



**G. Felici, A. Remshagen, K. Truemper**

**THE FUTILE QUESTIONING PROBLEM**

**R. 591 Luglio 2003**

**Giovanni Felici** – Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30 -  
00185 Roma, Italy. Email : felici@iasi.cnr.it.

**Anja Remshagen** – State University of West Georgia, Carrollton GA 30118, USA. Email :  
anja@westga.edu.

**Klaus Truemper** – University of Texas at Dallas, Box 830688, Richardson, TX 75083-0688,  
USA, Email: klaus@utdallas.edu.

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica, CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161

fax ++39-06-7716461

email: [iasi@iasi.rm.cnr.it](mailto:iasi@iasi.rm.cnr.it)

URL: <http://www.iasi.rm.cnr.it>

## Abstract

In the futile questioning problem, one must decide whether acquisition of additional information can possibly lead to proof of a conclusion. In the case considered here, solution of that problem demands evaluation of a quantified Boolean formula at the second level of the polynomial hierarchy. We call that evaluation problem Q-ALL SAT. In this paper we determine the complexity of Q-ALL SAT and of special subclasses, and develop a solution algorithm that utilizes decomposition, enumeration, and learning of clauses. We report first results for two sets of instances involving a robot route problem and a game problem. According to these preliminary results, the algorithm is quite effective. Finally, we summarize several variations of Q-ALL SAT that are of practical interest.



## 1. Introduction

Many intelligent systems interactively acquire information from the user and make decisions based on those data. For example, a diagnostic system may query the user about symptoms to establish a diagnostic goal that the system has selected by a preliminary analysis. If the goal happens to be unprovable regardless of the answers that may possibly be supplied by the user, then that goal is futile and should be abandoned. If the system can decide whether that situation is at hand, then fruitless lines of questioning can be cut short. We call the problem of making that decision the *futile questioning problem*. To solve that problem, one must evaluate a quantified Boolean formula at the second level of the polynomial hierarchy. Thus it is unlikely that one can develop a polynomial solution algorithm. That conclusion does not rule out the existence of reasonably effective or even fast algorithms for special subclasses of the problem. Such algorithms are very much needed since the problem—also known as a form of logic-based abduction—is of great practical importance. Indeed, the problem needs to be solved in many applications. Examples are conditional planning (Rintanen [1999]), regulatory compliance (Straach and Truemper [1999]), and diagnosis (Poole [1989]). Further applications are discussed by Eiter, Gottlob, and Leone [1997] and Truemper [2003].

Recent work for solving quantified Boolean formulas at the second or even higher levels is as follows. Kleine Büning, Karpinski, and Flögel [1995] propose a complete and sound resolution-based algorithm. However, due to the possibly exponential growth of the formula, these algorithms are not practical. Among the most promising algorithms for quantified Boolean formulas (QBF) are backtracking search algorithms, which have been proved to be very successful in solving the satisfiability problem SAT (Bayardo and Schrag [1997], João, Marques-Silva, and Sakallah [1999], Zhang [1997], Zhang, Madigan, Moskewicz, and Malik [2001]). Cadoli, Schaerf, Giovanardi, and Giovanardi [2003] explore different heuristics with their solver Evaluate to choose the branching literal and investigate properties of randomly generated quantified Boolean formulas. Giunchiglia, Narrizano, and Tacchella [2002], [2003] implement a learning scheme in their solver QuBE. Rintanen [2001] extends the Davis-Putnam procedure by unit propagation for quantified formulas. The QBF solver Quaffle by Zhang and Malik [2002] learns conflict clauses during the solution process. A satisfiability-driven learning scheme by Zhang and Malik [2002] augments the CNF formula by a formula in disjunctive normal form. Plaisted, Biere, and Zhu [2003] integrate a technique based on binary decision diagrams in their QBF solver QSAT.

The above mentioned algorithms treat quantified Boolean formulas where the quantifiers constitute a prefix of a formula in conjunctive normal form (CNF). In contrast, the futile questioning problem is composed of two CNF formulas. The problem demands to find an assignment for a subset of the variables such that the first formula is satisfiable while the second formula is unsatisfiable. We call that problem Q-ALL SAT. In principle, the QBF solvers for formulas where all quantifiers are a prefix of the formula, can be used to solve Q-ALL SAT instances. However, the required transformation tends to destroy structural properties that can be exploited in a direct attack on Q-ALL SAT.

