System Description
Generating Models by SEM*

Jian Zhang** and Hantao Zhang

Department of Computer Science
The University of Iowa
Iowa City, IA 52242, USA

{jizhang, hzhang}@cs.uiowa.edu

1 Introduction

SEM is a System for Enumerating finite Models of first-order theories, developed at the University of Iowa during the past two years. Given a theory defined by a set of axioms in multi-sorted first-order logic, SEM tries to find models of the theory. The models are given in the form of definition tables of the function symbols appearing in the axioms.

In the recent years, several model generation programs have been constructed. Some of them are based on first-order reasoning (e.g., SATCHMO [5] and MGTP [1]); some of them are based on constraint satisfaction (e.g., FINDER [7] and FALCON [10]); some of them are based on the propositional logic (e.g., ModGen [3] and MACE [6]). Like FINDER and FALCON, SEM uses the general constraint satisfaction techniques but is much more efficient than both FINDER and FALCON on most model generation problems.

2 General Information about SEM

SEM is implemented in C. The YACC tool is used for parsing. The current version has about 5000 lines of code. It can be compiled on Unix workstations, such as SPARCstations, SGI IRIX, HP 9000 and IBM RS/6000. The source code and related documents are available at the following address:

`ftp://ftp.cs.uiowa.edu/pub/hzhang/sem`

In SEM, the input specification for a problem consists of the following three parts:

Sorts; *Functions*; *Clauses*.

Sorts specify the size of each sort; *Functions* specify the signatures of each function. For the convenience of the user, the boolean functions and some conventional arithmetic functions are built into the system. The axioms are expressed

* Partially supported by the National Science Foundation under Grants CCR-9504205 and CCR-9357851.

** On leave from the Software Institute, Academia Sinica, Beijing 100080, P.R.China

as *Clauses*. They usually contain some variables which are assumed to be universally quantified over the given finite domains. The goal is to find a definition of the functions that satisfy the axioms.

As an example, we give the specification for the pigeonhole problem.

Example Pigeonhole problem (30 pigeons, 29 holes). (The character '%' starts a comment in a line.)

```
% Sorts
( pigeon [30] )
( hole [29] )

% Functions
{ h  : pigeon -> hole }
{ in : pigeon hole -> BOOL }

% Clauses.  '-' is 'not'; '|' is 'or'.
[ in(x,h(x)) ]
[ -in(x,z) | -in(y,z) | EQ(x,y) ]
```

SEM has some command-line options, including the following,

```
-e:  echo the input specification
-l:  do not use the LNH to eliminate isomorphism
-p:  do not print the models
-r:  print the execution sequences
-c<integer>:  cell selection strategy (0 .. 2)
-m<integer>:  maximum number of models
-t<integer>:  maximum execution time (in hours)
-f <filename>: load an unfinished job
```

By default, the program tries to find one model of the given problem, using the least number heuristic (LNH). If the model exists, it is printed.

3 The Algorithm and Data Structures

The basic mechanism of SEM is similar to that of FALCON [10] and FINDER [7]. That is, model generation is treated as a constraint satisfaction problem (CSP) and is solved by backtracking search techniques. To be specific, the variables of the CSP are the *cells* (i.e., ground terms like $f(0,1)$, $g(2)$) in the operation tables of the functions. The constraints of the CSP are the set of *ground* instances of the clauses obtained by substituting each element of a given domain for every variable in the clauses. Our goal is to find a set of assignments (e.g., $f(0,1) = 2$) such that the set of ground clauses hold.

The overall process of model finding can be described as a search tree. Each inner node corresponds to a cell. An edge represents the assignment of some value to the cell. The effects of the assignment are propagated by simplifying (or rewriting) the related constraints. SEM also provides an option for propagating negative assignments (i.e., clauses like $f(0,1) \neq 2$).

SEM employs several strategies to reduce the search space. When there are a number of cells whose values are not yet known, it chooses the cell which has the smallest number of possible values. This "first-fail" principle is combined with some form of forward checking, to make the search efficient. In addition, the program takes advantage of symmetries which are inherent in many applications, and uses the so-called "least number heuristic" (LNH) to avoid producing isomorphic models. (See [10] for more about LNH.)

The main data structures in the program are the cell tables and clauses. As usual, a clause is a list of literals, and a term is a tree. Each cell is associated with a list of all clauses in which it occurs. To improve the efficiency of constraint propagation and backtracking, we adopt some techniques which are similar to those of LDPP and SATO [9]. For example, there is a counter for each clause, indicating the number of active literals. For more details about the data structures and algorithms, see [11].

4 Performance and Applications

As a model finding tool, SEM can be used in AI problem solving, as well as in constructing certain combinatorial structures. In addition, it may serve as a companion to theorem provers, because it can produce counterexamples to false conjectures.

Example. Show that the equation

$$f(z, f(g(f(y,z)), f(g(f(y, g(f(y,x)))), y))) = x$$

is not a single axiom for group theory. (See [4].) Here f and g denote the multiplication and inverse operations, respectively. To do this, we add the formula

$$f(0, g(0)) \neq f(1, g(1))$$

which implies that a structure is not a group. With these two axioms, SEM can find a 4-element model in 1.85 seconds on a SGI IRIX 5.3 workstation.

