由一阶逻辑公式得到命题逻辑可满足性问题实例*

黄 拙[1,2], 张 健[1+]

[1](中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

[2](中国科学院 研究生院,北京 100039)

Generating SAT Instances from First-Order Formulas

HUANG Zhuo[1,2], ZHANG Jian[1+]

[1](Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100039, China)

+ Corresponding author: Phn: +86-10-62644790, E-mail: zj@ios.ac.cn, http://www.iscas.ac.cn

Received 2003-09-17; Accepted 2004-01-08

Huang Z, Zhang J. Generating SAT instances from first-order formulas. *Journal of Software*, 2005,16(3): 327–335. DOI: 10.1360/jos160327

Abstract: To solve the satisfiability (SAT) problem in propositional logic, many algorithms have been proposed in recent years. However, practical problems are often more naturally described as satisfying a set of first-order formulas. When the domain of interpretation is finite and its size is a fixed positive integer, the satisfiability problem in the first-order logic can be reduced to SAT. To facilitate the use of SAT solvers, this paper presents an algorithm for generating SAT instances from first-order clauses, and describes an automatic tool performing the transformation, together with some experimental results. Several different ways of adding formulas are also discussed to eliminate symmetries, which will reduce the search space. Experiments show that the algorithm is effective and can be used to solve many problems in mathematics and real-world applications.

Key words: satisfiability problem; first-order logic; propositional logic

摘 要: 命题逻辑可满足性(SAT)问题是计算机科学中的一个重要问题.近年来许多学者在这方面进行了大量的研究,提出了不少有效的算法.但是,很多实际问题如果用一组一阶逻辑公式来描述,往往更为自然.当解释的论域是一个固定大小的有限集合时,一阶逻辑公式的可满足性问题可以等价地归约为 SAT 问题.为了利用现有的高效 SAT 工具,提出了一种从一阶逻辑公式生成 SAT 问题实例的算法,并描述了一个自动的转换工具,给出了相应的实验结果.还讨论了通过增加公式来消除同构从而减小搜索空间的一些方法.实验表明,这一算法是有效的,可以用来解决数学研究和实际应用中的许多问题.

关键词: 可满足性问题;一阶逻辑;命题逻辑

中图法分类号: TP301 **文献标识码**: A

* Supported by the National Science Fund for Distinguished Young Scholars of China under Grant No.60125207 (国家杰出青年科学基金)

HUANG Zhuo was born in 1981. He is now a graduate student at the University of Florida, USA. His current research areas are satisfiability problem, first-order logic and propositional logic. **ZHANG Jian** was born in 1969. He is a research professor at Institute of Software, the Chinese Academy of Sciences. His research areas are automated reasoning, constraint satisfaction and formal methods.

1 Introduction

Many problems from various application domains can be regarded as deciding the satisfiability of certain logical formulas. In fact, the satisfiability problem in propositional logic, known as SAT, is a fundamental problem in computer science and artificial intelligence. Many researchers have been working on SAT, and many algorithms have been proposed to solve the problem. In the 1980's, the algorithms are typically analyzed theoretically. Later on, empirical evaluation received more attention. Several efficient SAT solvers have been implemented, such as SATO[1], Walksat[2] and zchaff[3]. Their performances are evaluated against some hard SAT instances. Most of the problem instances are randomly generated, but there are also some structured problem instances. In recent years, as SAT solvers become more powerful, people pay more attention to such important applications as mathematical problem solving, hardware verification and so on.

Practical problems are often more naturally described by a set of first-order formulas. In the problem library TPTP[4], fewer than 1% of the problems are described in pure propositional logic. With only a few exceptions (such as MACE[5]), we need to prepare a set of propositional formulas for SAT solvers, either manually or using problem-specific programs. This makes the use of SAT solvers inconvenient.

