Detecting Consistency of Database Rules
by Adapting Theorem Proving Methods

Francois Bry
Rainer Manthey

European Computer-Industry
Research Centre GmbH
Arabellastr. 17
D-8000 Muenchen 81
West Germany
# Table of Contents

1. Introduction  
2. Adapting Resolution to Finite Satisfiability Checking  
   2.1. Resolution Terminology  
   2.2. Resolution and Finite Satisfiability  
   2.3. Limiting the Functional Height  
   2.4. Compactification  
3. An alternative approach reconsidering the method of Davis and Putnam  
   3.1. Checking for finite satisfiability on the basis of Herbrand's theorem  
   3.2. A generalized Davis-Putnam procedure  
4. Conclusion
1. Introduction

With the advent of deductive (or logic) databases there has been a considerable increase of interest in databases that formally correspond to sets of closed function-free first-order formulas. Conventional relational databases can be considered as sets of closed formulas as well, the facts being represented by positive ground literals and the integrity constraints being non-atomic formulas of some special form. In logic databases [Gallaire 84], however, we have to expect - in addition to explicit facts - a variety of derivation and/or generation rules as well as integrity constraints that are no longer restricted to come from those classes of dependencies that have been proposed in the past.

As a consequence of this development the problem of determining whether a given database is consistent or not has to be reconsidered and new solutions are required. In conventional databases, consistency of the set of integrity constraints is in most cases automatically implied by syntactical properties of the dependency class under consideration: a set of functional dependencies, e.g., will never be inherently contradictory. However, unrestricted types of integrity constraints as well as deduction rules lead to a much more general notion of database consistency.

If constraints and rules are assumed or known to be consistent, the only remaining task for ensuring consistency of the database is to check whether the facts are legal wrt the rules. Investigations of this specialized kind of consistency checking have been made by several authors [Nicolas 79, Blaustein 81, Decker 85].

It is worth noting that consistency of rules has to be verified only when rules are modified, and not when only facts are updated.

This more general consistency of databases is equivalent to consistency of first-order formulas, which is known to be undecidable in general. There are special classes of formulas for which consistency is decidable (called solvable classes), but these are usually characterized by syntactical properties like quantifier constellation or arity of predicates. Such restrictions appear to be unsuitable for database purposes as it is not difficult to imagine meaningful integrity constraints or derivation rules that do not belong to any solvable class.

The converse property - inconsistency - is of course also undecidable, but unlike consistency it is semi-decidable, i.e., algorithms can be built that terminate reporting inconsistency for each input set that is inconsistent. For consistent input these procedures cannot be guaranteed to stop. Algorithms like that have been investigated for more than two decades in the field of automated theorem proving. Most theorem provers that are known today are in fact semi-decision procedures for
inconsistency of sets of formulas. For sure, inconsistency of a database has to be avoided, as answers to queries become completely unreliable if the underlying database is inconsistent. Why can’t we simply take existing theorem provers for checking database consistency?

Unfortunately, it is not sufficient to ensure that a given set of constraints or rules is not inconsistent. Certain consistent sets of formulas are also unacceptable in a database context. In order to characterize these cases it has to be recalled that consistency of a set $S$ of formulas can be defined alternatively in a proof-theoretic and a model-theoretic way:

- there is no formula $F$ such that both $F$ and $\neg F$ are derivable from $S$
- there is an interpretation in which all formulas in $S$ are true (i.e., $S$ has a model)

Both definitions are equivalent due to Goedel’s completeness theorem. If the model-theoretic version is considered, one usually speaks about satisfiability instead of consistency. The models of a satisfiable set of formulas may be either finite or infinite. There are, however, sets that are satisfiable, but don’t have any finite model. The property of a set of formulas to have at least one finite model is called ‘finite satisfiability’. Note that there is no proof-theoretic counterpart for that notion.

For databases, finite satisfiability is of considerable relevance. In conventional databases, any set of explicit facts (i.e., any database state) that is legal wrt a set IC of integrity constraints corresponds to a model of IC. Clearly this model has to be finite, as infinite states cannot be stored. In a deductive context, generation rules that don’t have any finite models are undesirable, because any activation of such a rule leads to an automatic generation of infinitely many new explicit facts. Derivation rules are also generally expected to represent finite sets of implicit facts in order to prevent infinite extensions of answers when derived relations are queried. In summary, it can be claimed that deductive databases - viewed as sets of closed formulas - have to be finitely satisfiable, i.e., when speaking about database consistency we are in fact referring to the finite satisfiability of a first-order theory.

Fig. 1 (on the following page) summarizes how the acceptability of a set $S$ of database constraints/rules is related to the satisfiability of $S$. From this diagram we can immediately conclude why a theorem prover is not sufficient for the task of determining whether $S$ is acceptable or not: if $S$ is not unsatisfiable, then $S$ is not necessarily finitely satisfiable.

---

1A simple example of such an "axiom of infinity" is the formula

$$\forall x \neg p(x,x) \land \exists y p(x,y) \land \forall y \exists z (p(x,y) \land p(y,z) \rightarrow p(x,z)).$$

The predicate '$p$' may be interpreted, e.g., as the $<$-relation over the natural numbers.
As opposed to unrestricted satisfiability, finite satisfiability is also a semi-decidable property [Trachtenbrot 50]. Thus a procedure is constructable that is guaranteed to detect the finite satisfiability of a given set of formulas in finite time, i.e., that stops for any acceptable set.

A "naive" approach to finite model detection - like, e.g., testing for increasing n whether a model with n individuals exists - suffers from the drawback that in case of non-existence of a finite model we run forever without knowing whether this failure is due to unsatisfiability or due to the fact that erroneously an "infinite world" has been axiomatized. Thus, a semi-decision procedure for finite satisfiability that simultaneously semi-decides unsatisfiability would be preferable, as it stops for unsatisfiable input, too, and therefore allows us to distinguish between the two possible reasons for unacceptability.

We show in this report that such a simultaneous procedure exists and investigate in which way it can be obtained. An "evolutionary" approach has been chosen, namely to extend an existing semi-decision procedure for unsatisfiability in order to enable it to detect finite satisfiability as well. This choice has been made because we did not find any reasonable proposal of a semi-decision procedure for finite satisfiability in literature that could serve as a candidate for an extension into a
simultaneous procedure. On the other hand a lot of semi-decision procedures for unsatisfiability have been proposed. A "revolutionary" approach, i.e., a design of a new method from scratch, seems to be unnecessary because most of the techniques required are already provided in existing theorem provers.

