Checking Consistency of Database Constraints: a Logical Basis

Francois BRY, Rainer MANTHEY

ECRC, Arabellastr. 17, 8000 Muenchen 81, West Germany

Abstract

This paper addresses the problem of consistency of a set of integrity constraints itself, independent from any state. It is pointed out that database constraints have not only to be consistent, but in addition to be finitely satisfiable. This stronger property reflects that the constraints have to admit a finite set of (stored as well as derivable) facts. As opposed to consistency, being undecidable, finite satisfiability is semi-decidable. For efficiency purposes we investigate methods that check both finite satisfiability as well as unsatisfiability. Two different methods are proposed which extend two alternative approaches to refutation.

1. Introduction

In a database context, a lot of work has been done on integrity enforcement, i.e., on checking the validity of a database state with respect to a given set of integrity constraints. The question whether the constraint set itself is consistent has till now received quite few attention, although the problem is fundamental ([12] constitutes a notable exception). Usually constraints are either tacitly assumed to be consistent, or the use of a theorem prover is suggested in order to detect inconsistencies. If constraints are restricted to come from classes like functional, multi-valued or implicational dependencies, consistency is already implied by the syntactical properties of the respective classes. However, as pointed out by many authors, more general kinds of constraints have to be admitted, and therefore the problem has to be addressed on a more general basis. As the formalisms of relational databases and predicate logic are so closely related, we will consider constraints as arbitrary closed and function-free first order formulas.

Consistency is a necessary wellformedness condition for constraint sets (as opposed to, e.g., non-redundancy which is a desirable, but not an indispensable requirement). An inconsistent set of constraints does not admit any valid database state. In terms of logic, database states can be considered as interpretations of the constraints. Valid states correspond to interpretations in which every constraint is true, i.e., to models of the constraint set. Inconsistent sets of formulas do not have any model - they are unsatisfiable. (The model-theoretic property 'satisfiability' is equivalent to the proof-theoretic property 'consistency' according to Goedel's Completeness Theorem).

As database states correspond to models of the constraints, it is not sufficient to guarantee the existence of any model in general, but finite models have to exist in particular. In conventional databases, constraints have to admit finite models as every state consists of a finite number of facts. In definite deductive databases (as defined in [9]) the set of deduction rules always has a finite minimal model, which is intended to be a model of the constraint set as well. Satisfiability does not necessarily imply *finite satisfiability*, i.e., the existence of a finite model. There are satisfiable sets of formulas - called 'axioms of infinity' - that have only infinite models. Consider, e.g., a set of integrity constraints for a managerial database containing (among others) the following constraints:

- Everybody works for somebody.
- Nobody works for himself.
- If x works for y and y works for z, then x works for z.

Expressed as first-order formulas, these three constraints correspond to a well-known axiom of infinity. Although each of them appears to be reasonable as such, an infinite number of individuals is required in any model of the set as a whole. This defect could be avoided by providing the first constraint with a proviso like, e.g., "everybody except the top-manager..". Much more complex axioms of infinity may be hidden inside a large and intricate set of constraints which cannot be so easily identified as in the example above. Therefore, in addition to preventing constraints from being unsatisfiable, axioms of infinity have to be avoided as well. Constraints have to be finitely satisfiable, as already briefly mentioned in [8].

Figure 1 illustrates how the three properties mentioned are related.

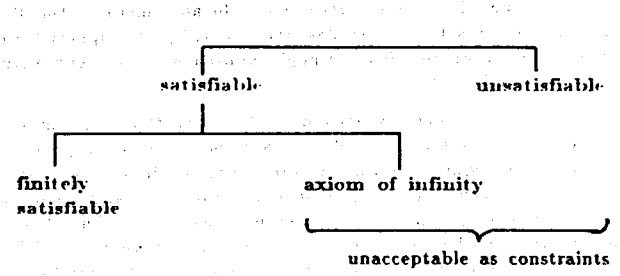

Fig. 1

Because of the undecidability of satisfiability, no algorithm can be constructed that stops for every possible set of formulas and reports whether this set is finitely satisfiable, unsatisfiable or an axiom of infinity. Finite satisfiability, as well as unsatisfiability, is undecidable [17], but both are at least semi-decidable: algorithms can be constructed that are guaranteed to report the respective property after finite (but indefinite) time if applied to a set that actually has this property, but possibly run forever else. Every refutation method is in fact a semi-decision procedure for unsatisfiability. Procedures of this kind have been in use as theorem provers

Proceedings of the Twelfth International Conference on Very Large Data Bases Kyoto, August, 1986

for more than two decades and have nowadays reached a considerable standard of efficiency.

The semi-decidability of both, finite satisfiability as well as unsatisfiability, implies that a procedure can be built that terminates for both, finitely satisfiable as well as unsatisfiable input. Such a procedure would be an adequate tool for deciding whether a given set of formulas is acceptable as constraints (within the limits imposed by undecidability, of course). The simplest way of obtaining a simultaneous semi-decision procedure would be to run two independent procedures for each of the two semi-decidable properties in parallel. However, any attempt to improve the unsatisfactory efficiency of the basic semi-decision method for finite satisfiability (to test for increasing n whether a model of cardinality n exists) inevitably leads to techniques that are also required for refutation. Therefore it seems reasonable to rely on existing refutation methods and to extend them in order to make them sound and complete for finite satisfiability as well.