On the other hand, some people have been working on satisfying first-order formulas in finite domains. This problem is known as finite model generation or finite model searching. A few efficient finite model searchers have been developed, such as FINDER[6] and SEM[7]. Although these tools are quite successful in solving some open problems in mathematics, they are very slow on some other problems (especially when many of the clauses are multi-literal), compared with current best SAT solvers.

In this paper, we describe an algorithm for generating SAT instances from first-order descriptions of practical problems. We first recall some basic concepts in the next section. Then in Section 3, we elaborate on our approach for transforming first-order formulas into propositional ones. In Section 4, we discuss how to add extra formulas which can eliminate some symmetries in the problem, so as to reduce the search time of a SAT solver. Then we give some experimental results in Section 5. Finally, we compare our approach with other similar methods and mention some possible improvements in the future.

2 Preliminaries

In this section, we introduce some concepts and notations in mathematical logic, which will be used later.

A propositional formula is constructed recursively from propositional variables and logical connectives like AND ('&'), OR ('|'), NOT ('~'). A literal is a variable or its negation (e.g. p, $\sim q$), and a clause is a disjunction of literals. A set of clauses is a conjunction of the clauses. A SAT solver accepts a set of clauses as input, and tries to find a Boolean value (TRUE or FALSE) for each variable, such that every input clause is true.

In the first-order logic, we have function and predicate symbols, in addition to variables and logical connectives. Function symbols without arguments are called constants, like a, b, c. From functions and variables, we can construct terms. For instance, *wifeOf*(x) is a term. Here *wifeOf* is a function symbol. If a term has a function symbol which is not a constant, the term is called a *complex term*.

A special predicate symbol is the equality predicate (EQ or '='). Its negation is denoted by NEQ or '!='. A first-order literal is a predicate applied to a term, or its negation. For instance, *isMale*(*husbandOf*(x)) is a literal. Here *isMale* is a predicate symbol and *husbandOf* is a function symbol. Again, a clause is a disjunction of literals. An example of clauses is: *isMale*(x) | *gender*(x)=*Female*. Note that every variable in a clause is assumed to be universally quantified. So the previous clause means, for any x, either x is male, or the gender of x is female. A clause is called a *ground clause* if it does not contain variables.

A set of first-order clauses is satisfiable if there is an interpretation of the function and predicate symbols in a non-empty domain such that every clause becomes true. There is no algorithm which can decide the satisfiability of an arbitrary first-order formula. Thus we focus on interpretations whose domains are finite, and the resulting problem is called finite model generation or finite model searching. Under this restriction, it is at least possible to decide the satisfiability by enumerating every possible interpretation.

Without loss of generality, we can assume that the non-empty domain is denoted by a set of integers, e.g. {0, 1, 2}. In many cases, a finite interpretation may be represented by some tables. Each table corresponds to the interpretation of a function (predicate) symbol. For instance, the following table satisfies the clause $f(x,y)=f(y,x)$.

F	0	1	2
0	2	1	0
1	1	2	1
2	0	1	2

Obviously, the table is symmetric with respect to the main diagonal, so for any x and y, $f(x,y)=f(y,x)$ is true. A finite model searcher accepts a set of first-order clauses as input and finds a model such as the above one if it exists. Thus the goal of a model searcher is to decide the values of all the entries in the table(s). An entry is also called a *cell*. Syntactically a cell is given by a term whose arguments are elements of the domain. For instance, the cells in the second row of the above table can be represented as $f(0,0)$, $f(0,1)$, $f(0,2)$.

3 Transforming First-Order Clauses to Propositional Clauses

At the beginning, we have first-order clauses with function symbols and variables. Our goal is to translate them into some propositional clauses such that, if the first-order clauses can be satisfied, the propositional clauses can also be satisfied, and vice versa. This is done in several stages, as explained below.