Nowadays the most efficient theorem proving procedures described in the standard literature (e.g., [Loveland 78, Chang 73, Wos 84]) are all based on the resolution principle. In the following chapter we investigate how resolution-based procedures can be adapted to finite satisfiability checking. However, the extensions that have to be made in order to guarantee the termination of such a procedure for input sets that have finite models are considerable!

Therefore we have alternatively reconsidered earlier theorem proving methods that do not rely on resolution. For theorem proving purposes these methods are far less efficient than resolution-based ones and have been discarded rather early because of that. However, they offer features that are able to overcome some of the problems that resolution-based methods have when adapted to finite satisfiability checking. In the third chapter we investigate how these early procedures can be extended in order to detect finite satisfiability as well. We describe how one of them - the method of Davis and Putnam - may be generalized in order to overcome some of the most obvious inefficiencies.

In this report we merely give a motivation for and an overview of the two approaches without formally defining them or proving their correctness. Two internal reports [Bry 85a, Manthey 85] are devoted to this point. A comparison of the two approaches wrt efficiency is beyond the scope of this report and will be the subject of forthcoming work.

As far as we know, the only paper that (briefly) mentions the relevance of finite satisfiability for databases is [Fagin 84]. Very few papers have been published til now that address the problem of database consistency in the general form as outlined above. [Kung 85] uses one of the earliest theorem proving procedures - the first-order tableaux method - for detecting inconsistency of database sets of constraints. This method is also used as a refutation method by [Schoenfeld 85]. In [Bry 85b] it has been extended to a procedure for detecting both finite satisfiability and inconsistency. An implementation of a version of this method exists at ECRC [Wallace 85]. However, we have come to the conclusion that compared with more recent theorem proving procedures the tableaux method will hardly be competitive - even after an adaptation to simultaneous detection of finite models. This is due to the same reasons that have caused the inefficiency of early implementations of this method in the theorem proving area.
Terminology and Notation:

A set of formulas (or clauses) is said to be finitely controllable if its satisfiability implies its finite satisfiability [Dreben 79]. If a set of formulas is finitely controllable, then its satisfiability (respectively, inconsistency) can be decided in finite time.

A clause is a finite set of literals. A literal is an atom [Loveland 78, p. 13] or a negated atom. A unit is a clause consisting of one literal. A variant of a clause C is any clause obtained from C by renaming of some or all of its variables. Two literals are complementary if one is an atom and the other the negation of the same atom. The set theoretic notations are used for clauses: if $C_1$ and $C_2$ are clauses, $C_1 \cup C_2$ is a clause, $|C_1|$ is the number of literals in $C_1$, etc... With each first-order formula in conjunctive normal form [Chang 73, p. 13] a (finite) set of clauses (the clausal form of the formula) is associated.

The following notations are used:

\[\text{univ}_S\]

\[\text{fact}_S\]

\[S, S_1, S_2, \ldots\]

\[C, C_1, C_2, \ldots\]

\[L, L_1, L_2, \ldots\]

\[L^c\]

\[t_2 \rightarrow t_1\]

\[C_0\]

\[p, q, r, \ldots\]

\[F, F_1, F_2, G, \ldots\]

\[A, B, C, \ldots\]

\[x, x_1, x_2, y, z, \ldots\]

(In examples we write sets of clauses as two-dimensional matrices where each line corresponds to a clause and each entry to a literal)
2. Adapting Resolution to Finite Satisfiability Checking

Methods based on resolution [Robinson 65] are known to be among the most efficient ones for proving inconsistency of (finite) sets of first-order formulas. Robinson's procedure is therefore a natural candidate for being extended into a method for detecting finite satisfiability. Surprisingly, the properties of resolution with respect to finite satisfiability don't seem to have been studied, although most of the research in automatic theorem proving has been centered on this method since 1965, when it was defined by Robinson. This research was mainly concerned with the definitions of strategies for improving the initial procedure. A method defined like Robinson's original resolution but constrained by a strategy is called a resolution procedure. A procedure is refutation complete if it is always able to detect inconsistency. Robinson's unrestricted resolution is refutation complete, but some strategies are too restrictive and lead to procedures which don't satisfy this property. Nevertheless, all resolution procedures are sound for inconsistency: they never report inconsistency when applied to satisfiable sets of formulas.

Since inconsistency is not a decidable property but only semi-decidable, there are some cases where a resolution method runs forever. In such a case, the input set is satisfiable. Nevertheless, it happens also that a resolution procedure stops after finite time without reporting inconsistency. In this case, the input set is satisfiable, provided the procedure is refutation complete. In other terms, a resolution procedure which is refutation complete is sound for satisfiability. In fact, the following more precise result holds: if a refutation complete resolution procedure stops without reporting inconsistency, then the input set is finitely satisfiable. This property is the basis of the method described in this chapter.

2.1. Resolution Terminology

A set of first-order formulas to be tested for inconsistency by a resolution procedure has to be translated into a special form, called its clausal form. This form is obtained by first putting the formulas in prenex normal form, with matrices in conjunctive form. Existentially quantified variables are replaced by convenient functional terms using new function symbols called Skolem functions. This translation of a set of formulas \( S \) into its clausal form \( C(S) \) is called skolemization. It is described in details in [Loveland 78]. In general skolemization does not preserve logical equivalence (because of the extra function symbols which may be needed), but it preserves satisfiability: a set of formulas \( S \) is satisfiable if and only if its clausal form \( C(S) \) is satisfiable. In fact, it is easy to prove that skolemization also preserves finite satisfiability. It is worth noting that skolemization doesn't produce any nesting of functional symbols (as \( F(x,G(y)) \)) if no function symbols appear in the considered set of formulas.

A classical and convenient way to describe Robinson's resolution is to see it as operating either on a single clause, or on a pair of clauses. When applied on a single clause, it produces, if possible,
a factor of the clause. When applied on a pair of clauses, it produces, if possible, one of their resolvents [Loveland 78]. Resolution is a derivation rule:

If \( F \) is a factor of \( C \), then \( C \models F \).

If \( R \) is a resolvent of \( C_1 \) and \( C_2 \), then \( \{C_1, C_2\} \models R \).

The empty clause is a clause containing no literals. If the empty clause is a resolvent of \( C_1 \) and \( C_2 \), then \( \{C_1, C_2\} \) is inconsistent. Since resolution is a derivation rule, if the empty clause is derivable from a set \( S \) by successive resolution steps, then \( S \) is inconsistent.