There are certain classes of formulas where satisfiability and finite satisfiability coincide, called finitely controllable. For sets of formulas coming from such a class, satisfiability is decidable. However, the known finitely controllable classes [7] appear to be too restricted for admitting only constraints that belong to such a class.

In this paper we describe two basic approaches to extending refutation methods into procedures that semi-decide finite satisfiability as well: the one is based on the resolution principle, while the other makes use of a subcase analysis based on splitting of clauses. Both approaches require a common feature (function evaluation) to be added to the underlying refutation principle in order to reach completeness for finite satisfiability. Section 2 describes and motivates this feature after having briefly introduced both approaches. Resolution requires another additional feature if completeness shall be guaranteed. This further extension is described in section 3 together with a suggestion for an improvement of the extended method. In section 4, improvements of the splitting-based approach are proposed that are necessary in order to make it competitive as compared to the resolution-based method.

The method defined in [12] for checking consistency of constraints is based on the first-order tableaux method (in its original non-clausal form as described in [16]). As opposed to the methods proposed here, Kung's approach is not complete for finite satisfiability.

The paper has been written in such a way that only an intuitive understanding of resolution and other theorem proving techniques is required. Formulas are expressed in clausal form, all functions occurring being Skolem functions. Transformation to clausal form is known to preserve satisfiability. This holds for finite satisfiability, too, for similar reasons. The tutorials [4] and [13] provide an introduction to refutation methods in general and to resolution in particular. The term 'resolution' is used throughout this paper according to Robinson's original terminology, i.e., including factorization.

2. Refutation Methods and Finite Satisfiability

A refutation method can be seen as a procedure that successively generates new sets of clauses starting from the set to refute. The generation stops as soon as certain halting conditions - based on syntactical properties of the sets - are fulfilled. For any unsatisfiable input, a refutation-complete procedure is guaranteed to stop, while for satisfiable input it may either stop or run forever. Refutation procedures differ mainly in the way in which the generation of new sets is organized and in the halting conditions employed. Two main classes of procedures can be distinguished with respect to these criteria.

The one class contains procedures that are based on the resolution principle. For a given input set S they generate a sequence $S=S_0 \rightarrow S_1 \rightarrow \ldots \rightarrow S_i \rightarrow S_{i+1} \rightarrow \ldots$ where S_{i+1} is constructed from S_i by addition of factors and/or resolvents of clauses in S_i (possibly combined with a subsequent deletion of tautologies or subsumed clauses). As resolution is an inference rule, we have:

$$(*) \quad \forall\, i \geq 0 \ (S_i \text{ satisfiable} \Leftrightarrow S_{i+1} \text{ satisfiable})$$

Whenever one of the S_i contains the empty clause $\square$, this set is unsatisfiable, because $\square$ is the clausal representation of falsehood. Therefore S is unsatisfiable, too, because of (*) and the generation of new sets stops. If, on the other hand, S is satisfiable, in general it will never stop as none of the S_i contains $\square$. In certain cases, however, a *saturated* set will be reached, i.e., a set that already contains all factors and resolvents (or at least variants of them) that are constructable from its members. In this case generation also stops (reporting satisfiability).

The second class of procedures contains most of the methods that have been proposed and implemented before the resolution principle was developed (e.g., clausal versions of the tableaux method - like the procedure of Gilmore [10] - or the method of Davis and Putnam [6]). In these procedures the generation of new sets follows a tree structure, as shown by figure 2.

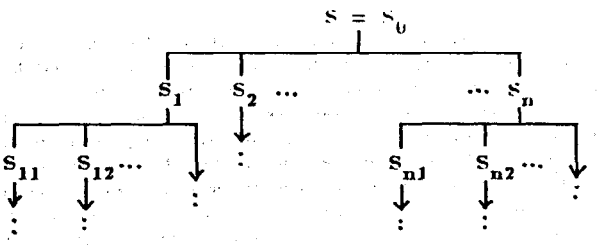

Fig. 2

The tree T_S of figure 2 is expanded (either depth- or breadth-first) reflecting a case analysis. The edges of T_S are constructed in such a way that the following holds:

$$(**) \quad \forall\, S_i \in T_S \ (S_i \text{ satisfiable} \Leftrightarrow S_i \text{ has at least one direct descendant that is satisfiable})$$

As soon as along a branch a set has been reached that contains two contradictory units, this branch is "closed" (i.e., not further expanded) because the respective set is obviously unsatisfiable. If all branches of the tree can be closed in this way, the unsatisfiability of S has been shown because of (**). There may be infinite branches - which can never be closed - as well as finite non-closed branches that cannot be further extended by the construction rules of the method. In the latter case, satisfiability of S is reported.

Figure 3 (see next page) shows the refutation of a four-clause set by means of unit resolution (sequential organization) as well as a clausal version of the tableaux method (tree organization) that uses instantiation of clauses in S and splitting of ground clauses as construction rules. Matrix notation for sets of clauses has been used in the examples each line representing a clause.