3.1 Eliminating the variables

We first collect the set of variables occurring in each clause and replace the variables with all the values they can take. Then we get a set of ground clauses. For example, suppose in the input, we have a clause $f(x,y)=x$ and the domain is {0,1}. Then we instantiate it into the following four ground clauses: $f(0,0)=0$, $f(0,1)=0$, $f(1,0)=1$, $f(1,1)=1$.

3.2 Translating ground clauses into semi-propositional clauses

Now we have a set of ground clauses which contain only function symbols, predicate symbols and domain elements (0,1,2,…). In this subsection, our goal is to translate these ground clauses into an intermediate form called semi-propositional clauses. In such a clause, every literal takes one of the following forms: *cell* = *d* or *cell* != *d*. In subsection 3.3, we will obtain propositional clauses from the semi-propositional clauses.

To get semi-propositional clauses, we record all the things people check when they test a ground clause. Basically we examine all the terms in the post-order and see whether the top predicate can be satisfied.

Since common sub-expressions often occur in clauses and this will reduce the efficiency of the algorithm, we define a data structure called Non-common Terms (NT) to get rid of the redundancy. Each NT comes from a term and each term can correspond to an NT. All the NTs form an NT list. The information in NTs is like the following.

Term type	Examples	NT
Fixed value	0, 1, false	(id, the value)
Cell	$f(0,1)$, $g(1,0,1)$	(id, cell-id, range of the cell, chosen value)
Complex term	$f(h(1),1)$, $h(h(2))$	(id, func-id, arity, pointer array which points to all the args, range, chosen value)

For the sake of conciseness, we remove the first kind of NTs from the NT list and add their information into the

NT that corresponds to the complex term they appear. A predicate can be regarded as a special kind of functions whose range is {true, false}.

Here is an example.

Suppose we are given the clause: $f(0,g(1,h(0)))=f(g(1,h(0)),2)$. It can be depicted as a tree.

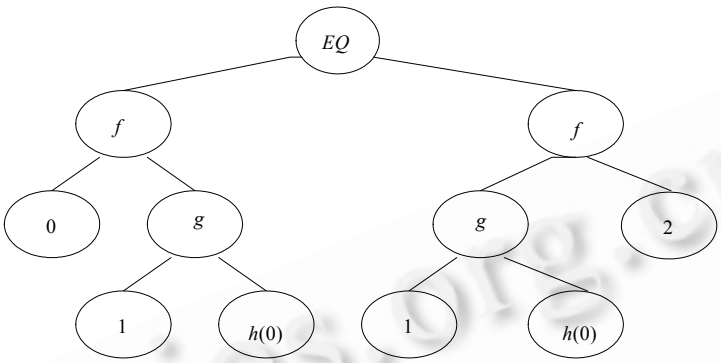

Fig.1 Tree structure of a clause

The term $h(0)$ is a cell and we can see that $g(1,h(0))$ is a common subexpression. In our algorithm we use postorder traversal to process this tree. The resulting NT list is as follows.

Id	Corresponding term	Content of NT
1	$h(0)$ (Cell)	The id of $h(0)$,…
2	$g(1,h(0))$	The id of g, 2, (fixed value 1, PNT(1)*),…
3	$f(0,g(1,h(0)))$	The id of f, 2, (fixed value 0, PNT(2)*),…
4	$f(g(1,h(0)),2)$	The id of f, 2, (PNT(2)*, fixed value 2),…
5	$f(0,g(1,h(0)))=f(g(1,h(0)),2)$	The id of EQ, (PNT(3)*, PNT(4)*),…

We omit the range and the chosen value in the table.

(*) PNT(i) is a function to get the i-th NT in the NT list.

Since we use postorder traversal in the algorithm, we can choose values for NT_1, NT_2,…, consecutively. When a complex term NT has a chosen value, all its arguments should have had the chosen value so we can map the function NT to one and only one Cell.

After getting the NT list for every clause, we translate the clause into semi-propositional clauses. For a better understanding, we present our algorithm starting with the simplest clauses.