Unrestricted resolution may also be viewed as a function \( R_0 \), which, when applied to a finite set \( S \) of clauses, produces another finite set \( R_0(S) \) of clauses defined as the union of \( S \) itself with the set of all factors of clauses in \( S \) and with the set of all resolvents of pairs of members of \( S \). Unrestricted resolution, as a semi-decision procedure for inconsistency may be viewed as successive application of this function \( R_0 \) to the previously built set of clauses. Defining \( R_0^0(S) \) as \( S \) and \( R_0^{n+1}(S) = R_0(R_0^n(S)) \) for all integers \( n \), an application of unrestricted resolution to \( S \) is the construction of the sequence of finite sets \( R_0^0(S), R_0^1(S), \ldots, R_0^n(S), \ldots \). The procedure stops and reports inconsistency when the empty clause is derived. It stops reporting satisfiability when a set \( T = R_0^n(S) \) is reached, such that any clause in \( R_0(T) \) is a variant of a clause in \( T \), i.e. is equal to a clause in \( T \) up to a consistent renaming of variables.

The functional definition of resolution given above is not suitable for all purposes. In particular, it doesn't take into account the order in which resolvents and factors are derived from a given set; it doesn't allow to describe any ordering strategy. Nevertheless, it provides a simple and sufficient framework for restriction strategies, i.e. strategies restricting the set of derived clauses.

A resolution procedure defined from unrestricted resolution \( R_0 \) by a restriction strategy can be expressed as a function \( R \). Applied to a (finite) set \( S \) of clauses, \( R \) produces a subset of \( R_0(S) \). As for \( R_0 \), \( R^0(S) \) and \( R^{n+1}(S) \) are notations for \( S \) and \( R(R^n(S)) \), respectively. A clause \( C \) will be said to be derivable from a set \( S \) by a resolution procedure \( R \) if there exists an integer \( n \) such that \( C \in R^n(S) \).

Soundness with respect to inconsistency of a resolution procedure \( R \) is expressed by:

\[
S \text{ is inconsistent} \iff (\exists n \in \mathbb{N}) \big( \Box \in R^n(S) \big)
\]

A resolution procedure \( R \) is refutation complete if the converse holds:

\[
S \text{ is inconsistent} \implies (\exists n \in \mathbb{N}) \big( \Box \in R^n(S) \big)
\]

A partial order \( \ll \) is defined on the set of all resolution procedures: given two resolution
procedures $R_1$ and $R_2$, $R_1 \ll R_2$ ($R_1$ is more restricted than $R_2$) if for all sets of clauses $S$, $R_1(S) \subseteq R_2(S)$. By definition, all resolution procedures are more restricted than $R_0$.

2.2. Resolution and Finite Satisfiability

The basis of the compactification method is the following result [Bry 85a]:

**Theorem 1:** Let $R$ be a refutation complete resolution procedure. If $R$ stops without reporting inconsistency when applied to a set of clauses $S$, then $S$ is finitely satisfiable.

In other terms, a resolution procedure which is refutation complete is sound for finite satisfiability. Although Theorem 1 is a rather natural result, we were not able to find it in the literature. If a resolution procedure would always stop when applied to finitely satisfiable sets of clauses, it would be complete for finite satisfiability, and therefore a semi-decision procedure for this property. This is not the case for unrestricted resolution $R_0$, as is shown by the following two examples.

**Example 1:** Let $S_1$ be the set of the two clauses 1 and 2 below. $S_1$ is finitely satisfiable since $\{p[a], q[a]\}$ induces a finite model of $S_1$.

\[
\begin{align*}
1 & \quad \neg q(x) \ p(F(x)) \\
2 & \quad q(F(y)) \ \neg p(F(y)) \\
3 & \quad p(F^2(y_1)) \ \neg p(F(y_1)) \quad 1(1)2(1) \ F(y) \rightarrow x \\
4 & \quad q(F^2(y_2)) \ \neg p(F(y_2)) \quad 2(2)3(1) \ F(y_1) \rightarrow y \\
5 & \quad p(F^2(y_3)) \ \neg p(F(y_3)) \quad 1(1)4(1) \ F^2(y_2) \rightarrow x \\
6 & \quad q(F^4(y_4)) \ \neg p(F(y_4)) \quad 2(2)5(1) \ F^2(y_3) \rightarrow y \\
7 & \quad p(F^5(y_5)) \ \neg p(F(y_5)) \quad 1(1)6(1) \ F^4(y_4) \rightarrow x \\
& \quad \vdots \\
& \quad p(F^{n-2}(y_{n-2})) \ \neg p(F(y_{n-2})) \\
& \quad \vdots
\end{align*}
\]

Nested functional terms are expressed by means of power notations: $F^2(1)$ stands for $F(F(1))$, $F^3(1)$ for $F(F(F(1)))$, etc... The notation "3(2)5(3) $x \rightarrow F(y)$" after a clause means that this clause is a resolvent of clauses 3 and 5, where the 2nd literal of clauses 3...
(3(2)) and the 3rd literal of clause 5 (5(3)) are resolved on. The substitution in clause 3 is expressed by the remaining part of the formula.

It can easily be proved that for all integers \( n \geq 3 \) the clause

\[
C_n = \{ p(F^{n-2}(y_{n-2})), \neg p(F(y_{n-2})) \}
\]

is in \( R_0^\infty(S_1) \). Since \( C_n \) is not a variant of any \( C_m \), for \( 1 \leq m \leq n - 1 \), \( R_0 \) never stops when applied on \( S_1 \). Note that for all \( n \geq 3 \), \( |C_n| = 2 \).

**Example 2:** Let \( S_2 \) be the set containing clauses 1 and 2 below.

1. \( p(x_1, x_2), q(x_2, x_3), \neg r(x_3, x_4) \)
2. \( \neg p(y_1, y_3), q(y_2, y_3), r(y_2, y_4) \)

\[
\downarrow
\]

3. \( q(z_2, z_3), q(z_4, z_5), \neg r(z_4, z_5), r(z_4, z_1) \)

\[
\downarrow
\]

4. \( p(x_1, x_2), q(x_2, x_3), q(x_3, x_4), \neg r(x_4, x_1) \)

\[
\downarrow
\]

\[
\vdots
\]

\[
n \ p(x_1, x_2), q(x_2, x_3), \ldots, q(x_{n-1}, x_n), \neg r(x_n, x_1)
\]

It can be proved that for all integers \( n \geq 3 \) the clause

\[
C_n = \{ p(x_1, x_2), q(x_2, x_3), \ldots, q(x_{n-1}, x_n), \neg r(x_n, x_1) \}
\]

is in \( R_0^\infty(S_2) \). No \( C_n \) is a variant of any \( C_m \), for \( 3 \leq m \leq n - 1 \). Therefore, \( R_0 \) doesn’t stop when applied on \( S_2 \), while this set is clearly finitely satistifiable.

