Journal of Automated Reasoning **17:** 1–22, 1996.

Constructing Finite Algebras with FALCON

JIAN ZHANG
Dept. of Computer Science, Univ. of Iowa, Iowa City, IA 52242, U.S.A. and
Institute of Software, Academia Sinica, Beijing 100080, P.R. China
e-mail: jzhang@cs.uiowa.edu

(Received: 15 November 1994; accepted: 12 April 1995)

Abstract. The generation of models and counterexamples is an important form of reasoning. In this paper, we give a formal account of a system, called **FALCON**, for constructing finite algebras from given equational axioms. The abstract algorithms, as well as some implementation details and sample applications, are presented. The generation of finite models is viewed as a constraint satisfaction problem, with ground instances of the axioms as constraints. One feature of the system is that it employs a very simple technique, called the *least number heuristic*, to eliminate isomorphic (partial) models, thus reducing the size of the search space. The correctness of the heuristic is proved. Some experimental data are given to show the performance and applications of the system.

AMS Subject Classification: 03B35, 68T15, 03C13, 03C05.

Key words: model generation, finite algebras, constraint satisfaction, least number heuristic.

1. Introduction

The notion of models is important in mathematical logic. It is easy to verify that some structure is a model of some theory. But it is very difficult (or even impossible) to find a model of an arbitrary consistent theory. By model generation we mean the following: Given a theory defined by a set of axioms, find a model of this theory. The model generation problem is undecidable in general. In this paper, we restrict ourselves to finite models of equational theories.

The significance of (finite) model generation can be seen from the following aspects:

- The existence of a model implies the consistency of the theory.
- Conventional theorem provers have difficulties when the conjecture to be proved is not a theorem. In such cases, one may find a model (i.e., counterexample) that shows the falsity of the conjecture.
- Finite models may provide some kind of semantic guidance that helps refutation-based theorem provers to find proofs more quickly.
- The existence (and enumeration) of finite objects is interesting in itself in some branches of mathematics.
- Finite models can help people to understand an abstract theory and to formulate conjectures.

But for a long time, because of its intrinsic difficulty, automatic model generation has not received as much attention as it deserves. Recently, several researchers have solved some previously open questions on the existence of certain quasigroups (or Latin squares), using their model generation programs (see, for example, [4, 9, 14, 20]). Slaney [13] also reported some experiments on guiding theorem provers with finite structures in propositional logics.

In this paper, we study the finite model generation problem for theories whose axioms are equations. Such theories abound in abstract algebra, which has been a typical application domain of automated theorem proving. We shall describe a system for constructing finite algebras, which is called FALCON. In our approach, the problem is considered as satisfying logical constraints in finite domains. To solve the problem, we use a backtracking search procedure. A simple yet powerful technique, called the least number heuristic, is used for rejecting isomorphism between the (partial) models, so that the size of the search space is significantly reduced. We shall illustrate the use of our system with various examples, and show its performance on some test problems.

The paper is organized as follows. The next section contains some basic concepts and notions that will be used later. Then several abstract procedures for finding finite models are presented. Following that is a brief description of the FALCON system and its applications. Finally, our approach is compared with other methods, and some of our future work is outlined.

2. Equational Theories and Their Models

2.1. Equations and Algebras

An equational theory has a set of (logical) equations as axioms. An equation takes the form of $t_1 = t_2$, where t_1 and t_2 are terms. Each term is constructed recursively from function symbols* and variables. Nullary function symbols are also called constant symbols. Usually we use the letters $a, b, c, \ldots$ for constants; $f, g, h, \ldots$ for functions; and $u, \ldots, z$ for variables. All variables in the equations are assumed to be universally quantified. An equation with no variables is called a *ground* equation.

A model of an equational theory is an algebra, that is, a nonempty set with appropriately defined functions, such that the axioms are satisfied. In this paper, we consider only *finite* models. A finite model is often represented by the operation tables of the functions. From now on, we shall denote by n some fixed positive integer, which means the cardinality (or the size) of a finite model. For any n-element model, we assume that its underlying set is $D_n = \{0, 1, \ldots, n-1\}$.

EXAMPLE 1. The first example was used in [21]. There is only one function symbol, f, which is binary. The axioms are as follows:

* We assume that each function symbol has a fixed arity.

```
(A1)  f(x,x) = x,
(A2)  f(f(x,y),x) = y.
```

Let us denote the corresponding theory by T_1. It has one model of size 4, which can be represented by the following table:

f	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

The elements of D_n can be regarded as special constants, which are implicitly introduced into every theory. Let us call them *numeral constants* (or simply *numerals* or *numbers*). It is understood that these constants are different from the ordinary function symbols and variable names. We also assume that in every model of size n, any equation of the form $i = j$, where i and j are two different numerals, evaluates to FALSE.

Sometimes the numeral constants will be treated like integers. For example, it is assumed that there is a linear ordering on them, namely, $0 < 1 < \cdots < n-1$. So we can talk about the *maximum* and *minimum* of a set of numerals. Occasionally the integer (-1) is needed, which is less than any element of D_n. We use $[l \ldots u]$ to denote the set $\{i \mid l \leqslant i \leqslant u\}$.

2.2. Working with Ground Equations

In the context of finite model generation, both the axioms and the model may be represented by some ground equations that contain numeral constants.

Suppose we are given a set E of equational axioms (which does not contain any numeral constant), and our goal is to find an n-element model of E. Since the variables in the equations are universally quantified, we may instantiate the axioms by substituting numerals for variables in all possible ways. We get the following set of ground equations:

$\{\ eq[\ldots, e_i/x_i, \ldots] \mid$
for every $eq \in E$ and
for every possible substitution $\langle \ldots, e_i/x_i, \ldots \rangle$,
where $e_i \in D_n$ and x_i is a variable in eq
$\}$.

Let us denote the above set by $GI(E, n)$. Obviously, an interpretation (over the set D_n) is a model of E if and only if it satisfies all the equations in $GI(E, n)$.