In this paper, we identify several easier subclasses of Q-ALL SAT. We also present an algorithm for Q-ALL SAT that in preliminary tests, reported below, has proved to be quite effective. The implementation relies on a compiler that decomposes a formula into an easy and a hard part. The compiler approach is particularly useful for certain applications, for example, for medical diagnosis, where a decisions must be made for various formulas each of which is derived from just one formula by fixing of some variables.

## 2. Complexity Results

An instance of Q-ALL SAT is a quantified propositional formula of the form

$$\exists Q(\exists X R \wedge \forall Y \neg S) \tag{1}$$

where  $Q$ ,  $X$ , and  $Y$  are disjoint variables sets,  $R$  is a propositional formula with variables  $Q$  and  $X$ , and  $S$  is a propositional formula with variables  $Q$  and  $Y$ . We consider here only formulas in *conjunctive normal form* (CNF), where *clauses* are joined by conjunction, and where each clause is a disjunction of *literals*. A literal is a variable or a negated variable. In the first (resp. second) case, the literal is called *positive* (resp. *negative*). The problem Q-ALL SAT demands to decide whether formula (1) evaluates to *True*. If the formula evaluates to *True*, a truth assignment to the  $Q$  variables must be supplied so that, for that assignment, the CNF formula  $R$  is satisfiable and  $S$  is unsatisfiable. The assignment constitutes a *solution* of formula (1). We define an instance of QBF as a quantified CNF formula where all quantifiers constitute a prefix of the CNF formula.

The problem Q-ALL SAT is  $\Sigma_2^P$ -complete. Thus, it is at the second level of the polynomial hierarchy, and existence of a polynomial-time solution algorithm is unlikely. For the polynomial hierarchy, see Stockmeyer [1976]. However, for several special cases, Q-ALL SAT is in P, or its complexity is at least reduced by one level in the polynomial hierarchy. We list results for several such easier subclasses. For details and proofs, see Remshagen and Truemper [2003].

A CNF formula is in *Horn form* if each clause contains at most one positive literal. Dowling and Gallier [1984] introduce a linear-time algorithm to solve the satisfiability problem for Horn formulas. Eiter and Gottlob [1995] show that a form of logic-based abduction that is closely related to Q-ALL SAT and that involves Horn formulas, is NP-complete. One can show that Q-ALL SAT is NP-complete for Horn formulas as well. Aspvall, Plass, and Tarjan [1979] describe a polynomial-time algorithm for QBF with 2CNF formulas, where each clause contains at most two literals. Using a similar approach, one can prove that Q-ALL SAT can be solved in polynomial time in case of 2CNF.

Consider instances of Q-ALL SAT where  $R$  is a Horn formula and  $S$  contains only positive literals from the variable set  $Q$ . The literals of  $Y$  are not constrained. We call the resulting subclass *antimonotone* Q-ALL SAT. Due to the properties of Horn formulas (see Dowling and Gallier [1984]), the complexity of an instance of antimonotone Q-ALL SAT is reduced to the complexity of the satisfiability problem  $S$ .

A CNF formula is *balanced* if the matrix representation of the formula does not have certain circuit submatrices, see Truemper [1998]. In that case, Q-ALL SAT can be solved in polynomial time.

The last subclass of Q-ALL SAT covered here is called *2cut* Q-ALL SAT. As before, let  $S$  be the CNF formula with variable set  $Q$  and  $Y$ . Assume the variable set  $Y$  is partitioned into  $Y_1$  and  $Y_2$ . All variables in  $Q$  occur nonnegated in  $S$ , and all variables in  $Y_1$  occur negated in  $S$ , except for possibly one clause whose literals are the nonnegated variables of  $Y_1$ . All clauses that contain at least one  $Q$  variable, do not contain  $Y_2$  variables. For each variable  $y$  in  $Y_1$  the following property holds. Consider all clauses that contain  $y$  and  $Q$  variables only. Let  $Q'$  be the set of  $Q$  variables occurring in these clauses. In each of these clauses, at most one of the variables in  $Q'$  does not occur. Due to the properties of 2cut Q-ALL SAT, it is possible to learn short clauses: any assignment to the  $Q$  variables that leads to satisfiability in  $S$  induces a clause that needs to be satisfied by all solutions of the Q-ALL SAT instance. The clause contains at most two literals which arise from  $Q$  variables. The problem 2cut Q-ALL SAT is NP-complete.

