# Solving Database Satisfiability Problems

François Bry and Sunna Torge
Computer Science, University of Munich
http://www.pms.informatik.uni-muenchen.de

**Abstract.** Databases can be formalized as models of first-order formulas expressing either static integrity constraints, views, updates, or dynamic integrity constraints. In designing and managing databases, not only satisfaction – *are given formulas satisfied in a given database?* – but also satisfiability – *are there databases satisfying given formulas?* – of first-order formulas has to be verified. In this paper, minimal finite satisfiability is proposed as a unifying framework for several database problems and a nonstandard deduction method for verifying database satisfiability is described.

## 1 Introduction

Several database issues are conveniently formalized when databases are seen as models of finite sets of first-order formulas.

If static integrity constraints are expressed as first-order formulas, the issue known as "database consistency verification", i.e., whether a database satisfies its static integrity constraints, can be formalized as the *satisfaction* of formulas in a model. In contrast, whether static integrity constraints are correctly designed in the sense that there exist databases enforcing them, can be formalized as a *satisfiability* problem. Since databases correspond to finite models, the satisfiability notion of concern is satisfiability over finite models, short "finite satisfiability".

Further database issues that can be seen as satisfaction problems are "view materialization" and "view querying". Further database issues that can be seen as finite satisfiability problems are the so-called "view update problem", i.e., if and how an update of views can be realized by updating the actually stored data, the issue called "integrity constraint repair", i.e., how an update could be modified so as not to violate some (static or dynamic) integrity constraints, and whether some (static or dynamic) integrity constraints might become redundant after some (possibly parametrized) update.

This paper is devoted first to proposing finite satisfiability as a unifying framework for formalizing and solving several database issues, second to describing a deduction method for verifying this property.

The rest of this extended abstract is organized as follows. In the next section, the aforementioned database issues are discussed in more details. Section 3 outlines a deduction method sound and complete for unsatisfiability and finite satisfiability. In Section 4, this method is enhanced to ensure model minimality yielding the automated reasoning tool needed for solving the considered database problems. Section 5 discusses related work in databases and automated deduction and gives perspectives for further research.

## 2 Satisfaction vs. Satisfiability in Databases

It is by now standard to view a database as a finite set of ground atoms [1], i.e., as a Herbrand interpretation [11]. Although this view best fits with the relational data model, it by no means precludes advanced data modeling features not directly expressible in classical logic such as object orientation, object identity, and cyclic terms. For the purpose of the present paper, such extensions can be ignored.

In the following, database satisfaction and satisfiability problems are contrasted in order to clarify what database satisfiability problems are. The better known database satisfaction problems are first recalled.

### 2.1 Database Satisfaction Problems

**Query Answering.** Database queries correspond to first-order logic formulas with free variables. Answering queries correspond to verifying the satisfaction of formulas in the logic interpretation representing the database.

**Database Consistency Verification.** Static integrity constraints correspond in logic to formulas without free variables. Verifying whether a database satisfies its integrity constraints, i.e. is consistent, consists in checking whether some formulas are satisfied in a logical interpretation.

Much research has been devoted to an efficient verification of this problem. The proposed methods basically perform an incremental verification based on the last update and on the assumption that before this update the database did satisfy its integrity constraints (cf. eg. [9]).

**View Materialization.** Views are predefined queries corresponding to formulas with free variables. These variables correspond to attribute values of items, tuples, or objects defined by the views. View materialization consists in efficiently computing and maintaining the data items specified by views. Like consistency verification, view materialization is performed by incremental methods based on the last update. Thus, view materialization basically consists in answering queries. Hence it is a satisfaction problem.

**View Querying.** View querying is the query-answering problem arising when views are not materialized. Query answering has to cope with view definitions that might involve recursive definitions. Like database consistency and view materialization, view querying formally reduces to the satisfaction of formulas in an interpretation.

### 2.2 Database Satisfiability Problems

Some database problems are solved by constructing all or part of a database in which a given finite set of formulas should be satisfied. Such problems can be expressed as the search for models of finite sets of formulas. Since databases are finite, only finite models are of interest. In the following, it is shown that the restriction to minimal models is possible and desirable.

**Static Integrity Constraint Design.** While designing a database, the static integrity constraints are often designed before the database is populated with data. In doing so, it might happen that the set of static integrity constraints is contradictory, i.e. logically inconsistent. It might also happen that this set is consistent, but only admits infinite models. In both cases, the static integrity constraints are ill-designed. Acceptable are those sets of static integrity constraints that have finite models, i.e. that are finitely satisfiable.

Such models are "sample" databases that can be used by the database designer for investigating typical data patterns [16]. Clearly, both in verifying finite satisfiability and in investigating data patterns, it is sufficient to restrict oneself to minimal sample databases, i.e. databases that no longer satisfy their static integrity constraints when some of there data items are removed. Such databases correspond in logic to minimal finite models of the static integrity constraints.