To find a finite model, we have to determine the value of each cell (or entry) in the operation tables of the functions. A cell corresponds to a term of the form $f(e_1, \ldots, e_k)$, where f is an ordinary function symbol of arity k, and each e_i

$(1 \leqslant i \leqslant k)$ is a numeral in D_n. Such a term will be called a *cell term*. Given E and n, the set of all cell terms is denoted by $CT(E, n)$.

If an equation has exactly one occurrence of an ordinary function symbol, it will be called a *simple* equation. A simple ground equation is called an *assignment*. The following are some examples of simple equations:

$$f(x, x) = x, \qquad g(0, x) = x, \qquad h(1) = 0, \qquad a = 0.$$

Among these, the last two are assignments. In general, an assignment takes the form of

$$ce = num \quad \text{or} \quad num = ce,$$

where ce is a cell term and num is a numeral. Intuitively, such an equation defines the value of the cell ce to be num. A set of assignments is *consistent* if it does not contain two assignments whose cell terms are the same but whose numerals differ.

An assignment may also be regarded as a rewrite rule, whose left-hand side is the cell term and whose right-hand side is the numeral. A consistent set of assignments can be used as rewrite rules to reduce ground terms and simplify the equations in $GI(E, n)$. We assume there is a procedure `reduce`(eq, As) that reduces an equation eq with respect to a consistent set As of assignments. For any $i, j \in D_n (i \neq j)$, the equation $i = i$ is reduced to TRUE, and the equation $i = j$ is reduced to FALSE, regardless of what value As may take.

A consistent set of assignments, in which all the cells are defined, represents a (complete) finite structure. If all the axioms are satisfied, that is, all the equations in $GI(E, n)$ are reduced to TRUE, then it represents a finite model.

EXAMPLE 1 (continued). For the theory T_1, the set $GI(T_1, 4)$ consists of the following equations:

$$f(0,0) = 0, \qquad (A1\_0)$$

$$\ldots$$

$$f(3,3) = 3, \qquad (A1\_3)$$

$$f(f(0,0),0) = 0, \qquad (A2\_0\_0)$$

$$\ldots$$

$$f(f(3,3),3) = 3, \qquad (A2\_3\_3)$$

where (A1_i) is generated from (A1) by substituting i for x, and (A2_i_j) is generated from (A2) by substituting i, j for x, y, respectively.

The four-element model can be represented as

$$\begin{array}{llll} f(0,0)=0, & f(0,1)=2, & f(0,2)=3, & f(0,3)=1, \\ f(1,0)=3, & f(1,1)=1, & f(1,2)=0, & f(1,3)=2, \\ f(2,0)=1, & f(2,1)=3, & f(2,2)=2, & f(2,3)=0, \\ f(3,0)=2, & f(3,1)=0, & f(3,2)=1, & f(3,3)=3. \end{array}$$

It is straightforward to verify that, in this model, each ground equation in $GI(T_1, 4)$ is reduced to TRUE.

Summary. In our language, there are variable names, ordinary function symbols, and numeral constants. A theory is defined by a set of equations that do not contain numerals. A finite model can be represented by the operation tables of the functions, or equivalently, by a set of assignments. To find a finite model of an equational theory, one has to give a value to each cell in the tables such that all the axioms hold. More precisely, constructing an n-element model of E is equivalent to finding a set of assignments that assigns exactly one numeral in D_n to each cell term in $CT(E, n)$, such that all the equations in $GI(E, n)$ are reduced to TRUE.

3. Procedures for Finding Finite Models

In the preceding section, we argued that a finite model is actually a set of assignments that makes a set of ground equations true. Thus finite model generation can be viewed as constraint satisfaction. In this section, we describe some abstract procedures for doing this. We shall use the notations introduced earlier.

3.1. Constraint Satisfaction by Backtracking

A constraint satisfaction problem (CSP) consists of

- a set of variables $\{ x_i \mid 1 \leqslant i \leqslant m \}$;
- a domain of values, V_i, for each variable x_i; and
- a collection of constraints.

The goal is to find suitable values for the variables such that the constraints are satisfied.

Backtracking procedures are often used to solve CSPs. Such a procedure works by extending partial solutions. A partial solution is a k-tuple $\langle x_1 = v_1, \ldots, x_k = v_k \rangle$, where each $v_i \in V_i$, and $k \leqslant m$. Initially $k = 0$. At each step of the procedure, we try to extend a k-tuple to a $(k+1)$-tuple, while not violating any constraint. If such an extension is impossible, we go back to the $(k-1)$-tuple $\langle x_1 = v_1, \ldots, x_{k-1} = v_{k-1} \rangle$ and consider another value for x_k. The procedure either ends with a complete solution or backtracks to the null tuple, which means there is no solution.

3.2. SATISFYING EQUATIONAL AXIOMS IN FINITE DOMAINS

Finite model generation in equational theories may be considered as a CSP. Here the "variables" are the cell terms in $CT(E,n)$, whose value domains are the same (i.e., the set D_n). The constraints are specified by the equational axioms or, equivalently, the set $GI(E,n)$ of ground equations.

The recursive procedure `SatE_0` in Figure 1 can be used to find an n-element model of E. The procedure has one parameter, $Pmod$ (for "partial model"), which is a set of assignments. Obviously, a model exists if and only if `SatE_0`($\emptyset$) returns TRUE.

The procedure `assignment_propagate()` uses the assignments in the current partial model to simplify ground equations in $GI(E,n)$. It is a propagation because during the process new assignments can be generated. This procedure is described in Figure 2.

```
bool SatE_0(Pmod)
{
   assignment_propagate(Pmod);
   if contradiction discovered return(FALSE);
   if all cells have been assigned values return(TRUE);
   choose a cell f(e1, ..., ek) which does not have a value;
   for i = 0 to n − 1 do
      if SatE_0(Pmod ∪ {f(e1, ..., ek) = i}) return(TRUE);
   return(FALSE);
}
```

Fig. 1. Search for a finite model.

```
assignment_propagate(Pmod)
{
   repeat until no new assignments are generated {
      for each ground equation eq ∈ GI(E, n) do {
         eq' = reduce(eq,Pmod);
         if eq' is of the form i = j (with i and j different)
         return; /* contradiction is found */
         if eq' is a simple equation
         add this new assignment to Pmod;
      }
   }
}
```

Fig. 2. Assignment propagation.