3.2.1 Translating a clause having only one EQ literal

We call the whole term appearing at the left of the EQ as the left term, and the whole term appearing at the right of the EQ as the right term. In the former example, we call $f(0,g(1,h(0)))$ the left term and $f(g(1,h(0)),2)$ the right term. We always add a new NT into the NT list for the left and right term unless the term has a fixed value.

Let CL denote the clause we process now. We assume NT_l is determined by the left term T_l, and NT_r is determined by the right term T_r. We remove the last NT in the NT list that corresponds to the predicate EQ and assume that there are m NTs left.

We consider the following two cases:

Case 1: Neither T_l nor T_r is a domain element.

Now we have m NTs and each of them can choose a value from its range. We restrict the choice such that NT_l and NT_r have the same value. From the discussion above, we can get the corresponding cells $C_1,C_2,\ldots,C_m$ one by one. For any k ($1\le k\le m$), let V_k be the value that the k-th NT takes. For every $V=(V_1,V_2,\ldots,V_m)$ such that $V_l=V_r$, we

generate two new clauses:

$$(C_1=V_1\&\ldots\&C_{l-1}=V_{l-1}\&C_l=V_l\&C_{l+1}=V_{l+1}\&\ldots\&C_{r-1}=V_{r-1}\&C_{r+1}=V_{r+1}\&\ldots\&C_m=V_m)\text{->}C_r=V_r \quad (1.1)$$

$$(C_1=V_1\&\ldots\&C_{l-1}=V_{l-1}\&C_{l+1}=V_{l+1}\&\ldots\&C_{r-1}=V_{r-1}\&C_r=V_r\&C_{r+1}=V_{r+1}\&\ldots\&C_m=V_m)\text{->}C_l=V_l \quad (1.2)$$

We can rewrite them into CNF form:

$$C_1!=V_1\,|\ldots|\,C_{l-1}!=V_{l-1}\,|\,C_l!=V_l\,|\,C_{l+1}!=V_{l+1}\,|\ldots|\,C_{r-1}!=V_{r-1}\,|\,C_{r+1}!=V_{r+1}\,|\ldots|\,C_m!=V_m\,|\,C_r=V_r \quad (1.1')$$

$$C_1!=V_1\,|\ldots|\,C_{l-1}!=V_{l-1}\,|\,C_{l+1}!=V_{l+1}\,|\ldots|\,C_{r-1}!=V_{r-1}\,|\,C_r!=V_r\,|\,C_{r+1}!=V_{r+1}\,|\ldots|\,C_m!=V_m\,|\,C_l=V_l \quad (1.2')$$

The set of all clauses like (1.1′) and (1.2′) is denoted by $CL1$.

If an interpretation mapping the cell C_k to V_k ($1\le k\le m$) can satisfy the clause CL, it will be a solution of (1.1′) and (1.2′), and if it can't satisfy CL, it won't be a solution. Other interpretations are always the solutions of (1.1′) and (1.2′). Since we generate similar clauses for all the possible value combinations, we will just get the set of all the suitable interpretations after solving the clause sets.

Case 2: One of T_l and T_r is a domain element.

If T_l is a domain element, we choose those vectors with V_r equal to that domain element, and let $CL1$ just include those clauses like (1.1′). If T_r is a domain element, we choose those vectors with V_1 equal to that domain element, and let $CL1$ just include those clauses like (1.2′).

