

# Avenues to Flexible Data Integrity Checking

Hendrik Decker

Instituto Tecnológico de Informática  
Ciudad Politécnica de la Innovación  
Campus de Vera, edificio 8G, E-46071 Valencia, Spain  
hendrik@iti.es

Davide Martinenghi

Free University of Bozen/Bolzano  
Faculty of Computer Science  
P.zza Domenicani, 3, I-39100 Bolzano, Italy  
martinenghi@inf.unibz.it

## Abstract

*Traditional methods for integrity checking in relational or deductive databases heavily rely on the assumption that data have integrity before the execution of updates. In this way, as has always been claimed, one can automatically derive strategies to check, in an incremental way, whether data preserve their integrity after the update. On the other hand, this consistency assumption greatly reduces applicability of such methods, since it is most often the case that small parts of a database do not comply with the integrity constraints, especially when the data are distributed or have been integrated from different sources. In this paper we revisit integrity checking from an inconsistency-tolerant viewpoint. We show that most methods for integrity checking (though not all) are still applicable in the presence of inconsistencies and may be used to guarantee that the satisfied instances of the integrity constraints will continue to be satisfied after the update.*

## 1 Introduction

Traditional methods for integrity checking try to overcome the infeasibility of brute force testing of integrity constraint satisfaction each time an update is executed by deriving incremental checks, tailored to the specific update, that guarantee compliance with the constraints in the updated state. The price to pay is that the integrity constraints must be fully satisfied in the “old” state before the update in order for these checks to help preserving the intended database semantics — or so it has been claimed so far. This nearly unsatisfiable requirement has rendered virtually all techniques for efficient integrity checking unusable in real cases and, indeed, very little attention has been given so far to incremental integrity checking in practical implementations, despite more than two decades of research in this area.

We show in this paper that most approaches to efficient

integrity checking (but unfortunately not all) are also applicable to databases that do not satisfy their integrity constraints, i.e., they have a property that we refer to as *inconsistency tolerance*. Such methods, in these cases, cannot be used to guarantee that the “new”, updated state will fully satisfy the integrity constraints (which is very unlikely, unless the update in question performs a complete consistency repair). However, their incremental checks may be exploited to guarantee something that is arguably as valuable, namely that all instances of the integrity constraints that were satisfied in the old state will remain satisfied in the new state. Therefore, in the presence of inconsistencies in a database, with an inconsistency-tolerant approach, we are able to efficiently preserve satisfaction of those parts of the integrity theory that were already satisfied before the update, while the violated parts will certainly not increase and may also, accidentally or by nature of the update, become satisfied in the new state. In other words, with these measures, integrity of a database can only improve. It should be clear then that inconsistency tolerance, by removing the strict requirement of full integrity in the old state, greatly extends applicability and flexibility of integrity checking methods without affecting their efficiency.

After giving an abstract characterisation of integrity checking methods in section 2 and introducing the notion of inconsistency tolerance in section 3, we revisit a few methods under this new perspective in section 4. We discuss related work in section 5 and conclude in section 6.

## 2 The gist of integrity checking methods

We provide here an apparatus of definitions that characterise the integrity checking problem. We refer to the terminology of deductive databases [1], and conceive a database as a finite set of *facts* and *rules*. *Integrity constraints* are semantic conditions on the data and may be expressed, e.g., as closed first-order formulas, perhaps in the form of DATALOG denials, which are supposed to always hold in the database; an *integrity theory* is a finite set of integrity

constraints. For simplicity, we refer to databases that have a unique standard model, such as stratified databases, and assume that database semantics is defined by this model.

For a database  $D$ , we write  $D \models F$  (resp.,  $D \not\models F$ ), where  $F$  is a closed formula, to indicate that  $F$  evaluates to *true* (resp., *false*) in  $D$ 's standard model; a similar notation is used for an integrity theory, evaluated as the conjunction of its elements. We also say that a database is *consistent* if its associated integrity theory evaluates to *true* in it, *inconsistent* otherwise.

A database *update*  $U$  can be regarded as a mapping  $U : \mathcal{D} \mapsto \mathcal{D}$ , where  $\mathcal{D}$  is the space of databases; we indicate the image of  $D$  via update  $U$  as  $D^U$ .