The result of assignment propagation is that either contradiction is found or the partial model becomes larger (or remains the same). As an example, let us look at the problem in Example 1. Suppose the current partial model is

$$PM_1\colon\ \{f(0,0)=0,\ f(0,1)=0\}.$$

Then the ground Equation (A2_0_1) is simplified to $0=1$, which is a contradiction. Suppose, instead, that the partial model is

$$PM_2\colon\ \{f(0,0)=0,\ f(0,1)=2\}.$$

Then the ground Equation (A2_0_1) is simplified to $f(2,0)=1$. This new assignment, in turn, simplifies the Equation (A2_2_0) to $f(1,2)=0$. And another new assignment is obtained. As a result of assignment propagation, the partial model PM_2 is extended to the larger partial model

$$PM_2'\colon\ \{f(0,0)=0,\ f(0,1)=2,\ f(2,0)=1,\ f(1,2)=0\}.$$

3.3. A Heuristic for Rejecting Isomorphism

The above procedure `SatE_0()` is sound and complete, but it is not efficient for many applications. One important source of inefficiency is that considerable symmetry exists in the search space. When numerals are used, we can represent one model in many different ways, simply by permuting the numerals. Thus, when searching for the values of the cells, the procedure considers more possibilities than necessary.

To deal with the symmetry problem, FALCON employs a heuristic that we call *the least number heuristic* (LNH). The basic idea is that, in the early stages of the search, only a few numerals are used, which have some distinguished properties. The remaining numerals have an equal status. Thus, when choosing a value for the next cell, we need to consider only one representative of those "unused" numerals.

The new model searching procedure, which employs the LNH, is described in Figure 3. It has an additional parameter, mdn (for "maximal designated number"), which is an integer. It is initialized to (-1). The intuitive meaning of this variable is as follows. The set D_n is divided into two disjoint subsets, $[0\ldots mdn]$ and $[(mdn+1)\ldots(n-1)]$. The numerals in the first subset have already been used in the search, while the numerals in the second subset have not yet been used. When choosing values for a new cell, we consider every numeral in the first subset; but in the second subset, we consider only one numeral – the smallest one. In the following we prove the correctness of the procedure `SatE_1`.

DEFINITION. A set G of ground equations is *symmetric* with respect to a set N of numeral constants if G remains the same under any permutation of the numeral constants in N.

```
bool SatE_1(Pmod,mdn)
{
    assignment_propagate(Pmod);
    if contradiction discovered return(FALSE);
    if all cells have been assigned values return(TRUE);
    choose a cell f(e_1, ..., e_k) which does not have a value;
    m = max(e_1, ..., e_k, mdn);
    for i = 0 to m do
        if SatE_1(Pmod ∪ {f(e_1, ..., e_k) = i},m) return(TRUE);
    if (m < n − 1)
        if SatE_1(Pmod ∪ {f(e_1, ..., e_k) = m + 1},m + 1) return(TRUE);
    return(FALSE);
}
```

Fig. 3. Finding a finite model with the LNH.

PROPOSITION. *The set $GI(E,n)$ is symmetric with respect to D_n.*

Proof. Straightforward from the definition of $GI(E,n)$ and from the assumption that no numeral appears in any equation of E.

LEMMA. *At each call of the procedure* `SatE_1`, *Pmod is symmetric with respect to* $[(mdn+1)\ldots(n-1)]$.

Proof. Initially, $mdn = -1$ and $Pmod = \emptyset$. So the lemma holds. The procedure `assignment_propagate()` preserves the symmetry. At any subsequent call of `SatE_1`, when $Pmod$ is extended by a new assignment, the value of mdn is updated accordingly.

THEOREM. *A model exists if and only if* `SatE_1`$(\emptyset,-1)$ *returns TRUE.*

Proof. The 'if' part is obvious. Now suppose there is a model that can be found by the procedure `SatE_0`. The difference between `SatE_0` and `SatE_1` is that the latter may consider fewer possibilities than the former, when trying to assign a value to the new cell. When $m+1 \geqslant n-1$, the two procedures consider the same set of values. Suppose $m+1 < n-1$, and `SatE_0` returns TRUE, by extending $Pmod$ with the new assignment $f(e_1,\ldots,e_k) = j$, where $m+1 < j < n$. Then by a permutation on numerals that exchanges j with $m+1$, `SatE_1` can also extend $Pmod$ with the assignment $f(e_1,\ldots,e_k) = m+1$.

With the least number heuristic, we need not consider many branches at the upper levels of the search tree. Thus the efficiency of the model finding process is increased. Note that the LNH is a restriction strategy, in that it prohibits assigning certain values to the new cell. The performance of the search procedure is also affected by the order in which the new cells are selected, even when the

heuristic is used. The numbers can be used up quickly if the order is inappropriate, for example, when the cells are chosen row by row or column by column. In FALCON, the search proceeds by increasing the size of the subalgebra each time by one. To be specific, let C_i be the set of cell terms whose maximum argument is i. Then we consider the cells in the set C_0, C_1, C_2, ..., successively. So, for example, if there is one binary function symbol f and one unary function symbol g, then the cells are selected in an order like the following one:

$$
\begin{aligned}
& f(0,0) \rightarrow g(0) \rightarrow \\
& f(0,1) \rightarrow f(1,0) \rightarrow f(1,1) \rightarrow g(1) \rightarrow \\
& f(0,2) \rightarrow f(2,0) \rightarrow f(1,2) \rightarrow f(2,1) \rightarrow f(2,2) \rightarrow g(2) \rightarrow \\
& \quad \ldots
\end{aligned}
$$

Now we give an example to show how FALCON finds a finite model.