3.2.2 Translating a clause that has only one NEQ literal

Now we consider a clause that has just one literal and the predicate of the literal is NEQ. To translate this clause, we repeat the work we did before. While none of T_l and T_r is a domain element, we still choose the vector $V=(V_1,V_2,\ldots,V_m)$ such that $V_l=V_r$, but we just generate one clause:

$$(C_1=V_1\&\ldots\&C_{l-1}=V_{l-1}\&C_l=V_l\&C_{l+1}=V_{l+1}\&\ldots\&C_{r-1}=V_{r-1}\&C_{r+1}=V_{r+1}\&\ldots\&C_m=V_m)\text{->}C_r!=V_r \quad (2.1)$$

It can be rewritten into the following clausal form:

$$C_1!=V_1\,|\ldots|\,C_{l-1}!=V_{l-1}\,|\,C_l!=V_l\,|\,C_{l+1}!=V_{l+1}\,|\ldots|\,C_{r-1}!=V_{r-1}\,|\,C_{r+1}!=V_{r+1}\,|\ldots|\,C_m!=V_m\,|\,C_r!=V_r \quad (2.1')$$

When one of T_l and T_r is a domain element, we do similar work as Case 2 in 3.2.1, except changing the last literal of the clause from $C_r=V_r$($C_l=V_l$) to $C_r!=V_r$($C_l!=V_l$).

3.2.3 Translating other clause that has only one literal

We assume the predicate of the literal is $P(\cdots)$. We repeat the work as Case 1 in subsection 3.2.1. Similarly, we choose $V=(V_1,V_2,\ldots,V_m)$, but now we needn't add any restriction on them. Let CP denote the corresponding cell of the $P(\cdots)$. So we can generate a clause like the following one:

$$(C_1=V_1\&\ldots\&C_m=V_m)\text{->CP=true (or CP=false if the literal is negative)} \quad (3.1)$$

It can be changed into the following clausal form:

$$C_1!=V_1\,|\ldots|\,C_m!=V_m|\text{CP=true(false)} \quad (3.1')$$

3.2.4 Translating a clause with more than one literals

We assume the clause CL is $L_1\,|\,L_2\,|\ldots|\,L_h$, where $L_1,L_2,\ldots,L_h$ are the literals of CL. We can get the corresponding clause sets of $L_1,L_2,\ldots,L_h$ by the method described above. Let us denote these clause sets by $CL1_1,CL1_2,\ldots,CL1_h$. So we can get the corresponding clause sets of CL as $(CL1_1\,|\,CL1_2\,|\ldots|\,CL1_h)$. We can change $CL1_1\,|\,CL1_2\,|\ldots|\,CL1_h$ into the CNF form.

3.3 Translating the semi-propositional clauses into propositional clauses

In addition to the above clauses, we should add some other clauses to keep the consistency. For every cell C, if its range of values is $\{0,\ldots,n-1\}$, we should add clauses like:

$$C=0\,|\,C=1\,|\ldots|\,C=n-1 \quad (4.1)$$

$$C!=a\,|\,C!=b\ (0\le a<b\le n-1) \quad (4.2)$$

The set of clauses like (4.1) and (4.2) is denoted by $CL2$.

Now we get a set of semi-propositional clauses. Then we change these clauses into propositional clauses by translating each equality like $C_j=V_j$ into a Boolean variable. Hence we get an SAT problem instance.

4 Symmetry Constraints

One advantage of the first-order reasoning is that we can employ more structural information in the problem formulation. In contrast to random problems, structured problems usually have a lot of symmetries. In other words, there is much isomorphism in the search space. Taking them into consideration may reduce the search space. One way is to use special extra constraints for a particular class of problems. For example, adding a certain form of inequalities greatly reduces the search time for the quasigroup problems[8]. This method is effective sometimes, but one needs insight to find the constraints.

A more general approach is to use the so-called Least Number Heuristic (LNH)[7]. We shall not give details about it here. But roughly speaking, it reduces the number of possible values for certain cells and thus prunes the search tree. For simplicity, in this section, we assume that the input has only one function f which is binary. Then using the LNH is essentially the same as adding the following formula (S1):

$(f(0,0)=0\&((f(0,1)=0\&(f(1,0)=\ldots))$
$\quad|\,(f(0,1)=1\&(f(1,0)=\ldots))$
$\quad|\,(f(0,1)=2\&(f(1,0)=\ldots))))$
$(f(0,0)=1\&((f(0,1)=0\&(f(1,0)=\ldots))$
$\quad|\,(f(0,1)=1\&(f(1,0)=\ldots))$
$\quad|\,(f(0,1)=2\&(f(1,0)=\ldots))))$

In general, the LNH is only effective at the first few levels of the search tree. But it can greatly reduce the search space when the size of the domain is large.

