisfiable.