Sacre : a Constraint Satisfaction Problem Based Theorem Prover

Jean-Michel Richer, Jean-Jacques Chabrier
LIRSIA, Burgundy University
U.F.R des Sciences et Techniques
Bâtiment Mirande - 9, Avenue Alain Savary, B.P. 400
21011 Dijon Cedex - France
{richer,chabrier}@crid.u-bourgogne.fr

Abstract

The purpose of this paper is to present a new approach for solving first-order predicate logic problems stated in conjunctive normal form. We propose to combine resolution with the Constraint Satisfaction Problem (CSP) paradigm to prove the inconsistency or find a model of a problem. The resulting method benefits from resolution and constraint satisfaction techniques and seems very efficient when confronted to some problems of the CADE-13 competition.

Introduction

From a general point of view we can classify methods for solving first-order predicate calculus problems stated in conjunctive normal form, into two categories. The first one is *consistency searching* or *proof searching* and is syntax-oriented. It consists of raising a contradiction from a set of clauses by applying inference rules. For example, Otter (McCune 94) uses *resolution, unit-resolution, hyperresolution*, while Setheo (Loveland 78) is based on *model elimination*.

The second one, *satisfiability checking*, also called *model finding*, is related to semantics and tries to find a model or a counterexample of a problem. In this last category we can draw a distinction between *saturation* and *extension* approaches. In the former case, we iteratively generate ground instantiations of the problem and test ground clause sets for unsatisfiability with a propositional calculus prover. In the latter case, we try to build a model of the problem by assuming new ground facts.

It seems, that in the case of propositional calculus, semantic methods are more efficient when compared to syntactic ones, whereas in predicate calculus, this is quite the contrary. One of the first satisfiability approaches was the one of Gilmore (Gilmore 60) which was a saturation approach and proved to be very inefficient. We believe it is because this kind of approach has to tackle with the whole Herbrand base while only a part of it is necessary.

Another approach is the Satchmo theorem prover (Manthey 88) which uses syntactic and semantic features to solve problems. It can be qualified as an extension approach. Satchmo is based on the *model generation* reasoning paradigm. However, Satchmo suffers from certain drawbacks. The first one is *range restriction* requiring that each head variable must occur in the body of a clause. The second one is the fact that Satchmo might choose a clause irrelevant to the current goal to be solved and thus would cause unnecessary model candidate extensions. This may result in a potential explosion of the search space. However, some improvements can be made, such as *relevancy testing* (Loveland 93) to avoid unnecessary case splittings.

Apart from those semantic approaches, Finder (Slaney 95) searches for finite models of first order theories presented as sets of clauses. Falcon (Zhang 96), for which model generation is viewed as constraint satisfaction, constructs finite algebras from given equational axioms. Finite models are able to provide some kind of semantic guidance that helps refutation-based theorem provers to find proofs more quickly (Slaney 94).

It is also possible to combine *resolution* with rewrite techniques so as to guide the search and design more efficient inference rules, such as the *problem reducation format* (Loveland 78) or the *simplified problem reduction format* (Plaisted 82), that permits to delete unachievable subgoals, or its extension the *modified problem reduction format* (Plaisted 88).

The key novelty introduced in this paper is the combination of *resolution* with the *Constraint Satisfaction Problem* (CSP) paradigm, so as to solve first-order predicate calculus problems, stated in conjunctive normal form. This combination is not fortuitous. First, consistency searching and model finding are both common problems related to logic and CSP. Second, the CSP techniques have proved to be very powerful to solve large combinatorial problems by applying stategies and heuristics that help guide the search and improve the resolution process by efficiently pruning the search space.

The resulting method, called *Sacre* [1] is based on a unique forward chaining rule and combines constraint satisfaction heuristics and techniques together with *resolution* and is able to prove the inconsistency or find a model of a problem. It is, to our knowledge, the first attempt of this kind ever tried in this direction.

The paper is organized as follows : in section 2, we will set forth some basic definitions of constraint satisfaction

[1] for SAtisfaction de Contraintes et REsolution - Constraint satisfaction and resolution

problems. The next section is devoted to the *Sacre* system. Section 4 provides some typical examples in order to outline the domain of application of *Sacre* . The last section exhibits some results for some of the problems of the CADE-13 competition.

Though the constraint satisfaction paradigm has been widely used to efficiently solve propositionnal calculus problems, so far, little research has been carried out on the resolution of first-order logic problems considered as CSPs. The main reason is due to the fact that first-order logic problems are not well suited to a CSP approach and tne major hurdles encountered

Constraint satisfaction problems

In this section we recall some basic notions related to constraint satisfaction problems (CSP). For a far more general and complete introduction please read (ECRC 92). We shall also assume that the reader is familiar with propositional and predicate calculus (see (Chang 73)).

Definition

The CSP framework was first introduced in (Montanari 74). This formalism enables the user to state problems in a declarative way in terms of relations between variables interpreted over different computational domains.

Definition 1 Constraint Satisfaction Problem . *A CSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is defined by*

- *a set $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$ of n variables;*
- *a set $\mathcal{D} = \{d_1, d_2, \cdots, d_n\}$ of n finite domains of $\mathcal{X}$;*
- *a set $\mathcal{C} = \{c_1, c_2, \cdots, c_m\}$ of m constraints. Each constraint $c_i = \{x_{i1}, \cdots, x_{ik}\}$ is a relation between the variables $x_{i1}, \cdots, x_{ik}$ where $i1, \cdots, ik$ belong to $\{1, n\}$.*

A constraint is said to be satisfied if there exists an assignment of its variables such that the relationship between the variables holds, otherwise a constraint is said to be violated. Solving a CSP consists in satisfying all the constraints of $\mathcal{C}$. Given a CSP, we can check if there exists a solution (consistency checking), find a solution, or all the solutions (model finding), find an optimal solution for a given criterion (optimization).

Example 1- Let us take a short example of a CSP :

$$(CSP_1)\begin{cases} \mathcal{X} = & \{x_1, x_2, x_3\}, \\ \mathcal{D} = & \{d_1, d_2, d_3\}, \text{ with } d_1 = d_2 = d_3 = \{0, 1, 2\} \\ \mathcal{C} = & \{x_1 + x_2 = x_3\}, \end{cases}$$

whose solutions are :

$$S_1 = \{\ (0,0,0),\ (0,1,1),\ (1,0,1),\ (1,1,2),\ (0,2,2),\ (2,0,2)\ \}$$

CSP in propositional calculus

Propositional calculus problems can efficiently be solved following a constraint satisfaction approach. It is possible to state any problem in conjunctive normal form in terms of a CSP. Each propositional variable becomes a member of $\mathcal{X}$ and ranges over the boolean domain $\{false, true\}$. Each clause is represented as a cardinality constraint. In the remainder of this paper we will note $[\![\, \alpha,\ \beta,\ L]\!]$ the cardinality operator (see (Van Hentenryck 91)), where α, β are two natural numbers, and L is a disjunction of literals. The original definition has been slightly modified to our convenience. On a semantic point of view the cardinality operator compels at least α and at most β literals of L to be interpreted to true. For example the clause $\neg p \vee q \vee r$ can be represented as $[\![\, 1,\ 3,\ \neg p, q, r]\!]$.

The *Score* method (see (Chabrier 95) and (Chabrier 96) for more details), which was developed in our laboratory is an efficient method for solving propositional problems stated in terms of cardinality constraints. The main novelty in *Score* is the combination of a constructive or systematic complete method (backtrack) with an incomplete repair method (local search).

The *Sacre* approach

Score , that has proved to be fairly efficient for random SAT problems (3-SAT) and structured problems (Ramsey, Pigeon-hole), led us naturally therefore to venture out beyond the limits of propositional calculus. To some extent, *Sacre* can be considered as an attempt at extending *Score* to first-order.

The first hurdle we had to overcome consisted in representing first-order predicate calculus problems into the CSP framework. Indeed, in propositional calculus, variables range over the finite boolean domain, while in predicate calculus, as we shall see later on, variables range over the potentially infinite Herbrand universe. (Note that *Sacre* can work on a subset of the Herbrand universe by restricting domains).

We now reconsider the definition of constraint satisfaction problems and alter it to take into account the special role held by first-order predicates.

Definition 2 Extended domain. *Definition corresponds to the standard resolution framework of CSPs, for which all domains are finite. We informally introduce here the notion of* extended domains*, for CSP, in which domains are able to be extended by constraints propagation during resolution.*

Example 2 Remember CSP_1 given in section . For a standard resolution, the assignment $x_1 = 1$ and $x_2 = 2$ is inconsistent, for $x_1 + x_2 = 3$ is not a member of the domain d_3 of x_3. If we solve this problem according to an extended approach, the previous assignment is legal and the value 3 is added to d_3. The solutions are then

$$S_{ext} = S_1 \cup \{\ (1,2,3),\ (2,1,3),\ (2,2,4)\ \}$$

By iterating this resolution process, domains d_1, d_2, d_3 will be extended to $d_1 = d_2 = d_3 = \mathbb{N}$.

Logic	CSP
clause	constraint
predicate	variable
unit-clause/atom/term	value

Table 1: Duality between logic and CSPs

CSP transformation

We now show that it is possible to transform a predicate calculus problem (P) stated in conjunctive normal form into an extended CSP over the Herbrand domain. We shall therefore point out the correspondence between CSPs and predicate calculus problems (cf. table 1).

The main idea behind *Sacre* is the translation of the *resolution principle* (Robinson 65) into the CSP framework. Binary resolution yields a new clause, called a resolvent, by deleting complementary unifiable literals of two parent clauses. The generation of the empty clause leads to inconsistency. But to generate the empty clause, one needs to infer two unifiable complementary unit clauses (for example $\neg p(a, X)$ and $p(Y, f(Z))$).

When we come to consider CSPs, the *resolution principle* is expressed as the impossibility for a value to be a member of two complementary domains, for example $d(x_p^-)$ and $d(x_p^+)$, associated with the predicate p, and respectively related to negative and positive literals of p, such that $d(x_p^-) \cap d(x_p^+) = \emptyset$.

Definition 3 Transformation of P into a CSP$^{\mathcal{H}}_{ext}$. *A first-order predicate calculus problem P stated in conjunctive normal form can be transformed as a CSP with extended domains over the Herbrand universe $\mathcal{H}$ of P (we note CSP$^{\mathcal{H}}_{ext}$), by applying the following rules :*

- *for each predicate p of P :*
  - *$\mathcal{D}$ is made up of the domains $d(x_p^+)$ and $d(x_p^-)$ defined over $\mathcal{H}$;*
    - *$d(x_p^+) = \{t,$ such that $p(t)\}$, represents the set of terms that are true under a partial interpretation of P,*
    - *$d(x_p^-) = \{t,$ such that $\neg p(t)\}$, represents the set of terms that are false under a partial interpretation of P,*
  - *$\mathcal{X}$ is made up of the variables x_p^+ and x_p^-, respectively ranging over $d(x_p^+)$ and $d(x_p^-)$, and related to atoms of the form $p(t)$ and $\neg p(t)$.*
  - *$\mathcal{C}$ initially contains the consistency constraints of the form $x_p^+ \cap x_p^- = \emptyset$ that maintains consistency at the logical level;*
- *unit clauses of P initially define the domains :*
  - *$d(x_p^+) = \{t_0\}$, for each unit clause $p(t_0)$ of P,*
  - *$d(x_p^-) = \{t_0\}$, for each unit clause $\neg p(t_0)$ of P,*
- *non unit clauses are transformed into* cardinality constraints. *A clause $p_1(t_1) \vee \cdots \vee p_k(t_k)$ is transformed into $[\![1,\, k,\, p_1(t_1), p_2(t_2), \cdots, p_k(t_k)]\!]$.*

remark 0.1 *Obviously, cardinality constraints are not directly expressed over the set of variables $\mathcal{X}$. In a certain sense, one could argue that the corresponding problem generated by the transformation is not a* pure *CSP. This is due to the special role held by variables inside terms. However, it is possible to express cardinality constraints according to x_p^+ and x_p^-. We chose here to keep the cardinality constraint notation for a comprehensive reading. For a more detailed explanation, please read (Richer 97).*

Example 3 The transformation of a predicate calculus problem into an extended CSP is illustrated figure 1.

$$(P) \quad \left\{ \begin{array}{l} p(a) \\ \neg p(X) \vee q(f(X)) \\ \neg q(f(f(X))) \end{array} \right.$$

$$(PSC_e^{\mathcal{H}}) \quad \left\{ \begin{array}{l} \mathcal{X} = \{\ x_p^-, x_p^+, x_q^-, x_q^+\ \}; \\ \mathcal{D} = \{\ d(x_p^-) = \emptyset,\ d(x_p^+) = \{a\}, \\ d(x_q^-) = \{f(f(X))\},\ d(x_q^+) = \emptyset\ \} \\ \mathcal{C} = \{\ [\![1,\, 2,\, \neg p(X), q(f(X))]\!]\ \} \end{array} \right.$$

Figure 1: Logic problem and related CSP$^{\mathcal{H}}_{ext}$

Furthermore, we had to tackle to the problem of solving cardinality constraints. Though it is possible to satisfy all the cardinality constraints of a predicate calculus problem, it is not sufficient enough to prove the satisfiability of a problem, while it is the case in propositional calculus. Consider the following inconsistent problem: $p(a)$, $\neg p(X) \vee p(f(X))$, $\neg p(f(f(a)))$, which can be transformed into the corresponding constraint problem : $[\![1,\, 1,\, p(a)]\!]$, $[\![1,\, 2,\, \neg p(X), p(f(X))]\!]$, $[\![1,\, 1,\, \neg p(f(f(a)))]\!]$. A consistent instantiation could be $\{p(a),\ p(f(a)),\ \neg p(f(f(a)))\ \}$ that satisfies all the constraints. This suggest that solving a problem in predicate calculus requires to use a deduction process that builds up the domains of all variables. This can be achieved by falsifying $n-1$ atoms of a cardinality constraint of the form $[\![1,\, n,\, p_1(t_1), \cdots, p_n t_n)]\!]$. If we can falsify atoms $p_1(t_1)$ to $p_{n-1}(t_{n-1})$, we can deduce, by satisfying the cardinality constraint, that the atom $p_n(t_n)$ must be true.

Resolution

The resolution procedure underlying the implementation of *Sacre* is based on a unique forward chaining rule. The principle is to build a partial interpretation by iteratively satisfying constraints. The partial interpretation may be viewed as an attempt at constructing a counterexample for refuting the given hypothesis. New atoms (or values) are then added to their related domains to perform further deductions and increase the partial interpretation. However, we are not forced to confine ourselves to ground atoms. For example, we can add the value X to any domain, if X is a variable. This prevents us from enumerating the entire Herbrand universe by using subsumption. To give the reader a yet clearer view of the resolution procedure, we shall refer to figures 2 and 3.

Algorithm CONSISTENCYCHECKING(P)
input : a $CSP^{\mathcal{H}}_{ext} = \{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$
output : true if the problem P is consistent, false otherwise.
begin
$V = \{x_p \in \mathcal{X}$ such that $d(x_p) \neq \emptyset\}$
$consistent = true$
while $(V \neq \emptyset)$ and $consistent$ **do**
choose $x_p \in V$
$V = V - \{x_p\}$
$C(x_p) = \{$ set of constraints such that $\neg p$ appears in the constraint $\}$
while $C(x_p) \neq \emptyset$ and $consistent$ **do**
choose $c \in C(x_p)$
$C(x_p) = C(x_p) - \{c\}$
$consistent$ = SatisfyConstraint(c)
end
end
return $consistent$
end