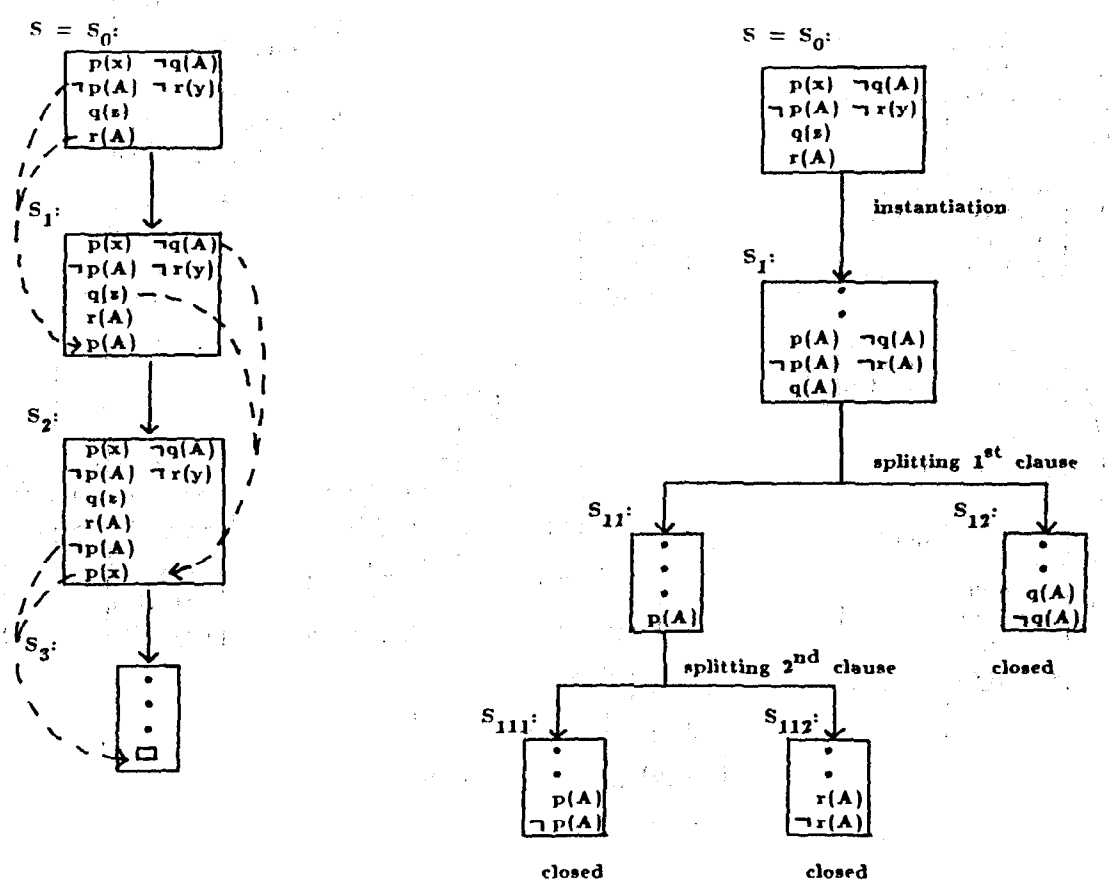

Fig. 3

For detecting unsatisfiability, the sequential approach is superior to the tree approach because closing *all* branches of a tree is more expensive than generating *one* set that contains □. This is one of the reasons why splitting-based methods have been discarded for theorem-proving purposes after resolution was introduced. Symmetrically, satisfiability is *reported* by such a splitting-based method as soon as one non-closed branch has been found (unless an infinite branch is entered) while resolution, e.g., has to wait until *all* possible factors and resolvents have been added (which possibly requires infinite time as well).

A further advantage of many splitting-based methods is that the length of clauses never increases (instances of clauses are added or clauses are replaced by shorter ones). This is not the case for resolution since in general a resolvent is longer than each of its parents.

All refutation-complete methods are necessarily sound for satisfiability: they never report satisfiability when applied to an unsatisfiable set. Undecidability of satisfiability, however, prevents them from being complete for this property. How do refutation procedures behave with respect to finite satisfiability - which is a semi-decidable property in contrast to satisfiability?

Whenever unrestricted resolution stops because a saturated set (*not* containing □) has been reached, the respective input set is finitely satisfiable [3]. Although this result appears to be rather natural, we were not able to find it in the literature. Splitting-based methods are also sound for finite satisfiability. The tableaux method, e.g., stops whenever a set has been constructed that contains a g-model.[1] This g-model directly represents a finite model of S, i.e., S has been shown to be finitely satisfiable in a constructive way.

Thus, both approaches are sound for finite satisfiability, but none of them is complete for this property as shown by figure 4 (see next page).

---

[1] A g-model [13] is a set U of ground units such that each ground instance of a clause in S is subsumed by a unit in U.