The clauses in \( S_1 \) are associated with formulas belonging to the monadic class (i.e. formulas with predicates of arity exactly one). The clauses in \( S_2 \) are associated with formulas in the Bernays-Schoenfinkel class (i.e. formulas in prenex normal form with prefix \( \exists \forall \)). These two classes are known to be finitely controllable and are usually considered as the simplest examples of finitely controllable classes. It is worth noting that \( R_0 \) is neither a decision procedure for satisfiability of monadic formulas nor of formulas of the Bernays-Schoenfinkel class. This point has been noticed by Joyner who has given some examples similar to Examples 1 and 2 [Joyner 76].

All clauses derived from a finite set of clauses \( S \) are constructed from a finite number of constants, predicates, and functions symbols: those occurring in \( S \). Therefore, if a resolution procedure runs forever when applied to a finite set, it generates clauses with unbounded nesting of functional terms, as in Example 1, or with unbounded number of variables, as in Example 2. Given
a clause $C$, let $f(C)$, the functional height of $C$, denote the height in nesting of functional terms occurring in $C$. Similarly, let $v(C)$, the variable height of $C$, denote the number of variables occurring in $C$. If $S$ is a set of clauses, $v(S)$ denotes $\max\{v(C) \mid C \in S\}$. A method for checking for finite satisfiability has to control both of these two parameters. When applied to a finitely satisfiable set of clauses $S$, $f(C)$ and $v(C)$ have to be limited for all clauses derived from $S$. When applied to an axiom of infinity, clauses with unlimited functional or variable height have to be derivable. There exists a resolution based method which limits the functional height, on which the compactification method is based.

2.3. Limiting the Functional Height

In [Joyner 76], a resolution procedure called $R_3$ is defined. It is proved to be refutation complete and not to produce any nesting of functional terms when applied to a formula in Skolem normal form, i.e. in prenex normal form with prefix $\forall^\exists^\ast$. $R_3$ doesn’t seem to be practical for an implementation, but from the existence of such a procedure it follows that the resolution procedure $R_1$ defined from $R_0$ by simply discarding all resolvents and factors $C$ such that $f(C) > 1$ is refutation complete on the prefix class $\forall^\exists^\ast$. A translation process is known, which associates to all formulas $F$ a formula $S(F)$ in Skolem normal form such that $F$ is satisfiable if and only if $S(F)$ is satisfiable [Mendelson 79, Ex. 2.79 p. 92]. In fact, it is possible to prove that even the following property holds: $F$ is finitely satisfiable if and only if $S(F)$ is finitely satisfiable [Bry 85a]. Applying $R_1$ to $S(F)$ instead of $F$ leads to a procedure which doesn’t increase the functional height and is refutation complete for all first-order formulas.

Given a formula $F$ in prenex normal form, but not in Skolem normal form, $S(F)$ is formed by considering the leftmost occurrence of an existentially quantified variable followed by a universally quantified variable in the prefix of $F$. From a formula:

$$\forall x_1 \ldots \forall x_n \exists y Q_1 z_1 \ldots Q_m z_m M(x_1, \ldots, x_n, y, z_1, \ldots, z_m)$$

a first formula is constructed:

$$\forall x_1 \ldots \forall x_n \forall y Q_1 z_1 \ldots Q_m z_m \exists w [M(x_1, \ldots, x_n, y, z_1, \ldots, z_m) \lor p(x_1, \ldots, x_n, y)] \land \neg p(x_1, \ldots, x_n, w)$$

where $p$ is a new predicate ($p$ doesn’t occur in the initial formula $F$). If there is an existentially quantified variable followed by a universally quantified one in the sequence $Q_1 z_1 \ldots Q_m z_m$, then this process is repeated. The number of extra literals added to the initial formula during this translation process depends linearly on the number of existentially quantified variables occurring in the initial formula.
Example 3: Consider the following formula \( F \):
\[
\exists y_1 \forall x \exists y_2 \forall z \ [m(y_1, x) \implies (n(x, y_2) \land n(y_2, z))]
\]
The clausal form of \( F \) is:
\[
\neg m(F_1, x) \land n(x, F_2(x))
\]
\[
\neg m(F_1, x) \land n(F_2(x), z)
\]
The Skolem normal form \( S(F) \) of \( F \) is constructed in two steps as follows:

1. \( \forall y_1 \forall x \exists y_2 \forall z \exists w_1 \ [(m(y_1, x) \implies (n(x, y_2) \land n(y_2, z))) \lor p_1(y_1)] \land \neg p_1(w_1) \)
2. \( \forall y_1 \forall x \forall y_2 \forall z \exists w_1 \exists w_2 \ [(m(y_1, x) \implies (n(x, y_2) \land n(y_2, z))) \lor p_1(y_1)] \land \neg p_1(w_1) \lor p_2(y_1, x, y_2) \land \neg p_2(y_1, x, w_2) \)
The clausal form of \( S(F) \) is:
\[
\neg m(y_1, x) \land n(x, y_2) \land p_1(y_1) \land p_2(y_1, x, y_2)
\]
\[
\neg m(y_1, x) \land n(y_2, z) \land p_1(y_1) \land p_2(y_1, x, y_2)
\]
\[
\neg p_1(G_1(y_1, x, y_2)) \land p_2(y_1, x, y_2)
\]
\[
\neg p_2(y_1, x, G_2(y_1, x, y_2))
\]

Functional nesting is simply avoided by applying the resolution procedure \( R_1 \) to the set of clauses induced by the Skolem normal form of the initial set of formulas (the Skolem normal form of a set of formulas is the Skolem normal form of the conjunction of its elements).