The integrity checking problem asks, given an integrity theory  $\Gamma$ , a database  $D$  consistent with  $\Gamma$ , and an update  $U$ , whether  $\Gamma$  is still satisfied in  $D^U$ , i.e., whether  $D^U \models \Gamma$ . Since evaluating  $\Gamma$  in  $D^U$  may be highly time consuming, the problem is reformulated so as to exploit incrementality of updates: by assuming  $D$ 's consistency, possibly only a subset of  $D^U$  needs to be checked. It is customary to look for an alternative integrity theory  $\Upsilon$  (called a *simplification*), which is supposed to be simpler to evaluate than  $\Gamma$ , while yielding equivalent results. If  $\Upsilon$  is an integrity theory to be evaluated in  $D^U$ , it is called a post-test; if in  $D$ , a pre-test.

**Definition 2.1** *Let  $\Gamma$  be an integrity theory and  $U$  an update. An integrity theory  $\Upsilon$  is a*

- post-test of  $\Gamma$  for  $U$  whenever  $D^U \models \Gamma$  iff  $D^U \models \Upsilon$
- pre-test of  $\Gamma$  for  $U$  whenever  $D^U \models \Gamma$  iff  $D \models \Upsilon$

for every database  $D$  consistent with  $\Gamma$ .

Clearly,  $\Gamma$  itself is a post-test of  $\Gamma$  for any update, but, as mentioned, one is interested in post-tests that are actually “simpler” than the original  $\Gamma$ , which can be obtained, e.g., by avoiding redundant checks of cases that are already known to satisfy integrity. In particular, one expects such tests to be easier to evaluate than reference pre- and post-tests, called *plain* tests, corresponding to somewhat non-simplified conditions that do not exploit any knowledge about integrity in the old state.

**Definition 2.2** *Let  $\Gamma$  be an integrity theory and  $U$  an update.*

1. An integrity theory  $\Sigma_0$  is a plain pre-test of  $\Gamma$  for  $U$ , denoted by  $\text{pre}_0^U(\Gamma)$ , if  $D \models \Sigma_0$  iff  $D^U \models \Gamma$  for every database  $D$ .
2. An integrity theory  $\Upsilon_0$  is a plain post-test of  $\Gamma$  for  $U$ , denoted by  $\text{post}_0^U(\Gamma)$ , if  $D^U \models \Upsilon_0$  iff  $D^U \models \Gamma$  for every database  $D$ .

Clearly,  $\Gamma$  is a plain post-test of itself for any update.

Due to space limitation, we do not discuss how to achieve simplification of integrity constraints and refer to surveys on the subject, e.g., [16]. We just point out that, with simplification-based methods (such as [17, 8, 13] for post-tests and [18, 5] for pre-tests), updates are committed only if  $D^U$ 's consistency is confirmed by the simplified test.

Methods for integrity checking may, alternatively, propose efficient checking strategies rather than symbolic simplifications; this is the case of, e.g., [19] and [10]. More generally, we may then identify an integrity checking method with a function  $\mathcal{M}(D, \Gamma, U)$  that takes as input a database  $D$ , an integrity theory  $\Gamma$  and an update  $U$  and outputs either *satisfied* or *violated* to indicate the integrity status of  $D^U$ .

### 3 Inconsistency tolerance

In some contexts, some violations of integrity constraints may be considered acceptable or even unavoidable, e.g., in distributed or integrated systems or when data come from unverified sources or are uncertain. In practice, inconsistencies are more of a rule than an exception in daily experience with DBMSs. Methods that cope with the presence of inconsistencies, commonly referred to as *paraconsistent*, are therefore highly in need.

We now discuss to which extent well-known techniques for integrity checking, that have been designed under the unrealistic premise that all data need to be consistent, can also be used in the presence of inconsistency. Their application, in fact, will gradually improve compliance of data with the integrity constraints.

Any simplification-based method for integrity checking is called inconsistency-tolerant if it guarantees preservation of satisfaction of instances of constraints that were satisfied before the update. To this end, we introduce the notion of case, to formalise the thought that integrity typically is violated not as a whole, but only by cases; satisfied cases of constraints can be separated from violated cases.