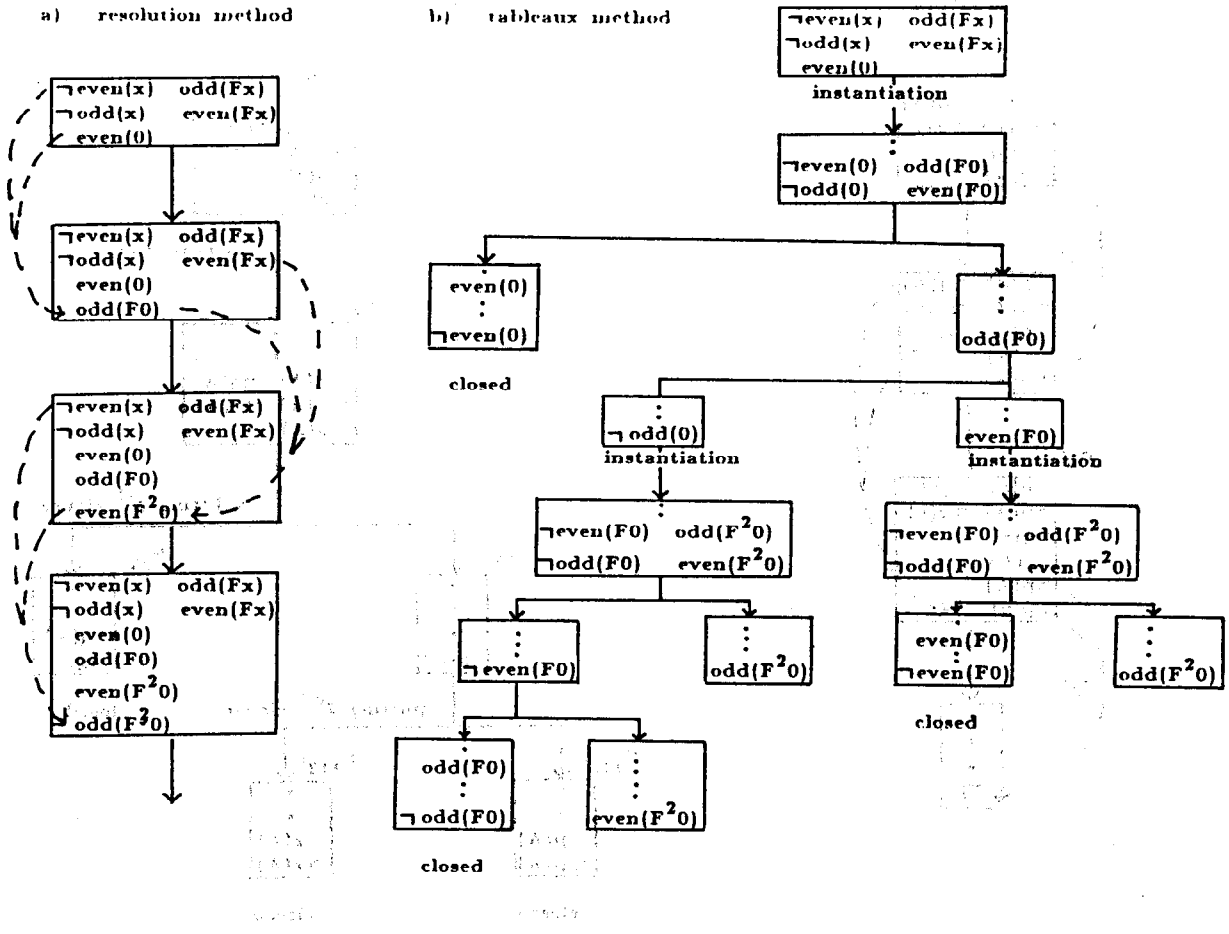

Fig. 4

Both methods run forever, although S has a very small finite model[2]: its only individual is the constant 0, F0 evaluates to 0 and both, even(0) and odd(0), are true. In both cases a purely syntactic mechanism causes an unrestricted growth in nesting of functional terms. In case a) unification leads to an infinite sequence of units

$$even(0) \rightarrow odd(F0) \rightarrow even(F^2 0) \rightarrow odd(F^3 0) \rightarrow \ldots$$

while in case b) instantiation produces a similar sequence along each non-closed branch, e.g.,

$$even(0) \rightarrow odd(F0) \rightarrow even(F0) \rightarrow odd(F^2 0) \rightarrow \ldots$$

along the rightmost one. None of the two methods offers a tool for identifying F0 with 0 and thus detecting that after such a function evaluation each of the infinite sequences would "collaps" into the finite sequence even(0) -> odd(0).

Indeed, adding an evaluation facility for ground functional terms to each of the approaches enables them to detect finite satisfiability in all those cases where infinite growth in function nesting prevents the original methods from stopping. Such a feature for the identification of ground terms is indispensable for finite model detection. This is related to the well-known fact that finiteness is not first-order expressable.

In order to evaluate a ground functional term, a case analysis is required as there may be several possible ground terms with which the functional term could be identified. Therefore resolution loses its sequential organization when extended by function evaluation and is turned into a tree-structure method, too. When added to resolution, evaluation of ground functional terms has, of course, to be combined with instantiation, because sets without any ground terms have to be handled as well. Many splitting-based methods already provide instantiation.

We have shown in [1] that the tableaux method with function evaluation is complete for finite satisfiability. For similar reasons, the same is true for the Davis-Putnam method. Resolution, however, is not complete for finite satisfiability

[2] Due to the fact that S does not completely axiomatize even-odd for the integers.

even after extending it by instantiation and function evaluation. As shown in the following section, infinite growth in clause length may prevent resolution from stopping even for sets without any function symbols.

Figure 5 gives another example presenting the extended tableaux method. This example is intended to show that, in order to evaluate functions, the ground level has to be reached, because in certain cases the decision whether a finite model exists or not requires an explicit identification of two ground terms such that a certain predicate is true for the one but false for the other.

3. The Resolution-Based Approach

In the previous section, we have claimed that extending resolution with function evaluation and instantiation is still not sufficient for achieving completeness for finite satisfiability. There is a second source of infinite growth apart from growth in function nesting [1]. The example of figure 6 (see next page) shows the problem. The clauses in the initial set express that the spouse of a woman is a man, and the spouse of a man is a woman. Resolution does never stop when applied to the initial set because the number of distinct variables (and thus the length of clauses) increases continuously. However, this set has finite models (e.g., one man which is not married). Obviously, function evaluation does not help as no functions occur.