**View Update.** Consider a simple Datalog view like $p(x, y, z) \leftarrow q(x, y),\ r(y, z), s(z)$. In order to delete a $p$ object, one has to delete one of the corresponding $q$, $r$ or $s$ object. This is conveniently expressed by the formula $\forall x \forall y \forall z\ del\_p(x, y, z) \Rightarrow (del\_q(x, y) \vee del\_r(y, z) \vee del\_s(z))$. Similarly, the following formula – or a refinement of it – can be used to formalize view insertions: $\forall x \forall y \forall z\ ins\_p(x, y, z) \Rightarrow (ins\_q(x, y) \wedge ins\_r(y, z) \wedge ins\_s(z))$. Using such formulas, it is possible to formalize updates of more complex views – possibly involving negation – than the example considered above. A solution to a view update specified by $ins\_p(a, b, c)$ then is a finite model of the considered database extended with the view update formulas. Clearly, only finite models are acceptable solutions. The minimality principle common in view update and more generally in knowledge revision [13] is conveyed by the restriction to minimal models.

**Integrity Constraint Repair.** This issue is that of extending a given update that would result in the violation of some static integrity constraint in such a manner that the modified update yields a consistent database [12]. Viewing an integrity constraint $C$ as a view $inconsistent \leftarrow \neg C$ makes it possible to reduce integrity constraint repair to the view update $del\_inconsistent$. Thus, like view updates, integrity constraint repairs can be formalized as satisfiability problems. Here again, the restriction to finite repair is necessary. Moreover, it is natural to restrict oneself to minimal repairs.

**Integrity Constraint Redundancy.** Since the verification of (static or dynamic) integrity constraints generally is a time consuming task, it is desirable to detect and discard redundant integrity constraints. An integrity constraint $C$ is redundant with respect to a set $S$ of integrity constraints if all databases satisfying the constraints in $S$ also satisfy $C$. For efficiency reasons, it makes sense to restrict the verification to databases that are minimal models of $S$. Up to the fact that databases are finite, this is the usual notion of redundancy. However, the finiteness condition makes it nonreducible to inconsistency as this is the case with classical redundancy.

## 3 The EP Tableau Method

In [10, 20] a deduction method called EP Tableau Method is proposed for verifying the finite satisfiability of finite sets of range restricted (or safe) formulas by generating models. The method performs a systematic search for "term models" of the considered formulas in the fashion introduced by the so-called "tableau methods" [21]. Term models are similar to Herbrand models except that their universes are not necessarily infinite.

The method proceeds by decomposing the considered formulas up till only literals remain. A conjunction $F \wedge G$ is decomposed into the two formulas $F$ and $G$, a disjunction $F \vee G$ leads to a case analysis: In a first case it is replaced by $F$, in a second case by $G$. Negated complex formulas are decomposed by propagating the negation into their subformulas according to the De Morgan laws. Implications and equivalences are dealt with according to their expressions in terms of negations, conjunctions, and disjunctions. Quantifiers are eliminated by instantiating the variables they qualify as follows. A quantification $\forall x F[x]$ leads to (1) generating formulas $F[t]$ for all possible ground terms $t$ currently available, and (2) keeping the formula $\forall x F[x]$ for making it possible to generate $F[t']$ for those ground terms $t'$ that might later arise. A formula $\exists x F[x]$ leads to a case analysis such that for each previously introduced ground term $t$ there is a case in which $\exists x F[x]$ is replaced by $F[t]$, and there is an additional case where $\exists x F[x]$ is replaced by $F[t_{new}]$ for some ground term $t_{new}$, which does not occur anywhere in the previously constructed search space. This treatment of existential formulas departs from the approach of conventional tableau methods. It ensures that the EP Tableau Method always constructs finite models if some exist.

The method is sound and complete for both, unsatisfiability and finite satisfiability. That is, when applied to one of the aforementioned database satisfiability problems it will always report the nonexistence of solutions, if the considered specification is inconsistent, or return a (finite) solution, if some exist. Since finite satisfiability is not decidable but only semi-decidable, the remaining cases – all models of the considered specification are infinite – cannot always be detected. The visualization tool described in [6] gives rise to trace (finite or infinite) model construction. This often helps to find bugs in specifications whose models are all infinite.

## 4 Ensuring Model Minimality

A model $M$ of some set $S$ of first-order formulas is minimal when removing some tuples from the relations in $M$ results in an interpretation $M'$ which is no model of $S$.

In [17, 19] a characterization of minimal models is given, which fits well with the principle of the EP Tableau Method. A set of ground atoms $M$ is a minimal model of a set $S$ of first-order formulas if (1) it is a model of $S$ and (2) $S \cup \{\neg B \mid B$ is a ground atom and $B \notin M\}$ logically implies $M$ (in the sense of classical logic). As explained in [17] this characterization yields an incremental test which can be performed during the construction of $M$ with a procedure like the EP Tableau Method. The resulting procedure remains complete for unsatisfiability and finite satisfiability and all models it returns are minimal models of the considered specification.