Each of the cases Horn form, 2CNF, antimonotone Q-ALL SAT, balancedness, and 2cut Q-ALL SAT can be identified in polynomial time.

### 3. The Solution Algorithm

The solution algorithm has to find a truth assignment for the variables of  $Q$  such that  $R$  is satisfiable and  $S$  is unsatisfiable. In recent years, there has been significant progress in the development of SAT solvers. Among the most successful complete procedures are backtracking search algorithms that integrate dynamic learning of clauses, see for example Bayardo and Schrag [1997], GRASP by João, Marques-Silva, and Sakallah [1999], SATO by Zhang [1997], or zChaff by Zhang, Madigan, Moskewicz, and Malik [2001]. We also use a backtracking scheme for Q-ALL SAT; but instead of treating  $R$  and  $S$  as individual CNF formulas, we consider the CNF formula  $R \wedge S$  of all clauses of  $R$  and  $S$ . Recall that  $R$  and  $S$  have the common variable set  $Q$ . The backtracking algorithm selects only the variables  $Q$ . There are two cases that cannot lead to a solution of a Q-ALL SAT instance and thus trigger backtracking in the search tree:

1. A truth assignment for some  $Q$  variables produces an empty clause in  $R$ .
2. A truth assignment for some  $Q$  variables leads to satisfiability in  $S$ , no matter to which truth values the remaining  $Q$  variables are set.

In the first case, the reason for the empty clause can be determined from the search tree and can be added as an *R-conflict clause* to  $R$ . Any learning scheme used in SAT solvers, for example the method in Bayardo and Schrag [1997], João, Marques-Silva, and Sakallah [1999], Zhang [1997], Zhang, Madigan, Moskewicz, or Malik [2001], can be applied in that step.

The second case occurs if the current assignment for some variables in  $Q$  and possibly for some additional variables in  $Y$  satisfies all clauses of  $S$ . We can learn a clause in that case as follows. The truth assignment of the fixed  $Q$  variables constitutes a clause that contains each fixed  $Q$  variable and that cuts off that assignment in the search tree. Hence, the clause can be added to  $R \wedge S$ . Let us call that clause the *S-conflict clause*. Observe that this clause is useless for the tree search since it is always satisfied after the succeeding backtracking step. Instead of adding the entire *S-conflict clause*, we first sharpen that clause by the following heuristic. First, all free  $Q$  variables are removed from the clauses of  $S$ . Then one by one, the fixed  $Q$  variables are deleted in the clauses of  $S$ , and a SAT procedure determines if  $S$  is still satisfiable. If deletion of a variable results in unsatisfiability of  $S$ , the variable is added again to the clauses of  $S$ . Otherwise, the corresponding literal is removed from the *S-conflict clause*. If the process eliminates at least one variable, the sharpened *S-conflict clause* is added to  $R$ . The heuristic deletes variables first that have been fixed last in the search tree. Thus, the variables fixed most recently are more likely removed than the variables fixed in the beginning of the search process, and the new clauses will more likely lead to nonlinear backjumping, that is, the algorithm can backtrack more than one node in the search tree. The heuristic to sharpen *S-conflict clauses* involves solving several satisfiability problems in  $S$ . This can be a serious time issue if  $S$  is very hard. However in real-world problems, the satisfiability problem  $S$  is often easy. In addition, all  $Q$  variables are either fixed or deleted when the heuristic solves  $S$ . So the resulting CNF formula is reduced substantially.

Fig. 1 supplies a summary of the algorithm.

In Step 1, the next variable is selected by a version of the MOMS (Maximum Occurrences in Minimum Size clauses) heuristic adapted to Q-ALL SAT as follows. For each  $Q$  variable  $q$ , we

---

*Input:*  $R$  with variable set  $Q$  and  $X$ .  $S$  with variable set  $Q$  and  $Y$ .  
 $X$  and  $Y$  are disjoint.

A partition of the variable set  $Q \cup X \cup Y$  into  $V_E$  and  $V_N$   
such that any subsystem of  $R \wedge S$  involving just the variables  
of  $V_E$  can be solved by a given linear-time algorithm.

*Output:* A solution, or “there is no solution.”