How can we take advantage of the symmetries when transforming first-order clauses into propositional clauses? In the following, we discuss several methods.

(1) As demonstrated in Ref.[9], we may slightly modify the finite model searcher SEM and ask it to generate a set of partial solutions while using the LNH. From each partial solution, we can generate a set of propositional clauses. As a result, from each set of first-order clauses, we get several sets of propositional clauses. If any of the latter sets is satisfiable, the original problem has a solution.

(2) MACE[5] has a command-line option '-c', which says that constants in the input should be assigned unique elements of the domain. It can eliminate much isomorphism in many cases. But it is not safe, in the sense that there may be a solution in which two constants are assigned the same value. This kind of solution will not be found when the option '-c' is used.

(3) Certainly we can translate the formula (S1) into a set of propositional clauses and add them as extra constraints. However, there are too many such clauses when the domain size is not too small.

(4) An approximation of LNH is to add the following clauses (S2):

$f(0,0)=0\,|\,f(0,0)=1.$
$f(0,1)=0\,|\,f(0,1)=1\,|\,f(0,1)=2.$
$f(1,0)=0\,|\,f(1,0)=1\,|\,f(1,0)=2\,|\,f(1,0)=3.$
……

This is only an approximation, because some combinations actually need not to be considered. For example, when $f(0,0)=0\&f(0,1)=1$, we should not consider the case $f(1,0)=3$ according to the LNH. Although it is an approximation of the LNH, it still prunes the search tree greatly, since we now need to examine only 2 (instead of n) possible values for $f(0,0)$, only 3 (instead of n) possible values for $f(0,1)$,… More accurate approximations are

possible. For example, we can add the constraint: $f(1,0)=3->f(0,1)=2$. The clauses (S2) also imply that $f(0,0)!=2$, $f(0,0)!=3,\ldots,f(0,0)!=n-1$. So some clauses in CL1 and CL2 can be eliminated. We can replace Eq.(4.1) by S2. In this way, we get SAT instances which are easier but not too large.

5 Experimental Results

Based on the above ideas, we have implemented an automatic tool for generating SAT instances, called SAGE. To deal with the symmetry problem, we choose to adopt the last method for adding extra clauses, for its simplicity and generality.

We have tested out tool on a number of well-known problems:

(1) Logic and abstract algebra: This class includes the problems "Cl_bn1", "Group" and "Ortho". Some of them are described in Ref.[10]. The problem "Ortho" is a previously open problem, solved by McCune in Ref.[11]. The domain size is 8.

(2) Combinatorics: Quasigroup existence problems described in Ref.[8]. Qg*i.j* means the *i*'th quasigroup problem with domain size *j*. Also included are some Latin square problems.

(3) Puzzles: "jobs" and "salt", which are well-known in AI and logic programming. Gra-16 is an interesting graph problem designed by the author.

The following table summarizes the results. All running times are given in seconds. We use SATO[1] Version 3.2.1 as our SAT solver, on a SUN Ultra SPARCstation 60.

Table 1 Comparison with other tools

Problem name	Number of generated clauses	SATO's time	Time after adding LNH	SEM's time	MACE's time
Group	22794	0.12	0.12	<0.01	0.21
Ortho	1582158	1199.19	57.76	1.66	2.52 (1.71)***
Jobs	1525	0.01	–	<0.01	–**
Salt	150	<0.01	–	<0.01	–**
QG1.7	68411	0.34	0.43	6.93	0.48
QG2.7	83531	1.98	0.46	8.44	0.56
QG5.10	42320	1475.09	25.92	0.02*	1533.83 (0.20)***
Gra-16	12544	0.22	1.02	>2 hours	–**
Latin 6	59514	3.21	–	62.19	497.0
Latin 7	158607	10771.49	–	>24 hours	>24hours

Notes: (*) If we don't use LNH for the qg5, the running time of SEM will be more than an hour.