**Definition 3.1** *Let  $W$  be an integrity constraint.*

1. Each variable  $x$  in  $W$  that is  $\forall$ -quantified but not dominated by any  $\exists$  quantifier (i.e.,  $\exists$  does not occur left of the quantifier of  $x$  in  $W$ ) in the prenex normal form of  $W$  is called a global variable of  $W$ . Let  $\text{global}(W)$  denote the set of global variables in  $W$ .
2. The formula  $W\sigma$  is called a case of  $W$  if  $\sigma$  is a substitution such that  $\text{Range}(\sigma) \subseteq \text{global}(W)$  and  $\text{Image}(\sigma) \cap \text{global}(W) = \emptyset$ .

Note that cases of an integrity constraint need not be ground, and each constraint  $W$  as well as each variant of  $W$  is a case of  $W$ .

Now we can define inconsistency tolerance of integrity checking methods.

**Definition 3.2 (Inconsistency tolerance)** *Let  $\Gamma$  be an integrity theory,  $U$  an update and  $\Gamma'$  a finite set of cases of constraints in  $\Gamma$ .*

1. *A pre-test  $\Sigma$  (resp., post-test  $\Upsilon$ ) of  $\Gamma$  for  $U$  is inconsistency-tolerant whenever  $D^U \models \Gamma'$  if  $D \models \Sigma$  (resp.,  $D^U \models \Gamma'$  if  $D^U \models \Upsilon$ ) for all databases  $D$  consistent with  $\Gamma'$ .*
2. *An integrity checking method  $\mathcal{M}$  is inconsistency-tolerant if  $\Gamma'$  is satisfied in  $D^U$  if  $\mathcal{M}(D, \Gamma, U) = \text{satisfied}$  for all database  $D$  such that  $D \models \Gamma'$ .*

We first note that inconsistency tolerance is not enjoyed in general, which immediately limits our ambition of possibly characterising all conceivable methods as inconsistency-tolerant.

**Example 3.1** *Consider a database  $D$  containing the sole fact  $p(b)$ , and thus violating constraint  $W = \leftarrow p(X)$ . Case  $W' = \leftarrow p(a)$  of  $W$  is satisfied in  $D$ ; an update  $U$  inserting fact  $p(a)$  in  $D$  makes  $D^U$  violate  $W'$ . Integrity theory  $\Upsilon = \exists X(p(X) \wedge X \neq a)$  is a post-test of  $W$  for  $U$ , since, whenever  $W$  is satisfied in  $D$ ,  $\Upsilon$  evaluates to false in  $D^U$ , which correctly indicates that the update always violates integrity. With a similar argument we can conclude that  $\Sigma = \exists X p(X)$  is a pre-test of  $W$  for  $U$ . We have  $D \models W'$ ,  $D^U \not\models W'$ , i.e., the update violates case  $W'$ , but we also have  $D \models \Sigma$  and  $D^U \models \Upsilon$ , therefore neither  $\Sigma$  nor  $\Upsilon$  are inconsistency-tolerant.*

Inconsistency tolerance is, however, enjoyed by all plain pre-tests and post-tests.

**Proposition 3.3** *Let  $\Gamma$  be an integrity theory,  $U$  an update. Any  $\text{pre}_0^U(\Gamma)$  (resp.,  $\text{post}_0^U(\Gamma)$ ) is an inconsistency-tolerant pre-test (resp., post-test) of  $\Gamma$  for  $U$ .*

## 4 Analysis of inconsistency tolerance in concrete methods

The basic idea of Nicolas' method [17] is that a simplified form of the integrity theory imposed on the database is obtained from a given update and the current state of the database. The essence of the approach in [17] can be summarised as follows.

For a database  $D$ , an integrity constraint  $W$  in prenex conjunctive normal form and a tuple  $r$  to be inserted into some relational table  $R$ , Nicolas' simplification method automatically generates a simplification  $\Gamma_R^+ = W\gamma_1 \wedge \dots \wedge W\gamma_m$ , where the  $\gamma_i$  are substitutions obtained from unifiers

of  $r$  and  $m$  different occurrences of negated atoms in  $W$  that unify with  $r$ . More precisely, the  $\gamma_i$  are restrictions of  $\text{mgu}'_s$  of  $R$  with negated atoms in  $W$ , restricted to the global variables in  $W$ . (The procedure is symmetric for deletions.)

The main theorem in [17] states that if  $W$  is known to hold in the old state  $D$ , then  $W$  also holds in the new state  $D^U$  iff  $\Gamma_R^+$  holds in  $D^U$ . For our purposes of inconsistency-tolerant integrity checking, we are especially interested in the if-half.