Figure 2: Consistency checking in *Sacre*

Function SATISFYCONSTRAINT(c)
input : $c = [\![\, 1, n, p_1(t_1), \cdots, p_n(t_n)]\!]$
output : true, if P is consistent, false otherwise
begin
for any most general unifier σ falsifying $n-1$ atoms $p_1(t_1), \cdots, p_{i-1}(t_{i-1}), p_{i+1}(t_{i+1}), \cdots, p_n(t_n)$ **do**
$\mu = \sigma(t_i)$
if $\mu \in d(x_{\neg p_i})$ **then return** $false$ **endif**
$d(x_{p_i}) = d(x_{p_i}) \cup \{\mu\}$
if $x_{p_i} \notin V$ **then** $V = V \cup \{x_{p_i}\}$ **endif**
end
return $true$
end

Figure 3: Constraint satisfaction

Heuristic and strategy tuning

Sacre is provided with many heuristics and restriction techniques. For example, when choosing a variable it is possible to select a min-domain (choose the variable with the minimum number of values in its domain), max-domain, min-constraint or max-constraint heuristic. New values can be rejected if their arity exceeds an upper bound or, the satisfaction of a constraint can be ended if a maximum number of values has been reached. The major difficulty is to combined these heuristics together so as to sufficiently decrease the search space without restricting it to a space hiding the solution. It is also worth mentioning that it is possible to choose between a depth-first or a breadth-first search.

Other features

Sacre also applies to problems written in first-order logic with equality. Demodulation with Lex Recursive Path Ordering has been implemented. Further versions will probably integrate paramodulation. One main feature of *Sacre* is the possibility to *direct* the search by orientating cardinality constraints (see section). This ability, as in Prolog, is intended to compute a solution with very little searching and has the potential of being quite efficient compared to a non-directed approach. For example, problem NUM084.010 could not be solved in less than one second without taking this feature into account (see table 2).

Soundness and completeness

Unfortunately *Sacre* is incomplete, but a lack of completeness can generally lead to more efficiency. This seems to be the case. Incompleteness is due to cardinality constraints that act as the *unit-resulting resolution* inference rule (Chang 70), which is sound but incomplete. Efficiency also stems from the lack of a *splitting rule* that would ensure completeness but would cause a loss of efficiency.

Applications

The following two examples afford a sufficient indication of the way in which problems can be stated and solved in *Sacre* either with a proof search or a model finding approach.

Consistency checking

Consider the problem of proving, that, in a monoid, if $x \circ x$ is the identity for every x, then $\circ$ is commutative. To formulate this problem we will use the ternary predicate $p(x, y, z)$ whose meaning is $x \circ y = z$. The problem can then be expressed by the following assertions (see figure 4).

```
Problem Group;

Clauses
  (1)   p(e,X,X).
  (2)   p(X,e,X).
  (3)   p(X,X,e).
  (4)   p(U,Z,W) | -p(X,Y,U) |
             -p(Y,Z,V) | -p(X,V,W).
  (5)   p(X,V,W) | -p(X,Y,U) |
            -p(Y,Z,V) | -p(U,Z,W).
  (6)   p(a,b,c).
  (7)  -p(b,a,c).

Find  consistency;

List p+, p-;
```

Figure 4: Group theory problem

The aim of this application is to demonstrate the ability of our solver to check the consistency of problems. For this problem, *Sacre* was able to build a counterexample such that the value (b, a, c) belongs to $d(x_p)$ and $d(x_{\neg p})$. This is equivalent to raising a contradiction and generating the empty clause by producing two unit clauses : $p(b, a, c)$ and $\neg p(b, a, c)$ (see figure 5).

```
consistency ? No
 p+ = { (b,a,c), (c,a,b), (b,c,a),
        (a,c,b), (c,b,a), (a,b,c),
        (e,X,X), (X,e,X), (X,X,e) }
 p- = { (b,a,c) }

inconsistent(s) value(s) for Xp = { (b,a,c) }
```

Figure 5: consistency checking for the `Group` problem

Model Finding

The map coloring problem consists in coloring rectangles on a map with three different colors (red, green, blue) such that two adjacent rectangles are not allowed to be painted with the same color. Our aim is to provide the solutions of this problem.