(**) Since MACE can't solve problems that have more than one sort, we didn't use it to solve these problems.

(***) The time within the parentheses is the execution time of MACE when special constraints are added to eliminate isomorphism.

The fourth column of the table shows the time of SATO when we add the first three clauses of (S2), together with the clause $f(1,0)=3->f(0,1)=2$.

We can see that, when combined with SATO, our tool can solve most problems quite efficiently. It is at least as good as MACE and SEM on most problems. For some problems, the performance of SATO+SAGE is much better. The table also shows that employing symmetries is quite useful on hard problems (especially those unsatisfiable problems such as Ortho and QG5.10). For the problem instance qg2.7, using approximate LNH reduces 3.50% variables, 12.97% clauses and 76.77% search time. But for some easy problems, it seems not very useful.

The translation time is omitted in the table. For many problems, the translation time is much less than the solving time. The search times can be reduced if we choose more efficient SAT solvers and if we add more constraints for eliminating isomorphism.

6 Related Work

There are other tools which are similar to ours. For example, Kim and Zhang[12] described a tool called ModGen, and McCune designed another tool called MACE[5]. Both ModGen and MACE can solve finite domain search problems stated in first-order logic. Each tool generates an equivalent SAT instance, solves it, and translates the SAT solution back into the first-order form.

Although the main design of these three tools is similar, there are still some differences between our tool and the others. Firstly, ModGen only uses SATO and MACE only uses ANL-DP to solve the generated SAT instance. However, there are many other efficient SAT solvers and it is still unknown which one is the most suitable for a certain problem. Our tool SAGE allows the user to choose an arbitrary SAT solver and offers an interface to translate the SAT solution into an understandable one. Secondly, we use some extra constraints to reduce the search space, and these constraints are independent of any particular problem. Another minor difference is that we first instantiate the first-order clauses (rather than flatten them into relational form). This gives us more chances of finding common sub-expressions. For example, suppose that in the input, there is a clause $h(g(f(y,z)),g(f(y,x)))=y$. After instantiating this clause, we get n^3 ground clauses, and n^2 of them have common subexpressions $g(f(y,z))$ or equivalently $g(f(y,x))$. For this example, MACE generates n^7 clauses, and SAGE generates $(n^7-n^6+n^4)$ clauses.

7 Conclusions

To use a SAT solver, one usually has to prepare a set of propositional clauses. In this paper, we have described a method and an automatic tool which generates such a clause set from an abstract problem description in first-order logic. Its input is a set of first-order clauses together with the domain size(s), and its output is a set of propositional clauses which can be accepted by most SAT solvers.

Most of the examples used in this paper come from mathematics. There are several reasons. Firstly, these problems are well known and difficult. Some of them are previously open problems. Secondly, it is easy to state the problems and communicate them with other researchers. Moreover, assisting mathematicians is an important goal of AI. In fact, Ref.[8] received the outstanding paper award in IJCAI-93, and McCune's success in solving a long-standing open problem is considered as a major achievement in AI. However, it should be noted that our tool is based on the general formalism of logic, and logical formulas can express problems from a wide variety of domains (e.g., planning, hardware verification). We believe that it will be quite useful. In the future, we shall try to explore more symmetries to reduce the search space, and try to generate fewer clauses.

References:

[1] Zhang H, SATO: An efficient propositional prover. In: McCune W, ed. Proc. of the 14th Int'l Conf. on Automated Deduction (CADE-14). LNAI 1249, 1997. 272−275.