In Table 1, we describe SEM's performance on some well-known test problems. The table gives the program's running times (in seconds) on a SGI workstation (IRIX 5.3) for finding the first model (if it exists).

SEM obtained several nontrivial results, some of which seem quite difficult for other systems to reproduce. For example, it completed the search for Gurevič algebras of size 8. More such results were reported in [11]. In addition, the search for QG7.16 was completed. (No model was found.) We also tried to use SEM to show the consistency of the theory TRC [2], but failed to find any model of size up to 9. In the future, we shall use SEM to solve more problems in logic and algebra.

Problem	size	time
Pigeon-hole	30 pigeons, 29 holes	0.93
Queens	8 queens	0.31
Jobs	8 jobs, 4 people	0.14
Groups	12 elements	8.35
Boolean-algebra	12 elements	1.18
Boolean-algebra	16 elements	0.65
QG5	11 elements	0.38
QG5	13 elements	27.14

Table 1. Performance of SEM

5 Comparison with Other Systems

In general, it is difficult to find finite models of arbitrary first-order theories by exhaustive search. In this section, we compare SEM with several other similar systems. (The running times of the programs were taken on a SGI IRIX 5.3.) SEM can solve a wide range of problems quite efficiently. For example, it needs about 8 seconds to find a noncommutative group of size 18, and about 28 seconds to complete the search for QG5.13. We will explain why it is so efficient, and what problems are most suitable for different systems. See [8] for a comparison between the programs MGTP, FINDER and LDPP (another propositional theorem prover).

The program MGTP [1, 8] uses range restricted clauses to represent a problem. Such a representation is very compact, and requires little memory, but matching has to be performed in the search. In contrast, the programs FINDER, FALCON and SEM use ground clauses, and the cost of matching is avoided. FINDER generates new clauses during the search process, to remember the reasons for failures. This may require quite some memory in some cases. FALCON and SEM do not have such a mechanism, and memory is rarely a problem in most applications.

SEM can find some quite large models very easily. Besides memory management, another important reason is the use of the least number heuristic (LNH) to avoid isomorphic partial models. This makes it possible prove the unsatisfiability of large pigeon-hole problems. The heuristic was first used in FALCON [10]. However, FALCON is restricted to equational theories. And even on equational problems, it is not so efficient as SEM, because it does not use sophisticated data structures and indexing techniques. For example, FALCON-2 spends 11.7 seconds to complete the search for a Boolean algebra of size 9, while SEM needs only 0.3 second.

Programs based on the propositional logic (such as LDPP, SATO and MACE) have difficulty in handling the first order clauses where the number of variables and functions is big, because it is too expensive to convert such clauses into propositional clauses. Programs based on ground clauses and constraint satisfac-

tion appear not to suffer from these problems. For example, to find a 6-element noncommutative group, MACE allocates 2937K memory; while SEM requires only 1468K to find a noncommutative group of size 18. SEM uses rewriting to deal with nested terms. This mechanism is not included in the Davis-Putnam-Loveland (DPL) algorithm. For those problems which can be naturally represented in the propositional logic, the DPL implementations are better. For example, without using the LNH, SEM spends 4.7 seconds to solve the 8-pigeon 7-hole problem, while SATO only needs 0.4 second.

All of the aforementioned programs are based on exhaustive search. Recently, we studied the local search techniques and integrated them into SEM. These techniques are more suitable for large satisfiable problems with many solutions (e.g., the n-queens problem). In the future, we will implement and test more search techniques in our system.

References

1. Fujita, M., Slaney, J., and Bennett, F., "Automatic generation of some results in finite algebra," *Proc. IJCAI-93*, 52–57, Chambéry, France.
2. Jech, T., "OTTER experiments in a system of combinatory logic," *J. Automated Reasoning* 14 (1995) 413–426.
3. Kim, S., and Zhang, H., "ModGen: Theorem proving by model generation," *Proc. AAAI-94*, 162–167, Seattle.
4. Kunen, K., "Single axioms for groups," *J. Automated Reasoning* 9 (1992) 291–308.
5. Manthey, R., and Bry, F., "SATCHMO: A theorem prover implemented in Prolog," *Proc. CADE-9* (1988) 415–434.
6. McCune, W., "A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems," Technical Report ANL/MCS-TM-194, Argonne National Laboratory (1994).
7. Slaney, J., "FINDER: Finite domain enumerator – system description," *Proc. CADE-12* (1994) 798–801.
8. Slaney, J., Fujita, M. and Stickel, M., "Automated reasoning and exhaustive search: Quasigroup existence problems," *Computers and Mathematics with Applications* 29 (1995) 115–132.
9. Zhang, H., and Stickel, M., "Implementing the Davis-Putnam algorithm by tries," Technical Report, University of Iowa (1994).
10. Zhang, J., "Constructing finite algebras with FALCON," accepted by *J. Automated Reasoning*.
11. Zhang, J., and Zhang, H., "SEM: a System for Enumerating Models," *Proc. IJCAI-95*, 298–303.