As a solution to this problem we propose another feature that we call *compactification*. Let v(S) denote the maximal number of variables in any clause in the set S. For $n < v(S)$, the n-compactification of S (denoted by $comp_n(S)$) is obtained by replacing each clause C with $m > n$ variables by a set of clauses with exactly n variables. This set is constructed by identifying (m-n+1) of the variables in C in all possible ways.

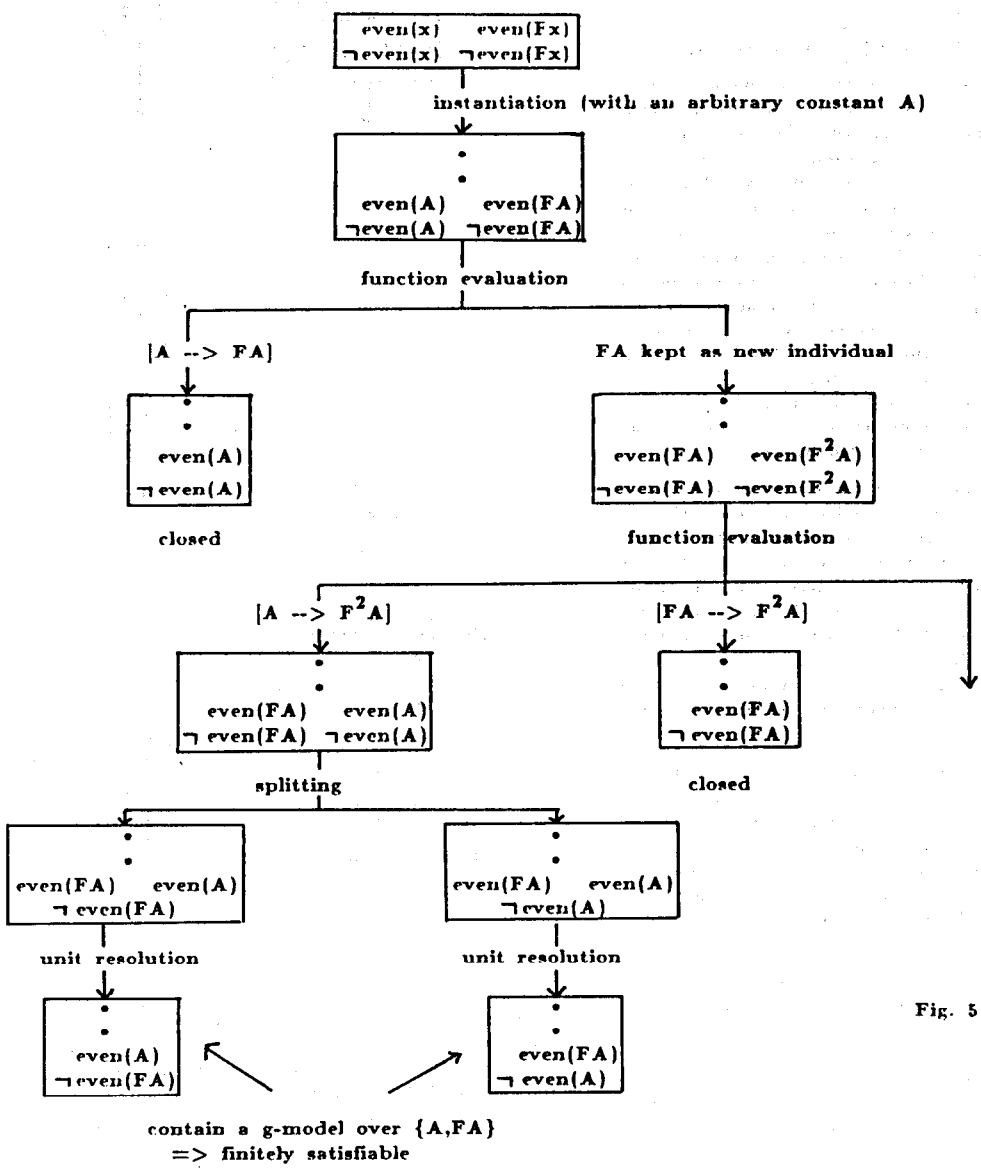

Fig. 5

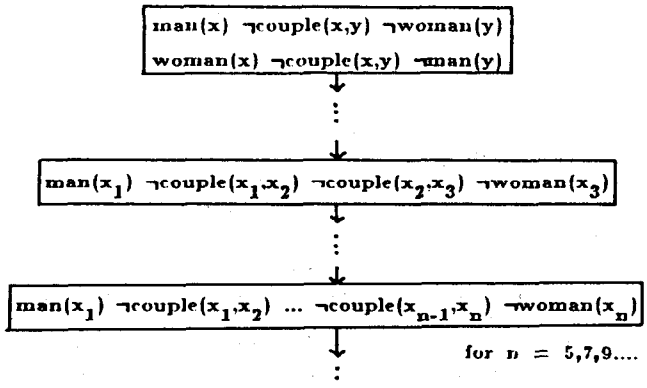

Fig. 6

As an example, we give the 2-compactification of the third clause:

$man(x_1)\ \neg couple(x_1,x_1)\ \neg couple(x_1,x_2)\ \neg woman(x_2)$

$man(x_1)\ \neg couple(x_1,x_2)\ \neg couple(x_2,x_1)\ \neg woman(x_1)$

$man(x_1)\ \neg couple(x_1,x_2)\ \neg couple(x_2,x_2)\ \neg woman(x_2)$

Compactification is applied as soon as resolution produces a clause that has more variables than any of the already existing clauses. After having constructed the respective n-compactification, resolution (and instantiation/function evaluation, if necessary) is applied to the compactified set. Whenever a clause with more than n variables is derived anew from the n-compactification, it is immediately n-compactified, too. If a saturated set is reached, the initial set has been shown finitely satisfiable (with a model of cardinality n). If resolution derives the empty clause for all possible subsequent function evaluations (if any), then such a model does not exist, and we have to backtrack and go on with the uncompactified set. In this case, compactification is invoked again after the next increasement in variable number has occurred and along another "side branch" the existence of a finite model is checked. The overall organization of this approach is illustrated by figure 7.