1. Select the next  $Q$  variable and a truth value. If all variables have been tried, output “there is no solution,” and stop.
  2. Do unit-resolution on the variables of  $X$  and  $Y$ .
  3. If all variables  $V_N$  are fixed, solve  $R \wedge S$  by the given linear-time algorithm.
  4. If the fixed  $Q$  variables satisfy all clauses of  $S$ , sharpen the  $S$ -conflict clause, and add it to  $R$ . Backtrack, and go to Step 1.
  5. If the fixed  $Q$  variables produce an empty clause in  $R$  and thus make  $R$  unsatisfiable, find an  $R$ -conflict clause, and add it to  $R$ . Backtrack, and go to Step 1.
  6. If the fixed  $Q$  variables produce an empty clause in  $S$  or if all  $Q$  variables are fixed:
    - (a) Solve  $R$ .
    - (b) If  $R$  is unsatisfiable, find an  $R$ -conflict clause, and add it to  $R$ . Backtrack, and go to Step 1.
    - (c) If  $R$  is satisfiable and  $S$  has an empty clause, output the values of the  $Q$  variables in any satisfying solution of  $R$  that respects the current fixing of  $Q$  variables, and stop.
    - (d) ( $R$  is satisfiable and all  $Q$  variables are fixed.) Solve  $S$ . If  $S$  is unsatisfiable, output the values of the  $Q$  variables, and stop.
  7. Go to Step 1.
- 

Figure 1: The solution algorithm.

count the nonnegated occurrences of  $q$  in  $R$  and the negated occurrences in  $S$  in all clauses of length  $i$ . Let this number be  $p_q(i)$ . Analogously, let  $n_q(i)$  be the number of negated occurrences of  $q$  in all clauses of  $R$  of length  $i$  and nonnegated occurrences of  $q$  in  $S$  in clauses of length  $i$ . Define a vector  $h_q$  whose  $i$ th element is

$$h_q(i) = p_q(i) + n_q(i) - \frac{1}{3} \cdot |p_q(i) - n_q(i)| \quad (2)$$

We select the variable  $q$  with the lexicographically largest vector  $h_q$ . The selected variable  $q$  is set to *True* if  $p_q(i) > n_q(i)$ , otherwise, it is set to *False*. The selected variable and truth value aim at inducing satisfiability in  $R$  and unsatisfiability in  $S$ .

In applications, one often must solve instances of Q-ALL SAT that are obtained from one formula (1) by fixing some of the variables in  $R$  and  $S$ . For effective solution of all such cases by the algorithm in Fig. 1, we generate that algorithm by a compiler as follows. The compiler partitions the variable set  $Q \cup X \cup Y$  of  $R \wedge S$  into two sets  $V_E$  and  $V_N$  that induce two CNF formulas  $(R \wedge S)_E$  and  $(R \wedge S)_N$ . The formula  $(R \wedge S)_E$  results from  $R \wedge S$  by deleting all literals arising from  $V_N$  from the clauses,  $(R \wedge S)_N$  results from deleting all literals in  $R \wedge S$  arising from  $V_E$ . The partition is done such that the CNF formula  $(R \wedge S)_E$  has as many variables as possible and such that  $(R \wedge S)_E$  is 2CNF or has Horn form after suitable complementing of some variables. Consider  $R \wedge S$  where all variables  $V_N$  are fixed to some truth values. The resulting formula can be solved in linear time since it is 2CNF or is Horn after suitable complementing of variables, and since these properties are maintained under deletion of variables or clauses. Thus, the solution algorithm (see Step 3 in Fig. 1) solves the reduced formula by a linear-time algorithm for both the 2CNF and Horn cases if all variables  $V_N$  are fixed.

#### 4. Computational Results

The solution algorithm has been implemented in a logic programming software (Leibniz System [2003]), which contains the compiler. The compiler can identify if one of the easy subclasses is at hand. The computational results were obtained on a Sun ULTRA 5 workstation.

For initial tests, we designed two sets of benchmark problems that, depending on choice of parameters, can be made progressively more difficult. At the same time, the problems are structured and, by their design, are closer to real-world problems than randomly generated instances.

The first problem set is a navigation problem where a robot has to navigate through a grid with obstacles. The possible positions of obstacles are constrained by some conditions. The instances differ by randomly placing some obstacle on the grid. Each instance has 64  $Q$  variables, 64  $X$  variables, and 128  $Y$  variables. The CNF formula  $R$  contains 924 to 926 clauses, while  $S$  contains 595 to 597 clauses. The problem asks one to answer the question whether obstacles can be placed so that the robot cannot reach the desired destination.