2.4. Compactification

Let \( F = \forall x_1 \ldots \forall x_{n+1} \ M(x_1, \ldots, x_{n+1}) \) be a formula in which \( n+1 \) (distinct) universally quantified variables occur. All formulas formed from \( F \) by identifying some \( x_i \)'s are logical consequences of \( F \). For example:
\[
F = \forall x_1 \ldots \forall x_n \ M(x_1, \ldots, x_n, x_n)
\]
If \( F \) is satisfied in a model \( M \) with a universe of cardinality \( n \), then the evaluation of \( F \) in \( M \) has to identify two of the \( x_i \)'s. In other terms, under the assumption that \( F \) has a model with a universe of cardinality \( n \), \( F \) is implied by the set of all 'compactified' formulas formed from \( F \) by identifying two of the \( x_i \)'s. Formally, given a clause \( C \) in which \( m > n \) variables occur, the compactification set at height \( n \) of \( C \), \( \text{comp}_n(C) \), is the set of all clauses obtained from \( C \) by identifying \( m - n - 1 \) variables in \( C \). For example, if \( C = \{ p(x), q(x, y), r(z) \} \), then \( \text{comp}_2(C) \) contains the following three clauses:
\[
p(x) \ q(x, x) \ r(z) \quad \text{x and y are identified}
\]
\[
p(x) \ q(x, y) \ r(x) \quad \text{x and z are identified}
\]
\[
p(x) \ q(x, y) \ r(y) \quad \text{y and z are identified}
\]
If at most \( n \) variables occur in a clause \( C \), \( \text{comp}_n(C) \) is equal to \( \{C\} \). If \( S \) is a set of clauses, the compactified of \( S \) at height \( n \) is the set \( \text{comp}_n(S) = \cup \{ \text{comp}_n(C) \mid C \in S \} \).

Let \( S \) be the set of clauses associated with the Skolem normal form of the initial formula. The compactification method performs the following steps:

1. Initialization: \( T \leftarrow S \). Go to 2.

2. Built \( R_1(T) \).
   If \( \square \in R_1(T) \), then stop reporting inconsistency.
   Else, if \( R_1(T) = T \), then stop reporting finite satisfiability.
   Otherwise, go to 3.

3. Test the existence of a model of \( R_1(T) \) with cardinality \( v(T) \), as described below.
   If such a model exists, then stop reporting finite satisfiability.
   Otherwise, \( T \leftarrow R_1(T) \) and go to 2.

Given a set \( S \) of clauses, the existence of a model of \( S \) with cardinality \( n \) is tested as follows:

1. Initialization: \( T \leftarrow \text{comp}_n(S) \).

2. Built \( R_1(T) \).
   If \( \square \in R_1(T) \), then stop reporting failure: \( T \) doesn’t have any model with cardinality at most \( n \).
   If \( \text{comp}_n(R_1(T)) = T \), then go to 3.
   Otherwise, \( T \leftarrow \text{comp}_n(R_1(T)) \) and go to 2.

3. If \( R_1(T) \) doesn’t contain any functional terms, stop with success.
   Otherwise, variables are instantiated with \( n \) constants and functional terms are evaluated.
   Each possible evaluation (i.e. resulting set of ground terms) is tested for consistency.
   Stop reporting success when a consistent evaluation is found.
   If there is no consistent evaluation, stop reporting failure.

Since \( R_1 \) doesn’t produce nested functional terms, and since each resolvent is compactified at height \( n \), step 2 of the test for the existence of a finite model cannot enter an infinite loop: a set \( T \) such that \( \text{comp}_n(R_1(T)) = T \), i.e. such that \( v(R_1(T)) = v(T) \), is necessarily reached.

Examples 4 and 5 below give an idea of the method. In example 4, the variable height never increases. Therefore, no test of finite cardinality is performed.

**Example 4:** \( C \) is the set containing the following clauses:

\[
q(x) \quad r(x) \\
\neg q(x) \quad s(x) \\
\neg r(A) \\
\neg r(F(x)) \quad s(F(x))
\]
The successive steps are:

1. $R_1^1(C)$ contains the new clauses:
   
   \[ r(x) \quad s(x) \\
   q(A) \\
   q(F(x)) \quad s(F(x)) \]

2. $R_1^2(C)$ contains the new clauses:
   
   \[ s(A) \quad s(F(x)) \]

No more steps are performed, since no new clauses are derivable from $R_1^2(C)$. Therefore, $C$ is finitely satisfiable.

The set of clauses of the next example is the Skolem normal form of the following formula:

\[ \forall x \exists u \forall y \left( p(u,x) \land \neg p(x,y) \land [p(x,y) = \exists u(p(u,y))] \right) \]  
This formula is an axiom of infinity: it is satisfiable, but it doesn’t have any finite models, therefore, the method never stops when applied to this formula.

Example 5: $C$ contains clauses

\[ p(u,x) \quad q(x,u) \]
\[ \neg p(x,x) \quad q(x,u) \]
\[ \neg p(x,y) \quad p(u,y) \quad q(x,u) \]
\[ \neg q(x,F(x,y)) \]

1. $R_1^1(C)$ and $R_1^2(C)$ don’t increase the variable height.

2. $R_1^2(C)$ contains the clause $\{p(x_1,x_2), q(x_2,x_3), q(x_3,x_1), q(x_3,x_4)\}$. This clause is ‘compactified’ into the following six clauses:

\[ p(x_1,x_1) \quad q(x_1,x_2) \quad q(x_2,x_3) \]
\[ p(x_1,x_2) \quad q(x_2,x_1) \quad q(x_2,x_3) \]
\[ p(x_1,x_2) \quad q(x_2,x_3) \quad q(x_3,x_1) \quad q(x_3,x_4) \]
\[ p(x_1,x_2) \quad q(x_2,x_3) \quad q(x_3,x_1) \quad q(x_2,x_4) \]
\[ p(x_1,x_2) \quad q(x_2,x_3) \quad q(x_3,x_1) \quad q(x_3,x_2) \]
\[ p(x_1,x_2) \quad q(x_2,x_3) \quad q(x_3,x_1) \quad q(x_2,x_3) \]

The empty clause is derived from this ‘compactified’ set of clauses, proving that the initial set of clauses doesn’t have any model of cardinality 5. Therefore, the cardinality 4 has to be tested. The test for the existence of a model with cardinality 4 will also fail, and so on...

The drawback of this approach is the explosion of the number of clauses: the ‘compactification’ at variable level $n$ of a clause containing $n - 1$ variables produces $\frac{n(n-1)}{2}$ distinct clauses. We have to
mention that, very often, many of these clauses are either tautologies, or subsumed by other clauses. This explosion is the 'price to pay' for delaying as much as possible the instantiation phase. In some cases, as in Example 4, it is fruitful.