The application of function evaluation within a sequence of resolution steps is organized in a similar way. Here the increasement of *functional height* serves as an indicator for invocation of function evaluation. The functional height of a term is the level of nesting of functions in that term, e.g., the functional height of F(Fxy, y) is 2. The maximal functional height of a term in S is denoted by f(S). Whenever resolution leads to an increasement of f(S), function evaluation is invoked on a "side branch" while resolution will go on without evaluating the new functional term on the "main branch" in case backtracking is required, as shown by figure 8.

If both kinds of increasements occur for a given S, invocations of both additional features - compactification as well as function evaluation - have to be merged, of course. A saturated set on a "side branch" indicates finite satisfiability, while closed "side branches" plus a closed "main branch" indicate unsatisfiability of S. For axioms of infinity the "main branch" never closes while all "side branches" are closed.

The extended resolution method outlined here may be combined with any strategy provided that refutation-completeness is preserved. In his paper mentioned, Joyner proposes a strategy that would save a lot of instantiations and function evaluation steps although it requires a transformation of the initial set: all resolvents and factors containing nested functional terms may be discarded if the initial set is the clausal representation of formulas in Skolem normal form [5], i.e., in

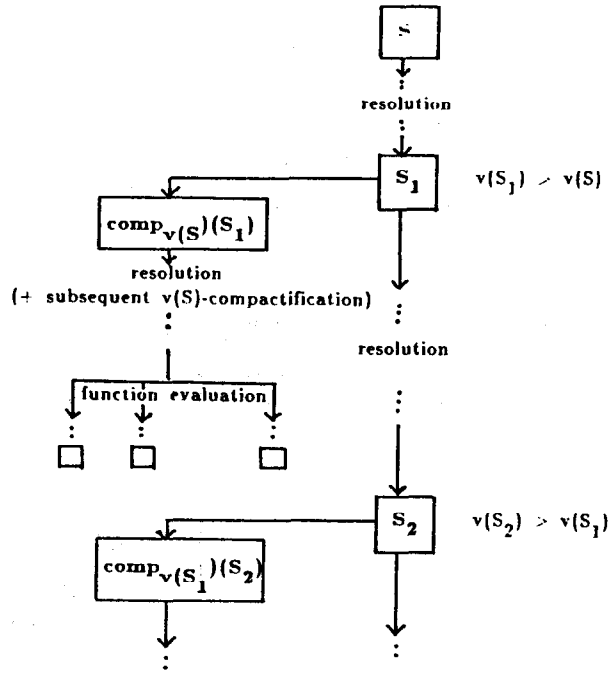

Fig. 7

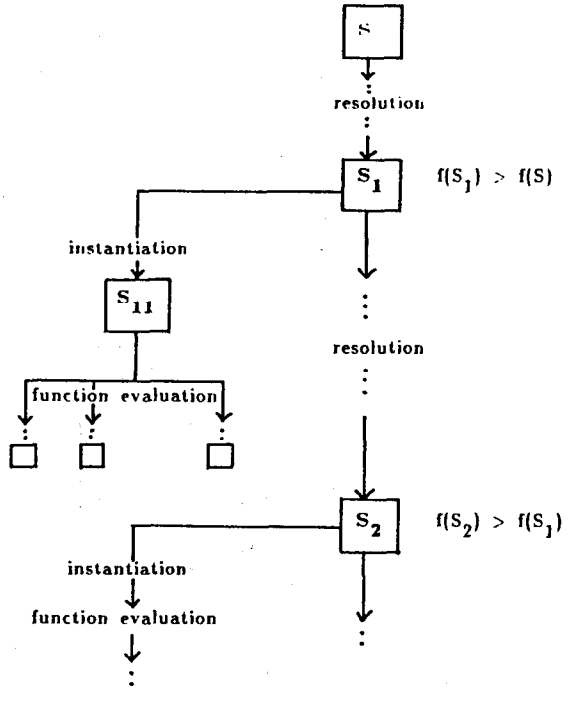

Fig. 8

prenex normal form with prefix $\forall^* \exists^*$. For any first-order formula F there exists a formula SNF(F) in Skolem normal form such that F is finitely satisfiable if and only if SNF(F) is finitely satisfiable. The construction of this normalized formula is illustrated by an example:

F: $\forall x \exists y \forall z \; [p(y,x) => q(y,z)]$

SNF(F): $\forall x \forall y \forall z \exists w \; ([[p(y,x) => q(y,z)] \vee s(x,y)) \wedge \neg s(x,w))$

The new predicate s is not allowed to occur in F. It was known before that this transformation preserves unsatisfiability. In [2] we have shown that this is also the case for finite satisfiability.