The second problem set models a problem where one must find a successful strategy in a game. The starting position of the player and the opponent are randomly generated. An instance in the set *game- $i$ - $j$*  has  $i$   $Q$  variables,  $255 + j$   $Y$  variables and no  $X$  variables. The CNF formula  $R$  consists of a single clause only. The number of clauses in  $S$  ranges from 700 to 900.

For all instances, the compilation takes less than 3 sec. Table 1 shows the results of the solution algorithm of Fig. 1 for the robot instances. The first three instances have a solution, that is, the quantified formula (1) evaluates to *True*. The number of  $R$  clauses in the third column is the number  $R$ -conflict clauses, that is the number of learned clauses due to unsatisfiability of  $R$ . The

instance	solution	no. $R$ clauses	no. $S$ clauses	time (sec)
robot_8_1	yes	1	0	1579
robot_8_2	yes	2	2	1988
robot_8_3	yes	0	0	0
robot_8_4	no	6	6	40
robot_8_5	no	3	2	241
robot_8_6	no	1	2	5

Table 1: The results of the algorithm described in Fig. 1 for the robot instances

instance	solution	time (sec)
robot_8_1	<i>True</i>	84
robot_8_2	<i>True</i>	33
robot_8_3	<i>True</i>	57
robot_8_4	<i>False</i>	54
robot_8_5	<i>False</i>	5
robot_8_6	<i>False</i>	7

Table 2: The results of a 2cut algorithm for the robot instances

number of  $S$  clauses in the fourth column is the number of sharpened  $S$ -conflict clauses that have been added to  $R$ . The run times differ significantly; the run time for robot\_8\_2 is about 33 min, while the run time for robot\_8\_3 is below 1 sec. On average, the robot instances that evaluate to *False* are solved faster than the instances that evaluate to *True*. Learning of sharpened  $S$ -conflict clauses prunes the search tree effectively. In case of robot\_8\_3, the branching rule leads rapidly to a solution, and no  $S$ -conflict clauses are added. In case of the instances robot\_8\_1 and robot\_8\_2, which have solutions as well, the branching rule is not an effective guide.

An analysis of the robot instances reveals that they belong to the subclass 2cut Q-ALL SAT. After running the general algorithm of Fig. 1, we applied a special version for 2cut Q-ALL SAT to the robot instances. Table 2 shows that the same instances are now solved within at most 84 sec.

The game instances generally do not fall within one of the easier subclasses. We list the computational results in Table 3 for the algorithm described in Section 3. For each problem set game\_ $i$ - $j$ , we have tested 12 instances. The third and fourth column show the average number of learned  $R$ -conflict clauses and  $S$ -conflict clauses, respectively. The average run time for each problem set game\_ $i$ - $j$  is given in the last column. The solution algorithm was aborted when the run time exceeded 1 hour. The number of aborted instances for each problem set is given in the column *unsolved*. In all instances, the algorithm added slightly more  $S$ -conflict clauses than  $R$ -conflict clauses. Observe that the CNF formula  $R$  originally consists of a single clause. Therefore, the  $R$ -conflict clauses can be learned due to added  $S$ -conflict clauses only. The high average run time of the game\_15\_20 instances has been caused by one instance with a run time of about 30 min. In all other instances, the run time is in the order of the given average time. Almost all instances are solved within 2 min, most are solved within 10 sec.

At this time, we have no performance data of other QBF algorithms on the robot and game instances. Thus, we can only say that, given the difficulty of Q-ALL SAT, and given the size of

instance	solution (yes/no)	no. $R$ clauses	no. $S$ clauses	unsolved	time (sec)
game_15_20	6/6	136	137	0	161.3
game_15_50	5/7	38	39	0	12.4
game_15_100	4/8	4	6	0	3.9
game_15_150	4/8	4	5	0	4.1
game_20_20	7/4	244	246	1	57.3
game_20_50	10/2	132	134	0	64.6
game_20_100	9/3	16	18	0	12.8
game_20_150	8/4	4	5	0	5.4
game_25_20	8/0	8	12	4	1.4
game_25_50	8/3	50	51	1	34.8
game_25_100	8/4	5	8	0	7.6
game_25_150	10/2	2	5	0	5.7

Table 3: Results of the algorithm of Fig. 1 for the game instances

the instances, the run times, and the fact that the algorithm solves all robot instances and 138 out of a total 144 game instances (=96%), one might judge the algorithm to be quite effective.