[2] Selman B, Kautz H, Cohen B. Noise strategies for improving local search. In: Proc. of the 12th National Conf. on Artificial Intelligence (AAAI'94). 1994. 337−343.

[3] Moskewicz M, *et al*. Chaff: Engineering an efficient SAT solver. In: Proc. of the 38th Design Automation Conf. Las Vegas, 2001. 530−535.

[4] Shutter C, Sutcliffe G. The TPTP problem library (TPTP v2.2.0). Technical Report 99/02, James Cook University, Townsville, 1999.

[5] McCune W, MACE 2.0 reference manual and guide. Technical Memo ANL/MCS-TM-249, Argonne National Laboratory, 2001.

[6] Slaney J, FINDER: Finite domain enumerator, system description. In: Bundy A, ed. Proc. of the 12th Int'l Conf. on Automated Deduction (CADE-12). LNCS 814, 1994. 798−801.

[7] Zhang J, Zhang H. SEM: A system for enumerating models. In: Mellish CS, ed. Proc. of the 14th Int'l Joint Conf. on Artificial Intelligence (IJCAI). 1995. 298−303.

[8] Fujita M, Slaney J, Bennett F. Automatic generation of some results in finite algebra. In: Bajcsy R, ed. Proc. of the 13th Int'l Joint Conf. on Artificial Intelligence (IJCAI). 1993. 52−57.

[9] Zhang J. Automatic symmetry breaking method combined with SAT. In: Proc. of the 16th ACM Symp. on Applied Computing. Las Vegas, 2001. 17−21.

[10] Zhang J. Problems on the generation of finite models. In: Bundy A, ed. Proc. of the 12th Int'l Conf. on Automated Deduction (CADE-12). LNAI 814, 1994. 753−757.

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

[12] Kim S, Zhang H. ModGen: Theorem proving by model generation. In: Proc. of the 12th National Conf. on Artificial Intelligence (AAAI'94). 1994. 162−167.

****

"数字媒体处理及数字博物馆技术"高级研讨班会议通知

由国家教育部资助的"数字媒体处理及数字博物馆技术"高级研讨班将于 2005 年 4 月 18-20 日在浙江大学举行。组织者邀请国内外著名的专家作有关多媒体/数字博物馆/虚拟现实研究及应用进展的报告。欢迎各位代表参加。特邀报告的专家有:

Takeo OJIKA 教授, 日本 GIFU 大学, 国际 VSMM 学会名誉主席

Malcolm Padmore 教授, 英国 SALFORD 大学虚拟环境研究中

Yifa Jiang 博士, 日本 Riken 生物信息处理研究中心

S. Karen 教授, 英国 SALFORD 大学虚拟环境研究中心

潘金贵教授, 南京大学, 多媒体研究所副所长

石教英教授, 浙江大学, 中国图像图形学会虚拟现实专委会主任

潘志庚研究员, 浙江大学, 中国图像图形学会虚拟现实专委会副主任兼秘书长

孙守迁教授, 浙江大学, 现代工业设计研究所所长

研讨班联系人:张明敏 浙江大学计算机学院, 杭州 310027 电话:0571-87990451

第 2 届"游戏与虚拟现实在教育中的应用"国际研讨会会议通知

2006 年 4 月 16−18 日

杭州,浙江大学

主办单位: 浙江大学, 中国图像图形学会虚拟现实专业委员会

大会名誉主席:潘云鹤院士(中国,浙江大学), Judith Brown (美国,ACM SIGGRAPH)

会议主席:石教英教授(中国,浙江大学), Jose Encarnacao (德国,INI-Graphics)

程序委员会主席:潘志庚研究员(中国,浙江大学), Martin Goebel(德国,IMK)

会议联系人:金小刚博士

地址:310027 杭州市浙江大学 CAD&CG 国家重点实验室

Website: http://www.cad.zju.edu.cn/games

E-mail: games@cad.zju.edu.cn