This method is based on a particular resolution procedure $R_1$. Since there exists some axiom of infinity (as that given in Example 5) which don't have any resolvents increasing the variable height, $R_1$ cannot be replaced by an arbitrarily chosen refutation complete resolution procedure. A resolution procedure more restrictive than $R_1$ and less restrictive than Joyner's $R_3$ would be convenient, but no such procedures are known.
3. An alternative approach reconsidering the method of Davis and Putnam

The second approach we propose for adapting a theorem proving method for semi-decision of finite satisfiability is based on a procedure suggested in [Davis-Putnam 60]. It belongs to a class of similar procedures that can be considered as direct implementations of Herbrand’s theorem: to test whether a set $S$ of clauses is unsatisfiable, it is sufficient to consider only finite sets of ground instances of $S$ that are obtained by instantiation of variables over a special domain, called the Herbrand universe of $S$. As unsatisfiability is decidable for finite sets of ground clauses, an unsatisfiable set of ground instances can be effectively detected - provided the respective instantiations are performed in a systematic way. The Herbrand universe $H_S$ of $S$ can be recursively defined:

$$
H_S^0 := \begin{cases} 
\{ c \} & \text{if } \text{univ}_S = 0 \\
\{ c' \in \text{univ}_S \mid c' \text{ is a constant} \} & \text{else}
\end{cases}
$$

$$
H_S^i := H_S^{i-1} \cup \{ \{t_1, \ldots, t_n\} \mid f \in \text{fct}_S \text{ and } t_1, \ldots, t_n \in H_S^{i-1} \} \quad (i > 0)
$$

$$
H_S := \cup \{ H_S^i \mid i \in \mathbb{N} \}
$$

This recursive definition of $H_S$ induces a procedure that successively generates sets

$$
S_0 \subseteq S_1 \subseteq \ldots \subseteq S_i \subseteq \ldots
$$

where $S_i$ is the set of all ground instances of clauses in $S$ over $H_S^i$. Each of the $S_i$ is tested for unsatisfiability directly after its generation. According to Herbrand’s theorem, $S$ is unsatisfiable iff one of the $S_i$ is unsatisfiable. The various theorem proving methods that are based on this paradigm differ mainly in the way they check for unsatisfiability of the $S_i$.

If on the other hand $S$ is satisfiable, then each of the $S_i$ will be satisfiable, too. As the procedure searches for an unsatisfiable $S_i$; however, it will run forever, except in those (few) cases where $H_S$ is finite and thus all the $S_i$ are identical. This happens only if $S$ contains no function symbols, i.e., $S$ is the clausal representation of a formula in the (solvable) Bernays-Schoenfinkel class. Finite satisfiability cannot be detected in the remaining cases!
3.1. Checking for finite satisfiability on the basis of Herbrand's theorem

The key to an adaption of methods based on Herbrand's theorem to finite model determination is provided by the following theorem given in [Manthey 85]:

**Theorem 2**: \( S \) is finitely satisfiable iff there is an \( i \geq 0 \) such that \( S_i \) is satisfiable in domain \( H_S^1 \).

Let \( G \) denote a set of ground clauses and \( D \) a subset of \( \text{univ}_G \). We call \( G \) satisfiable in domain \( D \) iff there is a substitution \( \sigma \) replacing each term in \( \text{univ}_G \) by a term in \( D \) such that:

1. \( f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)\sigma \) for each functional term \( f(t_1, \ldots, t_n) \)
2. \( D\sigma = D \)
3. \( G\sigma \) is satisfiable

In order to determine whether some \( S_i \) is satisfiable in domain \( H_S^1 \), it is necessary to find a substitution that replaces each term in \( \text{univ}_{S_i} \) by a term in \( H_S^1 \). It is even possible to restrict oneself to replacing only the terms in \( \text{univ}_{S_i} \setminus H_S^1 \), because \( H_S^1 \subseteq \text{univ}_{S_i} \subseteq H_S^{i+1} \). We call such a substitution a function evaluation over \( H_S^1 \). As \( H_S^{i+1} : H_S^1 \) is finite, a function evaluation over \( H_S^1 \) can be effectively determined by analysis of at most \( |H_S^{i+1} : H_S^1| \) cases.

In the same way as semi-decision procedures for unsatisfiability rely on Herbrand's theorem, semi-decision procedures for finite satisfiability that operate on the instance sets \( S_i \) of \( S \) are based on Theorem 2. In order to obtain a simultaneous semi-decision procedure for both properties, each \( S_i \) has to be tested for both, (propositional) unsatisfiability as well as satisfiability over domain \( H_S^1 \), according to:

**Theorem 3**:

a) \( S \) is unsatisfiable iff \( S_i \) is unsatisfiable for some \( i \geq 0 \) (Herbrand's theorem)

b) \( S \) is finitely satisfiable iff \( S_i \) is satisfiable in domain \( H_S^1 \) for some \( i \geq 0 \)

c) \( S \) is satisfiable, but not finitely satisfiable iff \( S_i \) is satisfiable, but unsatisfiable in domain \( H_S^1 \) for all \( i \geq 0 \).

Fig. 1a and Fig. 1b illustrate the two possible ways in which both tests can be sequentially coupled depending on which one is performed first. Although the two couplings seem to be symmetrical to each other, there is a considerable difference in efficiency due to the way in which the subsequent tests can be coupled. Whenever one of the tests for (propositional) unsatisfiability mentioned above stops, it has reached a satisfiable set \( S_i^* \) that logically implies \( S_i \). Thus a subsequent test for satisfiability over \( H_S^1 \) - like in case a) - can directly be applied to \( S_i^* \) instead of \( S_i \), i.e., the result of the first test can directly be used by the second. As \( S_i^* \) is considerably smaller
than $S_i$; this means a gain in efficiency as compared with an application of the second test to $S_i$. On the other hand, if a test for satisfiability of $S_i$ over $H_S$ stops it has reached a set $S_i^{**}$ that has been constructed from $S_i$ by means of function evaluations. Although the satisfiability of $S_i$ over $H_S$ can be concluded from that of $S_i^{**}$, $S_i^{**}$ does not logically imply $S_i$. Thus a subsequent detection of unsatisfiability of $S_i^{**}$ does not imply the unsatisfiability of $S_i$. Therefore, in case b) the second test also has to be applied to $S_i$ without being able to make use of the work done during the first test. Consequently a procedure that checks the (propositional) unsatisfiability of $S_i$ first - like in case a) - has to be preferred.

a) test for (propositional) unsatisfiability first:

- Fig. 2a -

- runs forever iff $S$ neither finitely satisfiable nor unsatisfiable

S satisfies over $H_S$ (stop)

S satisfies over $H_S$ (backtrack 1)

S finitely satisfiable (stop)
b) test for 'satisfiability over domain' first:

![Diagram](image)

- Fig. 2b -
3.2. A generalized Davis-Putnam procedure

As mentioned in the introduction, methods like the Davis-Putnam procedure have been discarded rather early for theorem-proving purposes because of their inefficiency. This is due to the fact that they have to perform a vast amount of instantiations in order to reach a level \( \text{level } i \) where \( S_i \) becomes unsatisfiable. We also mentioned that we have reconsidered these methods only because they offer a somehow natural solution to the problem of unlimited growth of clause length. The enormous advantage of resolution, however, namely to be able to do without (explicit) instantiation, still has considerable importance - even in a context where finite satisfiability is more important than unsatisfiability. It would be desirable to overcome the drawback of complete instantiation at least partially by adding new rules to the Davis-Putnam procedure that are no longer restricted to ground clauses.