## 5. Summary and Ongoing Work

This paper defines the problem Q-ALL SAT, identifies special classes and their complexity, describes a solution algorithm, and reports first results for the instances of two nontrivial problem classes. The computational results seem to indicate that the algorithm is quite effective.

This paper is a first report of an ongoing research project covering problems of quantified Boolean formulas. The project is to create an algorithm for Q-ALL SAT that rapidly and reliably solves all instances of important application classes up to a reasonable number of  $Q$  variables, other variables, and clauses. Part of that effort is identification of such application classes.

Q-ALL SAT ignores any costs associated with the values selected for the  $Q$  variables. Applications typically give rise to such costs, and thus lead to several variations of Q-ALL SAT. The ongoing research is developing solution algorithms for these problems as well.

Finally, some applications give rise to quantified logic problems at the third level of the polynomial hierarchy. Here, too, various various version involving costs and learned formulas may occur, and heuristic approaches are useful. For some results of the above extensions of Q-ALL SAT, see Truemper [2003].

## References

- [1979] Aspvall, B., Plass, M. F., Tarjan, E.: A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters* 8(3) (1979) 121–123
- [1997] Bayardo Jr., R. J., Schrag, R. C.: Using CSP look-back techniques to solve real world SAT instances. *Proceedings of the 14th National Conference on Artificial Intelligence* (1997) 203–208

- [2003] Cadoli, M., Schaerf, M., Giovanardi, A., Giovanardi, M.: An Algorithm to Evaluate Quantified Boolean Formulae and its Experimental Evaluation. *Journal of Automated Reasoning* (2003) to appear
- [1984] Dowling, W. F., Gallier, J. H.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming* 1 (1984) 267–284.
- [1995] Eiter, T., Gottlob, G.: The Complexity of Logic-based Abduction. *Journal of the Association for Computing Machinery* 42 (1995) 3–42.
- [1997] Eiter, T., Gottlob, G., Leone, N.: Abduction from Logic Programs: Semantics and Complexity. *Theoretical Computer Science* 189(1-2) (1997) 129-177.
- [2002] Giunchiglia, E., Narrizano, M., Tacchella, A.: Learning for Quantified Boolean Logic Satisfiability. *Proceedings of the 18th National Conference on Artificial Intelligence* (2002).
- [2003] Giunchiglia, E., Narrizano, M., Tacchella, A.: Backjumping for Quantified Boolean Logic satisfiability. *Artificial Intelligence* 145(1) (2003) 99.
- [1999] João, P., Marques-Silva, J. P., Sakallah, K. A.: GRASP – A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers* 48(5) (1999) 506–521.
- [1995] Kleine Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. *Information and Computation* 117(1) (1995) 12–18.
- [2003] Leibniz System (2003) Version 5.2, Leibniz, Plano, Texas.
- [2003] Plaisted, D., Biere, A., Zhu, Y.: A Satisfiability Procedure for Quantified Boolean Formulae *Discrete Applied Mathematics* (to appear)
- [1989] Poole, D.: Normality and Faults in Logic-Based Diagnosis. *Proceedings of the 11th International Joint Conference on Artificial Intelligence* (1989) 1304—1310.
- [2003] Remshagen, A., Truemper, K.: Complexity of the Futile Questioning Problem, working paper (2003)
- [2001] Rintanen, J.: Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formula. *International Conference on Logic Programming, Artificial Intelligence and Reasoning* (2001) 362–376
- [1999] Rintanen, J.: Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence* 10 (1999) 323–352
- [1976] Stockmeyer, L. J.:
- [1999] Straach, J., Truemper, K.: Learning to Ask Relevant Questions. *Artificial Intelligence* 111 (1999) 301–327
- [1998] Truemper, K.: *Effective Logic Computation*. Wiley (1998)
- [2003] Truemper, K.: *Design of Intelligent Systems*. (In preparation, anticipated completion 2003)

- [1997] Zhang, H.: SATO: An Efficient Propositional Prover. Proceedings of the International Conference on Automated Deduction (1997)
- [2001] Zhang, L., Madigan, C. F., Moskewicz M. H., Malik, S.: Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. Proceedings of the International Conference on Computer Aided Design (2001)
- [2002] Zhang, L., Malik, S.: Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. Proceedings of 8th International Conference on Principles and Practice of Constraint Programming (2002)
- [2002] Zhang, L., Malik, S.: Conflict Driven Learning in a Quantified Boolean Satisfiability Solver. Proceedings of the International Conference on Computer Aided Design (2002)