EXAMPLE 1 (continued). Suppose we wish to find all four-element models of the theory T_1. By default, the least number heuristic is used. The details of the search process are described as follows:

1. Initially, $mdn = -1$. Although $Pmod = \emptyset$, we can still obtain the four assignments
   $$f(0,0) = 0, \qquad f(1,1) = 1, \qquad f(2,2) = 2, \qquad f(3,3) = 3$$
   which are instances of the simple Equation (A1).
2. Next we choose the cell $f(0,1)$, whose value can be 0 or 1 or 2. (We do not consider the possibility of assigning 3 to $f(0,1)$, since $m = 1$.) Either $f(0,1) = 0$ or $f(0,1) = 1$ leads to a contradiction. Let $f(0,1) = 2$ (and mdn becomes 2). As the result of assignment propagation, we get $f(2,0) = 1$, $f(1,2) = 0$.
3. The cell $f(1,0)$ may take the value 0, 1, 2, or 3. Either $f(1,0) = 0$ or $f(1,0) = 1$ leads to a contradiction. Let $f(1,0) = 2$, and we get: $f(2,1) = 0$, $f(0,2) = 1$.
4. The cell $f(0,3)$ may take the value 0, 1, 2, or 3. But in each case, contradiction occurs.
5. Go back and change the value of $f(1,0)$ to 3. The assignments $f(2,1) = 0$ and $f(0,2) = 1$ are removed from the partial model, because they are consequences of the old assignment $f(1,0) = 2$. Propagation of $f(1,0) = 3$ results in two new assignments, namely, $f(3,1) = 0$ and $f(0,3) = 1$.
6. Through assignment propagation, we know that the cell $f(0,2)$ cannot take the value 0, 1, or 2. Let $f(0,2) = 3$. Then we can deduce that $f(3,0) = 2$ and $f(2,3) = 0$.
7. Similarly, let $f(2,1) = 3$, and get $f(3,2) = 1, f(1,3) = 2$. Now a complete model is found, which is the one given in Section 2.1 (or the first model shown below).

```
int SatE_2(Pmod,mdn)
{
    assignment_propagate(Pmod);
    if contradiction discovered return(0);
    if all cells have been assigned values return(1);
    choose a cell f(e1, ..., ek) which does not have a value;
    m = max(e1, ..., ek, mdn);
    sum = 0;
    for i = 0 to m do
        increase sum by SatE_2(Pmod ∪ {f(e1, ..., ek) = i},m);
    if (m < n − 1)
        increase sum by
        (n − 1 − m) * SatE_2(Pmod ∪ {f(e1, ..., ek) = m + 1}, m + 1);
    return(sum);
}
```

Fig. 4. Counting the number of finite models with the LNH.

When the LNH is used, only one model is generated. But if we choose *not* to use the heuristic, FALCON will produce the following two models.

f	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

f	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

These two models are isomorphic under the mapping

$$0 \mapsto 0, \qquad 1 \mapsto 1, \qquad 2 \mapsto 3, \qquad 3 \mapsto 2.$$

The least number heuristic can also be used in counting the number of finite models. Such a procedure is given in Figure 4. Its correctness proof is similar to that of `SatE_1`.

4. Implementation and Applications

4.1. The System

Given a set E of equational axioms and a positive integer n, FALCON attempts to find an n-element model of E (or several such models or all the models). Usually one is more interested in models having some specific properties, such as making another equation false. For convenience, FALCON also accepts inequalities of the form $t_1 \; != \; t_2$, where t_1 and t_2 are two terms.* Such an inequality means, for

* In general, additional programming is required in order to select some specific model from the generated ones.

all the variables in t_1 and t_2, the values of the two terms differ. So for example, $f(x,y)\ !=g(y)$ is the same as $\forall x \forall y\,(f(x,y) \neq g(y))$. From now on, we shall say the specification consists of a set of (unit) *clauses* rather than equations.

The system encourages the use of numeral constants in the specification. This often helps to find the model more quickly. The numerals should be as small as possible, so that the least number heuristic can be applied most effectively. When the clauses contain numerals, the variable *mdn* in the procedure `SatE_1` is initialized to the maximum numeral appearing in the specification.

EXAMPLE 2. To find noncommutative groups, one may give the following set of clauses:

```
f(x,0) = x
f(0,x) = x
f(x,g(x)) = 0
f(g(x),x) = 0
f(f(x,y),z) = f(x,f(y,z))
f(a,b) != f(b,a)
```

Here f and g denote the multiplication and inverse operations, respectively; a and b are two constant symbols. The identity element is denoted by 0. The last clause may be replaced by $f(1,2)\ !=\ f(2,1)$, if one notices that a must be different from b, and that neither a nor b can be 0.

FALCON was implemented in C. There are about 2000 lines of source code. The implementation of the system differs only slightly from the abstract procedures given in the preceding section. During instantiation of the equational axioms, a simple equation is turned directly into a set of assignments. For the purpose of efficiency, we used a backtracking procedure instead of the recursive procedures. During the search, when both sides of an inequality are reduced to the same numeral, we consider it as a contradiction. When a new cell is chosen, we select the "good" candidate values by assignment propagation. (A value is "bad" if assigning it to the cell will result in a contradiction.) When a clause evaluates to TRUE in the current partial model, it will not be used subsequently unless backtracking occurs and the partial model becomes smaller.

4.2. Examples of Applications

FALCON has found applications in abstract algebra and formal logic. It is very easy for the program to solve such problems as showing the independence of one axiom of ternary Boolean algebra and proving that all groups of order 7 are commutative [17, 19]. FALCON also solved some previously open questions. In combinatory logic, we proved that neither the fragment $\{Q, L\}$ nor the fragment $\{B, N_1\}$ has the strong fixed point property, by finding appropriate models as

counterexamples [18, 21, 22]. On Tarski's High School Identity problem [2], we showed that there is no Gurevič algebra of size 7.

In the following, we shall give some additional examples of FALCON's applications.* For each problem, we give the clauses, the size of the model, and FALCON's execution time on a SPARCstation 2.