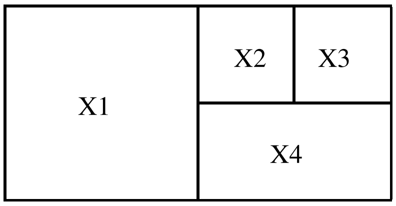

Figure 6: Map coloring

This application is very interesting for two reasons. First, in regard to constraint satisfaction, it proves that it is possible to state predicate calculus problems in terms of constraints and domains (see figure 7). We are here confronted to a *standard* CSP whose domains are fixed and won't be extended. Second, it is a short example that illustrates model finding. As mentioned earlier, the unique constraint of the problem has been stated as a *directed* constraint (<−). The solutions are given figure 8.

```
Problem ColorMap;

Domains
color+ = {
   (red, green), (red, blue ),
   (green, red), (green, blue ),
   (blue, red), (blue, green ) }

  color- = { (X,X) }

Clauses
coloring( X1, X2, X3, X4 ) <-
   color(X1,X2), color(X1,X4),
   color(X2,X3), color(X2,X4),
   color(X3,X4).

Find coloring(X1,X2,X3,X4);
```

Figure 7: The Map Coloring problem

```
  coloring+ = { (blue,green,blue,red),
                (blue,red,blue,green),
                (green,blue,green,red),
                (green,red,green,blue),
                (red,blue,red,green),
                (red,green,red,blue) }
```

Figure 8: Model finding for the `coloring` problem

Results

The *Sacre* method was implemented in the programming language C under Unix. In order to point out the real interest of our approach and the efficiency of our solver, we decided to tackle to some problems of the CADE-13 Automated Theorem Proving System competition (CADE13).

We present in this section some comparatives results between *Sacre* and two other theorem provers Otter and Setheo. Otter is one of the most complete and efficient theorem prover using inference rules based on *resolution* (unit resolution, binary resolution, hyperresolution). Setheo uses *model elimination*. Otter and Setheo were chosen for comparison because they took part in the CADE-13 competition and obtained the best results.

Table 2 shows the results for some problems. Times are given in seconds. A resolution time of one second means that it took one or less than one second to prove inconsistency. Blanks mean that the problem could not be solved within 300 seconds. The tests were run on a Sun Sparc Ultra 1 workstation. The columns Otter and Setho respectively provide the results of Otter 3.0.4. in auto mode, and the best results of Setheo 3.3. with the ”-dr” or ”-wdr” option.

Sacre performs well on some kind of problems but there remain some problems out of *Sacre* 's scope. The problems PLA004-1 and PLA011-2 can not be solved by *Sacre* or Otter. Implementing depth-first iterative deepening (Korf 85) could obviously be of great value from this perspective.

Conclusion and future work

In this paper we have shown that there exists a certain correspondence between first-order calculus problems in conjunctive normal form and constraint satisfaction problems. A new species of solvers can be built on this paradigm. Not only did we prove the validity of our approach but also its effectiveness with the *Sacre* prover. Improvements are still possible, and much work is under completion. We think it is possible to implement an oracle (like the autonomous mode of Otter(McCune 94)) able to determine the best strategies and heuristics for solving a given problem. Moreover, the introduction of a *splitting rule* would ensure completeness and would probably be more efficient than the SATCHMO splitting rule because we are not forced to work with ground terms. In the case of predicate calculus problems with non-recursive clauses and non-extended domains (as it is the case for the Map Coloring problem), we are confronted to *standard* CSP. Moreover, the specification of the domains of predicates, instead of their computation, combined with a non-domain extension, can greatly reduce the search space.

Problem	Otter	Sacre	Setheo	Problem	Otter	Sacre	Setheo
BOO006-1	44	1	1	RNG040-1	1	1	1
BOO012-1	130	5	1	SET008-1	5	1	1
BOO016-1	21	6	13	SET061-6	95	5	
GRP048-2	1	5	48	SET063-6	31	1	
NUM284-1.010	1	1		SYN200-1	2	1	1
PLA004-1			1	SYN202-1	5	1	1
PLA011-2			1	SYN271-1	5	2	1
RNG005-1	6	1	208	Group	1	1	1
RNG038-2	3	12	106	ColorMap	1	1	1

Table 2: Comparison between Otter, *Sacre* , Setheo.