We have made a first attempt in this direction by defining generalizations of the rules that are already present in this procedure. These generalized rules do not completely avoid the need for explicit instantiation, but in many cases finite satisfiability can be detected without instantiating at all. Moreover, the proposed generalizations are compatible with the function evaluation paradigm introduced above.

The procedure of Davis and Putnam mainly relies on three rules for the creation of new sets of clauses from a given set \( S \) and for the introduction of new subcases:

1. deletion of subsumed ground clauses:
   If \( S \) contains a ground clause \( C \) as well as a proper subclause of \( C \),
   then \( C \) can be deleted from \( S \).
   [This rule is based on the propositional tautology \( A \lor (A \land B) \leftrightarrow A \).]

2. ground unit resolution:
   If \( S \) contains a ground unit \( \{L\} \) as well as a ground clause \( C \) containing \( L^c \),
   then \( C \) can be replaced by \( C \setminus \{L^c\} \).
   [This rule is based on the tautology \( A \land \{A^c \lor B\} \leftrightarrow A \land B \).]

3. ground unit introduction:
   If a non-unit ground clause \( C \) in \( S \) contains a literal \( L \), then two new subcases are created:
   - In the one case the unit \( \{L\} \) is added to \( S \) and \( C \) is deleted (because it is subsumed by the new unit).
   - In the second case the unit \( \{L^c\} \) is added and \( C \) is replaced by \( C \setminus \{L\} \) (which results from resolving \( C \) against the new unit).
   [This rule is based on another tautology \( A \lor B \leftrightarrow A \lor (A^c \land B) \).]
The following example gives an impression of the way in which the Davis-Putnam procedure with function evaluation determines finite satisfiability:

Example 6:  

\[ S: \]

\[
\begin{align*}
q(X) & \quad r(x) \\
\neg q(X) & \quad s(x) \\
\neg r(A) & \\
s(Fx) & \quad \neg r(Fx)
\end{align*}
\]

\[ H_S^0 = \{A\} = \text{univ}_S \]
\[ H_S^1 = \{A, FA\} \quad H_S^1 \setminus H_S^0 = \{FA\} \]
\[ H_S^2 = \{A, FA, FFA\} \]

\[ S_0: \]

\[
\begin{align*}
q(A) & \quad r(A) \\
\neg q(A) & \quad s(A) \\
\neg r(A) & \\
s(FA) & \quad \neg r(FA)
\end{align*}
\]

\[ \text{ground unit resolution} \]

\[
\begin{align*}
q(A) & \\
\neg q(A) & \quad s(A) \\
\neg r(A) & \\
s(FA) & \quad \neg r(FA)
\end{align*}
\]

\[ \text{ground unit resolution} \]

\[
\begin{align*}
q(A) & \\
s(A) & \\
\neg r(A) & \\
s(FA) & \quad \neg r(FA)
\end{align*}
\]

\[ \text{ground unit introduction} \]

\[ S_0 \text{ is finitely satisfiable} \]

\[ \Rightarrow \quad S_0, \text{satisfiable in } H_S^0 \]
A generalization of rules 1 and 2 is quite obvious: deletion of subsumed (general) clauses as well as (general) unit resolution are well-known standard features in theorem proving.

Our generalization of the ground unit introduction rule, however, seems to be new. It is mainly due to the way we have defined rule 3 - which is different from the original definition of Davis/Putnam but equivalent, if combined with rules 1 and 2. It is possible to consider the first alternative case - introduction of a new unit \{L\} - as a kind of "test of a hypothesis": we assume that there is a model of S where \{L\} is true and test the reduced set \(S^a = (S \setminus \{C\}) \cup \{\{L\}\}\) for satisfiability. If \(S^a\) is satisfiable, we can stop, because satisfiability of \(S^a\) implies satisfiability of S. If \(S^a\) is unsatisfiable, we know that no model of S exists where \{L\} is true. Consequently \{L^c\} has to be true in every model and we can transform S into \(S^b = (S \setminus \{C\}) \cup \{\{L^c\},(C \setminus \{L\})\}\) which is smaller than S, too.

Of course we can as well take a non-ground literal \(L\) as a "unit candidate" in a similar way. If the "assumption" \(\{L\}\) fails, however, we cannot simply conclude that \(\{L^c\}\) is true in every model of S, because in this case the literal \(L\) contains at least one variable \(x\) which has to be seen as implicitly universally quantified. The formula \((\forall x[L] \lor \forall x[L^c])\), however, is not a tautology, whereas for ground literals \((L \lor L^c)\) is tautologous. If none of the non-ground "unit candidates" in S leads to a satisfiable set, we have to manage with the remaining rules - which possibly means that we have to instantiate, if neither subsumption nor unit resolution are applicable, too. In this case "the bill has to be paid" for the benefit that the generalized unit introduction rule gives us for rapid detection of finite satisfiability.

However, it is not possible to do without this generalized rule, because the applicability of general unit resolution depends on it. If only unit resolution would have been generalized to the non-ground case, the procedure would run forever even for some finitely satisfiable input sets. Unit resolution is a "critical" feature wrt unlimited production of new clauses and therefore has to be controlled by carefully ordering the application of the different rules. Our method applies them according to the following list of priorities:

1. subsumption (i.e., immediately; as soon as possible)
2. function evaluation
3. unit introduction
4. unit resolution
5. instantiation (i.e., only if absolutely unavoidable)

The method briefly described here is formally defined and justified in [Manthey 85].
At the end of this chapter we give two more examples: Expl. 7 shows how the set S from Expl. 6 is treated by our generalized method; Expl. 8 shows how a variant of the set from Expl. 1 is checked for finite satisfiability by the same method.