EXAMPLE 3 (posed by R. Padmanabhan). Find a model of the following set of clauses:

```
F(x,x) = x
G(x,x) = x
H(x,x) = x
F(x,y) = F(y,x)
G(x,y) = G(y,x)
H(x,y) = H(y,x)
F(x,F(y,z)) = F(F(x,y),z)
G(x,G(y,z)) = G(G(x,y),z)
H(x,H(y,z)) = H(H(x,y),z)
F(G(F(x,y),z),G(y,z)) = G(F(x,y),z)
G(F(G(x,y),z),F(y,z)) = F(G(x,y),z)
G(H(G(x,y),z),H(y,z)) = H(G(x,y),z)
H(G(H(x,y),z),G(y,z)) = G(H(x,y),z)
F(H(F(x,y),z),H(y,z)) = H(F(x,y),z)
H(F(H(a,b),c),F(b,c)) != F(H(a,b),c)
```

Here F, G, and H are (ACI) function symbols; a, b, and c are constants; and x, y, and z are universally quantified variables. FALCON found one model of size 4, another of size 5. The execution times are 7.45 seconds and 3.97 seconds, respectively. Thus we know that the axiom $H(F(H(x,y),z),F(y,z)) = F(H(x,y),z)$ is independent of the rest.

EXAMPLE 4. Find a finite algebra that satisfies all the axioms of a commutative ring with unit, except the associativity law.

```
s(x,0) = x
s(x,m(x)) = 0
s(x,y) = s(y,x)
s(x,s(y,z)) = s(s(x,y),z)
p(x,1) = x
p(1,x) = x
p(x,y) = p(y,x)
s(p(x,y),p(x,z)) = p(x,s(y,z))
s(p(x,z),p(y,z)) = p(s(x,y),z)
p(a,p(b,c)) != p(p(a,b),c)
```

* The problems in Example 3 and Example 4 were communicated by Stan Burris.

Here, m, s, and p denote minus, sum, and product, respectively. FALCON found an eight-element model in 350.98 seconds, which shows the independence of the associativity law.

EXAMPLE 5. A ring is Boolean if $x^2 = x$ for all x. MacHale [8] proved that, if a ring satisfies the identities $2x = 0$ and $x^{2^n+1} = x$ for a fixed integer $n > 1$, then it is Boolean. It is interesting to investigate, for other odd integers $m > 1$, whether the conditions $2x = 0$ and $x^m = x$ force a ring to be Boolean. The smallest such integer is 7. So let us find a non-Boolean ring satisfying the identities $x^7 = x$ and $2x = 0$, for all x.

```
s(x,0) = x
s(x,m(x)) = 0
s(x,y) = s(y,x)
s(x,s(y,z)) = s(s(x,y),z)
p(x,p(y,z)) = p(p(x,y),z)
s(p(x,y),p(x,z)) = p(x,s(y,z))
s(p(x,z),p(y,z)) = p(s(x,y),z)
a != p(a,a)
s(x,x) = 0
p(x,p(p(x,x),p(p(x,x),p(x,x)))) = x
```

FALCON found the following four-element model in 0.20 seconds.

```
 s | 0 1 2 3        p | 0 1 2 3
---+---------      ---+---------
 0 | 0 1 2 3        0 | 0 0 0 0
 1 | 1 0 3 2        1 | 0 2 3 1
 2 | 2 3 0 1        2 | 0 3 1 2
 3 | 3 2 1 0        3 | 0 1 2 3

 m | 0 1 2 3        a = 1
---+---------
   | 0 1 2 3
```

From this model, we conclude that, for any fixed positive integer $m = 6k+1 (k > 0)$, the identities $2x = 0$ and $x^m = x$ do not imply that a ring is Boolean.

EXAMPLE 6. Find nontrivial gyrogroups. A gyrogroup has three binary operators: m (multiplication), ld (left division), and rd (right division). The axioms are the following nine equations. (All the variables w, x, y, and z are universally quantified.)

```
rd(m(x,y),y) = x
m(rd(x,y),y) = x
```

```
ld(y,m(y,x)) = x
m(y,ld(y,x)) = x
rd(x,x) = ld(y,y)
m(w,m(m(z,x),z)) = m(m(m(w,z),x),z)
m(rd(m(m(z,x),y),m(x,y)),y) = rd(m(z,m(y,x)),x)
m(m(y,x),m(y,x)) = m(m(m(x,y),y),x)
m(rd(m(m(w,y),x),m(y,x)),rd(m(m(z,y),x),m(y,x)))
    = rd(m(m(m(w,z),y),x),m(y,x))
```

It is known that Abelian groups are gyrogroups. The challenge is to find finite nontrivial gyrogroups that are not Abelian groups. The existence of such models had been open for a long time. Only recently have mathematicians found some models of cardinality 8, using some deep mathematical methods [15]. Jonathan Smith asked whether bigger models exist. It was not difficult for FALCON to find one such model of size 8 and another of size 16. The execution times are 4.40 and 127.12 seconds, respectively. The multiplication table of the 16-element model is as follows.

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	15	14	12	13
3	3	2	1	0	7	6	5	4	11	10	9	8	14	15	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	11	10	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	10	11	9	8
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	7	6	4	5
11	11	10	9	8	15	14	13	12	3	2	1	0	6	7	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	3	2	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	2	3	1	0

The operation tables of ld and rd can be derived from that of m. In my experiment, the input specification consisted of all the axioms except the last one. A little additional code was written to check whether any model of the first eight axioms satisfies the ninth axiom but does not satisfy the associativity law $m(x, m(y, z)) = m(m(x, y), z)$.

EXAMPLE 7. Symmetric modes. A mode [11] is an algebra with one binary operation f, whose axioms are the following:

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

A mode is n-symmetric if it satisfies the equation

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

where f is repeated n times. For various values of n, FALCON produced some small n-symmetric modes, which helped Anna Romanowska and her colleagues make the following conjecture [10]:

> For a prime number p, if a p-symmetric mode has fewer than p elements, then it is left-zero, namely, satisfying the identity $f(x, y) = x$.

The conjecture was proved later.