Although operating with formulas in Skolem normal form allows to discard clauses with nested functions, function evaluation and instantiation do not become superfluous. There are still cases in which the evaluation of unnested functional terms is required in order to guarantee the soundness of the extended method.

4. Improvements of the Splitting-Based Approach

In Section 2, splitting-based refutation methods were shown to be complete for finite satisfiability if combined with function evaluation. We had chosen the tableaux method as a representative of this class of methods because of the simplicity of its splitting rule. However, this method is by far too inefficient for practical applications if compared, e.g., with resolution-based methods. Two reasons are responsible for this inefficiency:

- the only construction rule of the tableaux1 method - clause splitting - is too primitive
- a vast amount of instantiations is required because clause splitting is only applicable to ground clauses

A more efficient set of rules for testing unsatisfiability of a set of ground clauses is available in the Davis-Putnam procedure. The rules of this method take into account several clauses in S instead of looking only at a single, isolated clause in each step. This leads to trees which are in general considerably smaller than the trees constructable with the rules of the tableaux method. Four construction rules are provided by the Davis-Putnam procedure:

1. deletion of tautological and subsumed clauses
2. ground unit resolution
3. introduction of new units: if a *pure* ground literal L occurs in S (i.e., a literal the complement of which does not occur in S), then the unit {L} can be added to S (allowing a subsequent elimination of all clauses that contain L as they are subsumed by the new unit)
4. complement splitting: if L is a non-pure ground literal in S, then two subcases can be introduced. In the one case the unit {L}, in the other case $\{L^c\}$ is added (L^c denotes the complement of L). (In each of the cases the new unit subsumes at least one clause and can be resolved against at least one literal.)

Nevertheless, the Davis-Putnam method suffers from the same drawback as the tableaux method, namely to require instantiations as all its rules operate only on ground clauses. The number of instances of a clause depends exponentially on the number of ground terms that are used for instantiation. Resolution refutation procedures do not need any explicit instantiation at all. This fact makes them superior to both, Davis-Putnam as well as tableaux method for refutation purposes. But, as pointed out in the previous section, instantiation has to be added to resolution, in order to yield completeness for finite satisfiability. However, these inevitable instantiation steps are performed as late as possible and only in those cases where absolutely necessary for making function evaluation possible. It would be desirable to reduce the number of instantiations required by the Davis-Putnam procedure in a comparative way.

A necessary prerequisite for such a reduction is that the construction rules of the method are somehow generalized to the non-ground level (in the same way as general resolution has been originally introduced as a generalization of the propositional "cut" rule). The first three Davis-Putnam rules can be easily generalized. Elimination of tautologies and subsumed clauses is a standard feature of many reduction strategies for resolution procedures. Unit resolution is a special case of general resolution (known to be refutation-complete for the important class of Horn clauses). The notion of a pure literal is also easily extendable to non-ground literals if instances and variants of the complementary literal are taken into account.

Problems arise if the complement splitting rule shall be extended. The introduction of alternative L-L^c cases is justified by the fact that ($L \vee L^c$) is a tautology for ground literals. On the general level, any variable x in a literal L has to be regarded as implicitly universally quantified. The disjunction ($\forall x|L] \vee \forall x|L^c]$), however, is not a tautology. Therefore a direct generalization of complement splitting to the general level is not possible without losing completeness. In [4, p. 184] generalized splitting rules are investigated that overcome this problem by keeping track of all variable substitutions performed along the alternative branches and checking their compatibility at the end. It is not clear how such methods can be adapted for finite satisfiability checking.

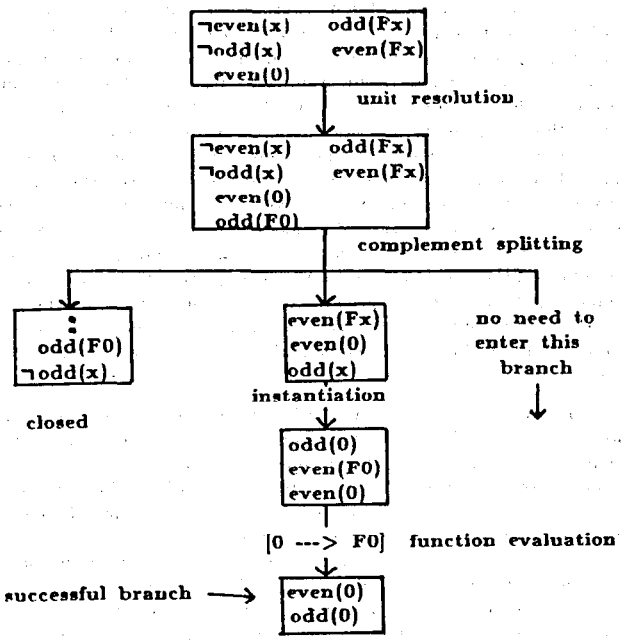

Fig. 9

The solution we propose is based on the idea to view each of the alternative case introduced by complement splitting as a kind of "test of an assumption". Does any finite model of S exist in which the (additional) unit {L} - respectively $\{L^c\}$ - is true? If one of the assumptions leads to a success, the other assumption needs not be tested any more. If both assumptions lead to a closed branch, then it has only been shown that these assumptions are not compatible with S, but not that S is unsatisfiable because of the above reasons. A third branch

has therefore to be provided by each complement splitting step on which, in case of "failure of both assumptions", the two literals L and L^c are removed from the list of "unit candidates" and other non-pure literals are tested. If all of them lead to closed branches, then instantiation has to be invoked on the remaining branch. However, this is the only case in which instantiation is required at all by the generalized version of the Davis-Putnam method described here. If S is unsatisfiable, then none of the "unit candidates" will be successful, and instantiation is inevitable. For many satisfiable sets, however, a few applications of the generalized complement splitting lead to a non-closed finite branch already.