Further experimentation and improvements will determine how worthwhile are approcah is.

Availability

Sacre is available at the following url :

`http://recife.u-bourgogne.fr:8081/~richer/SACRE/Sacre.html`

and is provided with a graphic user interface written in Java. The current implementation works on SUN workstations with Solaris.

References

Special Issue : The CADE-13 Automated Theorem Proving System Competition, Journal of Automated Reasoning, vol 18, 2, April, 1997.

CHANG C.-L., *The unit proof and the input proof in theorem proving*, *Journal of the ACM*, vol 17, n° 4, pp 698–707, 1970.

CHANG C.-L., LEE R. C.-T., *Symbolic logic and mechanical theorem proving*, Academic Press, Inc., London, 1973.

CHABRIER J., JULIARD V., CHABRIER J-J., *Score (FD/B) : An Efficient Complete Local-Based Search Method for Satisfiability Problems*, In *CP'95 Workshop on Studying and Solving Really Hard Problems*, Cassis, France, 1995.

CHABRIER J., RICHER J-M., CHABRIER J-J., *Resolution of Structured SAT Problems with Score (FD/B)* , In *Proceedings of the CP'96 Workshop on Constraint Programming Applications*, Cambridge, Massachusetts, U.S.A., 1996.

FRÜHWIRTH T. and al., *Constraint logic programming - An informal introduction*, ECRC internal report, ECRC-92-6i, Munich, Germany, 1992.

GILMORE P. C., *A proof method for quantification theory : its justification and realization*, IBM JRD, pp 28–35, 1960.

KORF R. E., *Depth-first iterative-deepening : an optimal admissible tree search* , Artificial Intelligence, vol 27, n° 1, pp 97–109, September 1985.

LOVELAND D. W., *Automated Theorem Proving : A Logical Basis*, North Holland, New York, 1978.

LOVELAND D. W., REED D. W., WILSON D. S., *SATCHMORE : SATCHMO with RElevancy*, *Journal of Automated Reasoning*, vol 14, n° 2, pp 325–351, 1995.

MANTHEY R., BRY F., *Satchmo : A theorem Prover Implemented In Prolog* , In *Proceedings of the International Conference on Automated Reasoning*, CADE-88, vol 310 of LNCS, pp 415-434, Springer-Verlag, 1988.

MCCUNE W. W., *Otter 3.0 Reference Manual and Guide*, Argonne National Laboratory, Mathematics and Computer Science Division, UC-405, ANL-94/6, January, 1994.

MONTANARI U., *Networks of Constraints: Fundamental Properties and Applications to Picture Processing*, Information Science, vol. 7, n° 2, pp. 95–132, 1974.

PLAISTED D. A., *A Simplified Problem Reduction Format*, Artificial Intelligence, vol 18, pp 227-261, 1982.

PLAISTED D. A., *Non-Horn Clause Logic Programming Without Contrapositives*, Journal of Automated Reasoning, vol 4, pp 287–325, 1988.

RICHER J.-M., *Technical Report - Constraint representation of logic problems*, LIRSIA, Burgundy University, December, 1997.

ROBINSON J. A., *A Machine-Oriented Logic Based on the Resolution Principle*, Journal of the ACM, vol 12, pp 23–41, 1965.

SLANEY J., LUSK E., MCCUNE W., *SCOTT : Semantically Constrained OTTER System Description*, Technical Report TR-ARP-3-94, Australian National Laboratory, 1994.

SLANEY J., *FINDER (Finite Domain Enumerator) : version 3.0 notes and guides*, Center for Information Science Research, Australian National University, Canberra, 1995.

VAN HENTENRYCK P., and DEVILLE Y., *The Cardinality Operator : A New Logical Connective for Constraint Logic Programming*, In K. Furukawa, editor, Proc. of the 8th ICLP, Paris, France 24-28 June 1991, pages 745–759, The MIT Press, Cambridge, Mass. 1991.

ZHANG Jian, *Constructing Finite Algebras with FALCON*, Journal of Automated Reasoning, vol 17, pp 1–22, 1996.