In summary, model-finding programs like FALCON can be used to prove that one axiom is independent of other axioms or that some conjecture does not follow from a set of axioms. Finite models also help us to understand new theories. Thus model generators play a complementary role to that of conventional theorem provers in the study of abstract theories. In addition, they can show the existence or nonexistence of certain mathematical entities that are interesting in themselves.

Some of the above-mentioned problems can also be solved by other model generators like FINDER [12], MGTP [5], MACE [9], and ModGen [6]. But on other problems, it seems not easy for those programs to reproduce the results obtained by FALCON. In the next section, we shall compare FALCON with various methods and tools for model generation.

5. Related Work

Several different approaches exist for generating models with computer programs. Winker [17] used a conventional general-purpose theorem prover; much human guidance was required in his experiments. Tammet [16] described a resolution-based method for building finite models. But it cannot handle formulas with equality predicates. Bourely, Caferra, and Peltier [1] implemented a prototype system based on the theorem prover OTTER. One of its features is that infinite models may be generated. Little is known, however, about its performance in the finite case.

Some recent systems, such as FINDER [12] and MGTP-G [5, 14], rely on search techniques to find finite models automatically and efficiently. Decision procedures for the propositional logic may also be used to find finite models in the first-order case [6, 7, 9, 20]. For example, the programs DDPP, SATO, and MACE, which are based on the Davis–Putnam procedure [3], have been used to determine the existence of quasigroups satisfying certain identities.

Among the various methods and systems, FINDER [12] shares many characteristics with FALCON.* Both programs are based on backtracking procedures and can find only finite models. In addition, both of them are based on first-order ground formulas. However, FALCON employs the least number heuristic to eliminate isomorphism in the search space, whereas FINDER does not have such a built-in mechanism. One feature of FINDER is that it deduces new clauses during the search, so that it never backtracks twice for the same reason [12, 14]. When there are too many clauses in its database, FINDER divides the search space into several disjoint subspaces and searches them separately, repeating the preprocessing routine with each one. In contrast, FALCON does not generate any new clauses. During its search process, the ground equations can only become simpler or even disappear.

TABLE I. Generating finite algebras with FINDER and MACE

Problem		MACE			FINDER			
Instance	M	b	t_0	t_1	s	b	t_0	t_1
AG.4	4	3	0.36	0.01	1	7	0.02	0.05
.5	6	5	1.31	0.13	1	59	0.02	0.37
.6	60	59	4.08	2.42	1	980	0.03	8.92
.7	120	143	10.36	18.42	59	23303	1.13	234.07
.8	1920	2050	23.25	469.98				+
NG.4	0	1	0.36	0.01	1	6	0.02	0.03
.5	0	2	1.32	0.04	1	45	0.02	0.50
.6	18	18	4.06	1.07	5	786	0.12	12.83
.7	0	120	10.26	3.04	75	25900	1.60	529.03
.8	480	1110	23.35	174.93				+
.9	0		*					
RU.4	6	5	3.83	0.12	1	13	0.03	0.43
.5	6	5	18.36	1.28	1	182	0.03	5.97
.6	24	34	65.28	18.26	236	79359	5.53	2072.70
.7	120		*					+
LT.4	3	2	0.67	0.02	1	0	0.02	0.00
.5	19	18	2.52	0.13	1	4	0.03	0.05
.6	213	212	7.39	3.20	1	82	0.03	1.03
.7	3761	3768	19.09	125.39	1	897	0.05	26.67
BA.4	1	0	3.54	0.14	1	4	0.02	0.03
.5	0	4	17.59	1.23	1	660	0.03	0.97
.6	0		*					+

* It should be noted that FINDER is a general-purpose model generator that accepts many-sorted clauses, while FALCON only allows one-sorted unit clauses (i.e., equations and inequalities).

TABLE II. Generating finite algebras with FALCON

Problem		Using LNH			LNH Not Used		
Instance	t_0	m	b_1	t_1	M	b_2	t_2
AG.4	0.00	3	4	0.05	4	6	0.08
.5	0.00	1	7	0.22	6	26	0.67
.6	0.01	6	13	0.68	60	126	6.07
.7	0.02	1	20	1.97	120	828	64.35
.8	0.03	15	33	4.73	1920	6690	741.27
NG.4	0.00	0	4	0.05	0	4	0.05
.5	0.01	0	6	0.17	0	12	0.32
.6	0.02	3	7	0.48	18	42	2.38
.7	0.02	0	11	1.30	0	264	22.40
.8	0.03	4	20	3.37	480	2132	287.72
.9	0.03	0	31	7.70			+
RU.4	0.02	5	7	0.25	6	9	0.35
.5	0.03	1	9	1.27	6	29	3.40
.6	0.06	1	15	3.50	24	158	32.50
.7	0.07	1	22	9.43	120	1390	461.50
LT.4	0.01	3	3	0.01	3	3	0.01
.5	0.02	19	19	0.15	19	19	0.15
.6	0.02	194	194	3.28	213	213	3.60
.7	0.03	2831	2836	79.43	3761	3779	108.48
BA.4	0.01	1	1	0.08	1	1	0.08
.5	0.02	0	3	0.27	0	5	0.45
.6	0.02	0	9	0.90	0	33	3.05
.7	0.05	0	13	2.68	0	163	25.23
.8	0.07	4	18	6.65	120	1271	336.53

5.1. EXPERIMENTS

Now we give some experimental data comparing FALCON with two of the above-mentioned systems: FINDER (version 3.0.1) and MACE (version 1.0.0). Both of them were implemented in C and can accept equational clauses.

Table I and Table II show the performances of the three programs on generating some common finite algebras: Abelian groups (AG), noncommutative groups (NG), rings with unit (RU), lattices (LT), and Boolean algebras (BA). Table III shows their performances on finding counterexamples (which can refute some conjectures). The programs are tested on the following problems:

1. Group Theory
   - GRP: Find a noncommutative group satisfying $(xy)^4 = x^4y^4$.
2. Ring Theory

- RNA: Show that the associative law is independent. (Example 4.)
- RNB: Find a non-Boolean ring satisfying the clauses in Example 5.