Figure 9 (see previous page) shows how the set used in figure 4 is checked for finite satisfiability using the generalized Davis-Putnam procedure with function evaluation.

5. Conclusion

In this paper we have introduced finite satisfiability as a necessary wellformedness condition for database constraint sets. It reflects the requirement that constraints have to admit finite sets of facts (in a conventional as well as in a deductive context). Finite satisfiability is a stronger property than consistency thus automatically implying the latter.

In order to obtain a method for finite satisfiability checking we have chosen to extend existing refutation procedures. Two different approaches to finite satisfiability checking have been investigated that are based on two different approaches to refutation. Both require the same extension, namely the addition of an evaluation facility for ground functional terms in order to control growth in function nesting. This addition prevents production of infinitely many clauses by identifying certain ground terms. The approach based on the resolution principle has to be further extended with a feature for the limitation of growth in clause length, called compactification. Of course, the addition of these features to refutation procedures will decrease their efficiency for unsatisfiable input. However, this price has to be paid if completeness for finite satisfiability shall be reached. The virtues of both approaches cannot be simply combined, as the undecidability of satisfiability prevents any refutation-complete procedure from removing both kinds of growth at the same time. Each of the two methods obtained by the extensions mentioned is sound and complete for finite satisfiability as well as for unsatisfiability.

The two methods are justified and described in detail in [2] and [14], respectively. At the moment, we don't see any other reasonable solution to the problem of constructing a simultaneous semi-decision procedure for finite satisfiability and unsatisfiability. Although we have no results about the differences in efficiency between the two methods, we believe that a splitting-based approach will be preferable. Resolution produces too many clauses especially if applied to compactified sets of clauses. Moreover, the big advantage of resolution-based refutation procedures - namely to do without any instantiation - is lost, as function evaluation has necessarily to be performed on the ground level. A first prototype implementation of the splitting-based method written in Prolog is meanwhile available.

Additional points that may influence the work on a more elaborate version are:

1. For many data models (especially those providing generalization hierarchies) a many-sorted logic is more appropriate. Introduction of many-sortedness is known to improve the efficiency of the methods discussed in this paper.

2. The question of suitable strategies has not been addressed in this paper at all. A thorough investigation of this topic is inevitable. For resolution a lot of strategies have already been introduced in the context of refutation. It has to be investigated whether they can be adapted for the extended method as well. Strategies for the generalized Davis-Putnam procedure should especially provide criteria for making good choices of "unit candidates" when applying complement splitting.

3. Very often we can expect that a considerable part of the constraint set under consideration consists of dependencies that are known to be finitely satisfiable because of their syntactical structure. Strategies should be developed that take advantage of this knowledge. Similar techniques can be useful in a context where constraint sets are modified.

Acknowledgement: We would like to thank Jean-Marie Nicolas for his advice and encouragement during the preparation of this paper.

References

1. Bry, F. Note on Consistency Checking of Database Schemas. Int. Report KB-4, ECRC, Jul., 1985.

2. Bry F. The Compactification Method. Int. Report KB-6, ECRC, Sept., 1985.

3. Bry, F. On Resolution and Finite Satisfiability. in preparation.

4. Chang, C.-L. and Lee R.C.-T. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

5. Church, A. *Introduction to Mathematical Logic.* Princeton University Press, 1956. cited in [11].

6. Davis, M. and Putnam, H. "A Computing Procedure for Quantification Theory". *JACM* 7, 3 (Jul. 1960), 201- 215. also in [15]:125-139.

7. Dreben, B. and Goldfarb, W. *The Decision Problem. Solvable Classes of Quantificational Formulas.* Addison-Wesley, 1979.

8. Fagin, R. and Vardi M.Y. The Theory of Data Dependencies - An Overview. Proc. of ICALP, 1984.

9. Gallaire, H., Minker J. and Nicolas, J.-M. "Logic and Databases: A Deductive Approach". *ACM Comp. Surv. 16,* 2 (Jun. 1984), 153-185.

10. Gilmore, P.C. "A Proof Method for Quantification Theory: Its Justification and Realization". *IBM J. of Res. and Dev. 4* (1960), 28-35. also in [15]:151-158.

11. Joyner, W.H. "Resolution Strategies as Decision Procedure". *JACM 23.* 3 (Jul. 1976), 398-417.

12. Kung, C.H. *A Temporal Framework for Information Systems Specification and Verification.* Ph.D. Th., U. of Trondheim, Norway, Apr. 1984.

13. Loveland, D.W. *Automated Theorem Proving: a Logical Basis.* North-Holland, 1978.

14. Manthey, R. A Generalized Davis-Putnam Procedure Able to Detect Finite Satisfiability. in preparation.

15. Siekmann, J. and Wrightson, C.. *Automation of Reasonning, Vol. 1.* Springer Verlag, 1983.

16. Smullyan, R. *First-Order Logic.* Springer Verlag, 1968.

17. Trachtenbrot, B.A. "Impossibility of an Algorithm for the Decision Problem in Finite Classes". *Dokl. Acad. Nauk. SSSR 70* (1950). in Russian, trans. in English in Amer. Soc. Translations, Series 2, 23:1-5, 1963.