## 5 Related Work and Perspectives

**Related work in Databases.** Many *ad hoc* methods have been proposed for solving database satisfiability problems. [7, 9] first observed that integrity constraint design is a satisfiability problem. In [16] this approach is adapted to the related issue of investing patterns of data compelling to the integrity constraints. In [4] a method is proposed for verifying the satisfiability of integrity constraints in object-oriented databases. Whether this method is complete for finite satisfiability is not clear. Much literature has been devoted to the view update problem and many methods have been proposed to solve (instances of) it. This problem has been first presented as a satisfiability problem in [5]. However, this article does not mention the restriction to minimal models. The need for model minimality is mentioned in [2, 3]. Integrity constraint repair has been investigated in [12] where an *ad hoc* method is proposed. Compared with generating repairs as finite minimal models, this method has the drawback of not always terminating when finite repairs exist. Integrity constraint redundancy is rarely mentioned explicitly. The related issue of finite entailment ($\models_{\text{fin}}$) is mentioned in [1]. Up till now, both issues have received little attention.

**Related work in Automated Deduction.** The EP Tableau method is based on a principle mentioned in [8, 14, 15]. In [18, 22] model generators are described that construct models up to a given cardinality. In contrast, the EP Tableau Method does not require such an upper bound. This makes it complete not only for finite satisfiability but also for unsatisfiability. This capability of the EP Tableau Method is essential for solving database satisfiability problems.

**Perspectives.** Extended as described in section 4, the EP Tableau Method is no perfect tool for solving database satisfiability problems. Although it provides with the needed reasoning core, it neither supports the usual data modeling constructs nor a user-friendly specification language. It has to be extended with advanced modeling constructs such as cyclic terms, object identity, and inheritance. A more convenient language than first-order logic is needed for specifying practical database problems. In a companion project a "controlled English" language is currently coupled with the EP Tableau Method. Such a formalism with a natural language flavor should be more appealing to database designers than logic.

## References


[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] C. Aravindan and P. Baumgartner. A rational and efficient algorithm for view deletions in databases. In *Proc. Int. Logic Programming Symp*. MIT Press, 1997.

[3] C. Aravindan and P. Baumgartner. Theorem proving techniques for view deletions in databases. Technical report, Computer Science, University of Coblence, 1998.

[4] V. Benzaken and X. Schaefer. Static management of integrity in object-oriented databases: Design and implementation. In *Proc. 6th Int. Conf. on Extending Database Technology*, LNCS 1377, 1998.

[5] F. Bry. Intensional updates: Abduction via deduction. In *Proc. 7th Int. Conf. on Logic Programming*. MIT Press, 1990.

[6] F. Bry, N. Eisinger, H. Schütz, and S. Torge. SIC: An interactive tool for the design of integrity constraints (system description). In *Proc. Demo Session, Int. Conf. on Extend. Datab. Techn.*, 1998.

[7] F. Bry and R. Manthey. Checking consistency of database constraints: A logical basis. In *Proc. 12th Int. Conf. on Very Large Data Bases*, 1986.

[8] F. Bry and R. Manthey. Proving finite satisfiability of deductive databases. In *Proc. 1st Workshop on Computer Science Logic*, LNCS 329. Springer-Verlag, 1987.

[9] F. Bry and R. Manthey. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In *Proc. Int. Conf. Extending Data Base Technology*, 1988.

[10] F. Bry and S. Torge. A deduction method complete for refutation and finite satisfiability. In *Proc. 6th European Workshop on Logics in Artificial Intelligence*, LNAI 1489. Springer-Verlag, 1998.

[11] C. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, 1990.

[12] M. Gertz and U. Lipeck. An extensible framework for repairing constraint violations. In *Proc. 1st Working Conf. on Integrity and Internal Control in Information Systems*. Chapman & Hall, 1997.

[13] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, 1988.

[14] J. Hintikka. Model minimization – an alternative to circumscription. *Jour. of Automated Reasoning*, 4, 1988.

[15] S. Lorenz. A tableau prover for domain minimization. *Jour. of Automated Reasoning*, 13, 1994.

[16] A. Neufeld, G. Moerkotte, and P. C. Lockemann. Generating consistent data tests: Restricting the search space by a generator formula. *VLDB Jour.*, 2:173–213, 1993.

[17] I. Niemelä. A tableau calculus for minimal model reasoning. In *Proc. 5th Int. Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 1071. Springer-Verlag, 1996.

[18] J. Slaney. FINDER (finite domain enumerator): Notes and guides. Technical report, Australian National University, Automated Reasoning Project, 1992.

[19] M. A. Suchenek. First-order syntactic characterizations of minimal entailment, domain-minimal entailment, and Herbrand entailment. *Jour. of Automated Reasoning*, 10:237–263, 1993.

[20] S. Torge. Überprüfung der Erfüllbarkeit im Endlichen: Ein Verfahren und seine Anwendung. PhD thesis, Computer Science, University of Munich, 1998.

[21] G. Wrightson, ed. Special issue on automated reasoning with analytic tableaux – part I, part II. *Jour. of Automated Reasoning*, 13(2,3), 1994.

[22] J. Zhang and H. Zhang. SEM: A system for enumerating models. In *Proc. Int. Joint Conf. on Artificial Intelligence*, volume 1, 1995.