3. Lattice Theory
   - LNM: Find a lattice which is not modular.
4. Combinatory Logic. Show that some fragments do not have the strong fixed point property.
   - CL1: the fragment $\{ Q, L \}$ (with $Q = 0, L = 1$)
   - CL2: the fragment $\{ B, N_1 \}$ (with $B = N_1 = 0$)
   - CL3: the fragment $\{ B, N_1 \}$ (with $B = 0, N_1 = 1$).

For each problem, the same set of clauses was given to all the programs. (In our experiments, the most natural specifications were used.) The execution times are measured in seconds. The data were obtained on a SPARCstation 2.

TABLE III. Finding counterexamples

Problem	Model	MACE		FINDER		FALCON		
Instance	exists?	t_0	t_1	t_0	t_1	t_0	t_1	t_2
GRP.4	N	40.13	0.09	0.02	0.05	0.00	0.08	0.08
.5	N	307.11	2.05	0.02	1.80	0.01	0.27	0.48
.6	N	*		0.63	499.15	0.02	0.65	3.05
.7	N				+	0.02	1.53	27.00
.8	Y				+	0.03	0.58	1.10
RNA.4	N	3.93	0.51	0.02	1.57	0.01	3.82	7.88
.5	N	19.34	4.30		+	0.02	24.72	164.33
.6	N	*				0.05	107.12	2851.70
.7	N					0.07	562.97	+
.8	Y					0.10	350.87	+
RNB.4	Y	3.70	0.11	0.02	3.30	0.02	0.20	0.25
.5	N	17.68	0.65	0.02	0.22	0.03	0.68	21.20
.6	N	64.28	3.55	0.03	2.08	0.06	1.67	326.57
.7	N	*		0.03	3.55	0.08	3.37	+
.8	Y	*			+	0.12	4.22	6.75
LNM.4	N	1.82	0.03	0.02	0.05	0.00	0.03	0.07
.5	Y	8.18	0.13	0.02	0.15	0.02	0.23	0.78
.6	Y	28.80	0.80	0.03	1.07	0.03	0.45	17.48
CL1.4	Y	9.00	1.70	0.02	1.55	0.00	0.12	0.20
.5	Y	51.74	2.74	0.10	51.03	0.02	0.20	0.30
CL2.4	Y	48.82	2.27	0.02	3.17	0.00	0.18	0.20
.5	Y	*		0.07	78.82	0.02	0.50	0.65
CL3.4	N	48.70	55.10	0.08	55.60	0.00	28.67	1670.52
.5	Y	*		0.05	86.72	0.02	0.92	14.83

In the three tables, the first entry in each row is of the form $name.n$ or simply $.n$, where $name$ is the name of the problem, and n is the size of the model to be searched for. The other columns provide the following information:

- Total number of models found, when the LNH is used (m), and when no isomorphism rejection method is used (M).
- Execution times: preprocessing time (t_0); search time (t_1 or t_2).
- Search space. Different programs may produce different types of statistics about the search space. For MACE and FALCON, we give the number of branches of the search tree (b, b_1 or b_2).* For FINDER, we give the number of backtracks (b) as well as the number of subspaces created (s).

Thus for FINDER and MACE, the execution time for each problem is given by $(t_0 + t_1)$, and M models are found. For FALCON, when it is executed with the LNH turned off, the time is given by $(t_0 + t_2)$, the search tree has b_2 branches, and M models are found; otherwise (i.e., when the LNH is used), the time is $(t_0 + t_1)$, the search tree has b_1 branches, and only m models are found. In the tables, '$*$' indicates that the program runs out of memory, and '$+$' indicates that the execution time of the program exceeds one hour.

5.2. Comparison and Discussion

From the three tables, we can see that FALCON's overall performance is very good and that the LNH is quite effective for eliminating isomorphism. For example, FALCON finds only one Abelian group of order 7, while the other programs find 120 models that are isomorphic to each other. The heuristic is crucial when finding large models and when there are only a few models.

Isomorphism rejection also played a key role in the solution of previously open questions on the existence of certain quasigroups [4, 14]. The method used in those experiments is to add some additional clauses to the problem specifications. This is not so convenient and is not applicable to many problems considered in this paper.

As mentioned earlier, there are some similarities between FALCON and FINDER. But the lack of a built-in isomorphism rejecting mechanism makes it difficult for FINDER to find nontrivial models. Even when the LNH is not used, FALCON is more efficient than FINDER on many problems. One reason is that there are too many backtracks for FINDER to remember. So the database grows rapidly. As a consequence, many subspaces have to be created, and much repetitive work has to be done. The situation is better when the program solves the lattice problems, whose constraints can be easily satisfied.

We should remark that FINDER is quite flexible, providing the user with many options. But in our experiments, we did not take much advantage of this

* In FALCON, the detection of a bad value for a new cell (through assignment propagation) is not counted as a branch. So, for instance, the search tree of $T1.4$ (in Example 1) has 2 branches.

flexibility to write "efficient" specifications. A few exceptions are: when a binary function is commutative, we declared it as `'commutative'`, instead of writing an axiom like $f(x,y) = f(y,x)$; and on the combinatory logic problems, we used the setting `'change-order -T'` on the binary function.

Now we make some comparison between first-order model generators and propositional satisfiability testing programs like MACE [9] and ModGen [6]. From the above tables, one can see that, MACE tends to run out of memory when the size of the model is moderately large (say, greater than 6 or 7) and when there are complicated axioms with deeply nested terms or containing several variables. When memory is not a problem, MACE is very fast. But a large portion of its time is spent on preparing the clauses. The reason for these difficulties is that, in most cases, a lot of propositional variables and clauses are required to represent a first-order finite model generation problem. For a specific example, suppose the size of the model is n. Then a binary function symbol corresponds to n^3 propositional variables; and if the function is associative, we need n^6 clauses. Thus, in general, it is not appropriate to transform first-order model generation problems to propositional satisfiability problems.