**Example 7:**  (each set is labelled with the set of unit candidates that remain to be tested)

\[
S:
\begin{align*}
q(x) & \quad r(x) \\
q(x) & \quad s(x) \\
\neg r(A) & \\
s(Fx) & \quad \neg r(Fx)
\end{align*}
\]

\(r(x)\) is no unit candidate because it is complementary to the unit literal \(\neg r(A)\)

\[\{q(x), \neg q(x), s(x), s(Fx), \neg r(Fx)\}\]

unit introduction

\[S^a:
\begin{align*}
q(x) \\
\neg q(x) & \quad s(x) \\
\neg r(A) & \\
s(Fx) & \quad \neg r(Fx)
\end{align*}
\]

\[\{s(x), s(Fx), \neg r(Fx)\}\]

unit introduction

\(\neg\)subsumption

\[S^{aa}:
\begin{align*}
s(x) \\
q(x) \\
\neg r(A)
\end{align*}
\]

\[\{} \]

closed under rules 1 - 5

\[S \text{ finitely satisfiable} \]

\[S^b:
\begin{align*}
q(x) & \quad r(x) \\
\neg r(A) & \\
s(Fx) & \quad \neg r(Fx)
\end{align*}
\]

\[\{q(x), \neg r(Fx)\}\]

identical to \(S^a\), but one unit candidate less

\[S^{ab}:
\begin{align*}
s(x) \\
q(x) \\
\neg r(A)
\end{align*}
\]

\[\} \]

identical to \(S^a\), but one unit candidate less:

\[\{s(Fx), \neg r(Fx)\}\]

no need to test these alternatives, as a solution has been found on the leftmost branch
Example 8:

S:

\[
\begin{align*}
&\ p(x) \ p(Fx) \\
&\ \neg p(x) \ \neg p(Fx) \\
&\ p(A) \ p(FA) \\
&\ \neg p(A) \ \neg p(FA)
\end{align*}
\]

\{-\}

no unit candidates because each literal in S is "complementary" to a whole clause

\[
\{p(A), \neg p(A)\}
\]

unit resolution

\[
\begin{align*}
&\ \vdash \\
&\ p(A) \\
&\ \neg p(A) \ \neg p(FA)
\end{align*}
\]

\{-\}

function evaluation \[A \rightarrow FA\]

\[
\begin{align*}
&\ \vdash \\
&\ \neg p(FA) \\
&\ p(A)
\end{align*}
\]

\{-\}

unit resolution

function evaluation \[A \rightarrow F^2A\]

\[
\begin{align*}
&\ \vdash \\
&\ \neg p(FA) \\
&\ p(A) \\
&\ p(F^2A)
\end{align*}
\]

\[
\begin{align*}
&\ \vdash \\
&\ \neg p(FA) \\
&\ p(A)
\end{align*}
\]

function evaluation \[FA \rightarrow F^2A\]

\[
\begin{align*}
&\ \vdash \\
&\ \neg p(FA) \\
&\ p(A)
\end{align*}
\]

closed under rules 1-5

\[\Rightarrow \ S \text{ finitely satisfiable}\]
4. Conclusion

In this report we have investigated alternative approaches to extend an algorithm for theorem proving into a simultaneous semi-decision procedure for unsatisfiability and finite satisfiability of a set of formulas. We consider this kind of procedure as being more appropriate for database consistency checking than theorem provers, because finite satisfiability reflects more precisely than consistency which kind of formulas are acceptable as integrity constraints or deductive rules.

Two alternative approaches have been presented: the one based on the resolution principle, the other reconsidering an earlier theorem proving technique. Both have their advantages as well as their drawbacks. The resolution-based method controls growth in nesting of functional terms rather easily whereas the features needed for limitation of growth in clause length are expensive. The alternative method is not able to lengthen clauses at all as it relies on a stepwise reduction of clause length, but this has to be paid by the requirement of an extensive case analysis. However, it is impossible to simply combine the advantages of the two methods because solving simultaneously both problems - growth in function nesting as well as growth in clause length - necessarily results in a loss of completeness for unsatisfiability.

The next phase of our work will be mainly devoted to an evaluation of the methods relative to each other. This will be done with the support of a rapid and simple PROLOG prototype for both procedures which will enable us to master bigger examples that cannot be solved by hand anymore. Of course such a first prototype will not give us any result concerning the absolute efficiency and practicability of the proposed methods, but merely serves as an aid in deciding if one of the approaches is more suitable for a serious implementation than the other.

Apart from that we will continue to look for improvements of the different approaches as well as for possibilities of combining at least partially useful features from both methods in order to enhance their overall efficiency.
References

[Blaustein 81] Blaustein, B. T.
Enforcing Database Assertions: Techniques and Applications.

[Bry 85a] Bry F.
The Compactification Method.

[Bry 85b] Bry, F.
Note on Consistency Checking of Databases Schemas.

[Chang 73] Chang, C.-L. and Lee R. C.-T.
Symbolic Logic and Mechanical Theorem Proving.

[Davis-Putnam 60] Davis, M. and Putnam, H.
A Computing Procedure for Quantification Theory.

[Decker 85] Decker H.
Expression and Enforcement of Integrity Constraints in Prolog KB, Version 0.

[Dreben 79] Dreben, B. and Goldfarb, W.
The Decision Problem. Solvable Classes of Quantificational Formulas.
Addison-Wesley, Reading, Massachusetts, 01867, 1979.

[Fagin 84] Fagin, R. and Vardi M. Y.
The Theory of Data Dependencies - An Overview.

[Gallaire 84] Gallaire, H., Minker J. and Nicolas, J.-M.
Logic and Databases: A Deductive Approach.

[Joyner 73] Joyner, W. H.
Automatic Theorem-Proving and the Decision Problem.
(Ph.D. thesis).

[Joyner 76] Joyner, W. H.
Resolution Strategies as Decision Procedure.

A Tableaux Approach for Consistency Checking.
In Working Conference on Theoretical and Formal Aspects of Information Systems.
Sitges, Spain.
[Loveland 78] Loveland, D. W.
*Automated Theorem Proving: a Logical Basis.*

[Manthey 85] Manthey, R.
*A Generalized Davis-Putnam Procedure Able to Detect Finite Satisfiability.*
[in preparation].

[Mendelson 79] Mendelson, E.
*Introduction to Mathematical Logic.*

[Nicolas 79] Nicolas, J.-M.
*Logic for Improving Integrity Checking in Relational Databases.*
Also in Acta Inf. 18, 3 (Dec), pp. 227-253.

[Robinson 65] Robinson, J. A.
A Machine-Oriented Logic Based on the Resolution Principle.

[Schoenfeld 85] Schoenfeld, W.
Prolog Extensions Based on Tableau Calculus.
In *Proceedings of the 9th International Joint Conference on Artificial Intelligence.*
Los Angeles.

[Trachtenbrot 50] Trachtenbrot, B. A.
Impossibility of an Algorithm for the Decision Problem in Finite Classes.

[Wallace 85] Wallace, M.
*Consistency Report.*

[Wos 84] Wos, L., Overbeek, R., Lusk, E. and Boyle, J.
*Automated Reasoning, Introduction and Applications.*