The strength or weakness of a particular approach is obviously related to the problems it is used to solve. For instance, FINDER 3 can solve nontrivial quasigroup problems with which FINDER 2 has much difficulty; but FINDER 2 is more efficient on some other problems [12]. Many existing benchmarks (like the n-queens problem and the quasigroup problems) contain such constraints that, once a cell takes some value, a number of other cells cannot take that value. From the syntactic point of view, the corresponding axioms are quite simple. In contrast, the axioms in many equational problems are more complicated. Usually the value of one cell does not have a direct relationship with that of another cell. Rather, a number of (dynamically changing) cells are interrelated. Another characteristic of FALCON's applications is that positive equalities play a dominant role. The inequalities are used only for testing partial models.

6. Concluding Remarks

In this paper, we described the program FALCON, which constructs finite models of equational theories. Experimental results indicate that it is quite efficient, compared with similar systems. Several examples were given to demonstrate the feasibility of finite model generation and its potential applications.

In our approach, model generation is viewed as constraint satisfaction, and backtracking search is used to solve the problem. We choose to work with ground equations. Thus, it is not necessary to do unification, which is quite time consuming; yet for many applications, memory is not a serious issue.

We proposed the least number heuristic for eliminating isomorphic partial models. It is a very simple restriction strategy, but it is quite powerful. In many cases, it can reduce the size of the search space significantly, as shown by our

experiments. As a built-in mechanism, it relieves the user from writing fancy specifications. The heuristic is also very general and can be applied to many problems. It may be adopted in other model-searching systems. However, isomorphism rejection still deserves further study. (As a motivating example, let us mention that FALCON generates 6 Abelian groups of order 6. Each of them is isomorphic to the cyclic group.)

When implementing FALCON, we did not consider any strategies for directing the search (e.g., heuristics for selecting the next cell). Nor did we pay much attention to the efficiency of assignment propagation. Considerable work can be done to improve the performance of the system. Currently, FALCON accepts only equations and inequalities. As part of our future research, we intend to build systems for first-order and many-sorted theories. We also plan to investigate more applications of such systems.

Acknowledgements

This work was done while I was a student at the Institute of Software, Academia Sinica, Beijing, supervised and supported by C. S. Tang. The paper was written during my visit at the University of Iowa, which was sponsored by Hantao Zhang. I am grateful to Yunmei Dong, Huimin Lin, and Larry Wos for their encouragement; to Joel Berman, Stan Burris, Anna Romanowska, and Jonathan Smith for their interest in FALCON. Larry Wos, Hantao Zhang, and the anonymous reviewers made helpful comments on earlier versions of this paper.

References

1. Bourely, C., Caferra, R., and Peltier, N.: A method for building models automatically: Experiments with an extension of OTTER, in *Proc. 12th Int. Conf. on Automated Deduction* (*CADE-12*), *LNAI* 814, Springer, 1994, pp. 72–86.
2. Burris, S. and Lee, S.: Tarski's high school identities, *Amer. Math. Monthly* **100** (1993), 231–236.
3. Davis, M. and Putnam, H.: A computing procedure for quantification theory, *J. ACM* **7** (1960), 201–215.
4. Fujita, M., Slaney, J., and Bennett, F.: Automatic generation of some results in finite algebra, in *Proc. 13th Int. Joint Conf. on Artificial Intelligence* (*IJCAI-93*), 1993, pp. 52–57.
5. Hasegawa, R., et al.: MGTP: A parallel theorem prover based on lazy model generation, in *Proc. 11th Int. Conf. on Automated Deduction* (*CADE-11*), *LNAI* 607, Springer, 1992, pp. 776–780.
6. Kim, S. and Zhang, H.: ModGen: Theorem proving by model generation, in *Proc. AAAI-94*, Seattle, 1994, pp. 162–167.
7. Lee, S.-J. and Plaisted, D. A.: Problem solving by searching for models with a theorem prover, *Artificial Intelligence* **69** (1994), 205–233.
8. MacHale, D.: A remark on Boolean rings, *Mathematics Magazine* **63** (1990), 248–249.
9. McCune, W.: A Davis–Putnam Program and Its Application to Finite First-Order Model Search: Quasigroup Existence Problems, Tech. Memo ANL/MCS-TM-194, Argonne National Laboratory, Argonne, IL, 1994.
10. Romanowska, A. B.: Private Communication, 1994.
11. Romanowska, A. B. and Smith, J. D. H.: *Modal Theory: An Algebraic Approach to Order, Geometry, and Convexity*, Heldermann Verlag, Berlin, 1985.

12. Slaney, J.: FINDER: Finite Domain Enumerator. Version 3.0 Notes and Guide, Australian National University, 1993.
13. Slaney, J.: SCOTT: A model-guided theorem prover, in *Proc. 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)*, 1993, pp. 109–114.
14. Slaney, J., Fujita, M. and Stickel, M.: Automated reasoning and exhaustive search: Quasigroup existence problems, *Computers and Mathematics with Applications* **29**(2) (1995), 115–132.
15. Smith, J. D. H.: Private Communication, 1994.
16. Tammet, T.: Using resolution for deciding solvable classes and building finite models, *Baltic Computer Science: Selected Papers, LNCS* 502, Springer (1991), 33–64.
17. Winker, S.: Generation and verification of finite models and counterexamples using an automated theorem prover answering two open questions, *J. ACM* **29** (1982), 273–284.
18. Wos, L.: The kernel strategy and its use for the study of combinatory logic, *J. Automated Reasoning* **10** (1993), 287–343.
19. Wos, L.: *Automated Reasoning: 33 Basic Research Problems*, Prentice Hall, Englewood Cliffs, NJ, 1988.
20. Zhang, H., and Stickel, M.: Implementing the Davis–Putnam Algorithm by Tries, Technical Report, University of Iowa, 1994.
21. Zhang, J.: Search for models of equational theories, in *Proc. 3rd Int. Conf. for Young Computer Scientists (ICYCS-93)*, Beijing, 1993, pp. 2.60–63.
22. Zhang, J.: Problems on the generation of finite models, in *Proc. 12th Int. Conf. on Automated Deduction (CADE-12)*, *LNAI* 814, Springer, 1994, pp. 753–757.