Obviously, the requirement that the integrity constraint  $W$  be satisfied in the old state  $D$  is very central in this theorem. Now, we want to relax this premise. To simply drop it would not leave anything to look at. The intuition of what we are after is that there are only some violated instances of  $W$  (which can be found when evaluating all of  $W$ , not just a simplified form, against  $D$ ), while all of the “rest” of  $W$  is satisfied. And it is this big rest that we are interested in, hoping that the given update will not introduce more violated instances.

We can look formally at cases of  $W$  that are satisfied before the update. In fact, it is possible to show by tracing Nicolas' proof line by line that such cases of  $W$  will remain satisfied after the update if the simplified form of  $W$  evaluates to true.

**Theorem 4.1** *Let  $D$  be a database,  $W$  an integrity constraint,  $U$  the insertion of tuple  $r$  in relation  $R$ , and  $\Gamma_R^+$  the simplification of  $W$  wrt  $r$  generated by Nicolas' method. Let  $W'$  be a case of  $W$  such that  $D \models W'$ . Then,  $D^U \models W'$  if  $D^U \models \Gamma_R^+$ .*

Reconstructing the proof of Nicolas' simplification theorem in an inconsistency-tolerance perspective was possible, since the notion of case, albeit hidden, is already present in the original proof, which exhibits sets of substitutions to be applied to the integrity constraints.

The main simplification theorem in [14] and also its proof are formal generalisations of [17]. And in fact it is possible to show that also the method in [14] is inconsistency-tolerant, in the same sense as [17].

The method of [19] is based on a refinement of SLDNF-resolution, extended by some forward reasoning steps, by which it is possible to delimit the search space to be traversed to those parts of the union of database and integrity constraints that are actually affected by a given update. The procedure asserts that integrity remains satisfied in  $D^U$  if the resulting search space, using an element from  $U$  as top clause, is finitely failed; integrity is violated if the search space contains a refutation indicating inconsistency. The SLDNF procedure itself behaves “paraconsistently”, in that it does not take into account irrelevant clauses for refuting constraints. In fact, it is possible to show that the method of [19] is also inconsistency-tolerant in the sense of definition 3.2.

We conclude this section with a notable example of a method that does not comply with definition 3.2. Example 3.1 may seem artificial and, arguably, does not correspond to the output of any existing method. However, we have found out that the well-known method by Gupta et al. [10] is *not* inconsistency-tolerant. The integrity constraints considered by their method are of the form

$$(*) \quad \perp \leftarrow L \wedge R_1 \wedge \dots \wedge R_n \wedge C_1 \wedge \dots \wedge C_k$$

in which the head  $\perp$  is a predicate that, if derived, indicates inconsistency,  $L$  is a literal referring to a *local* (and thus accessible) predicate, the  $R_i$ 's are literals referring to *remote* predicates that cannot be accessed to check the integrity status of the database, while the  $C_j$ 's are arithmetic comparisons such that the variables occurring in them also occur in  $L$  or one of the  $R_i$ 's<sup>1</sup>; an update is an insertion of a tuple in  $L$ 's relation.

Their main result (theorem 5.2 in [10]) is based on the notion of *reduction* of a constraint: the reduction of a constraint  $W$  by tuple  $t$  inserted in  $L$ 's local predicate, written  $RED(t, L, W)$ , is obtained by substituting the components of  $t$  for the corresponding variables in  $L$ , and then eliminating  $L$ . Then to check whether a constraint  $W$  of the form (\*) is satisfied after the insertion of  $t$ , and assuming  $W$  was satisfied before the insertion, it suffices to check whether  $RED(t, L, W) \subseteq \cup_{s \text{ in } L} RED(s, L, W)$ , where  $\subseteq$  denotes query containment.

For example,  $W = \leftarrow l(X, Y) \wedge r(Z) \wedge X \leq Z \leq Y$  indicates that no  $Z$  in  $r$  may occur in an interval whose ends are specified by  $l$ . Suppose  $D = \{l(3, 6), l(5, 10)\}$  and  $U$  is the insertion of  $l(4, 8)$ ; then one can conclude that  $W$  is not violated in  $D^U$ , since

$$r(Z) \wedge 4 \leq Z \leq 8 \subseteq (r(Z) \wedge 3 \leq Z \leq 6) \cup (r(Z) \wedge 5 \leq Z \leq 10),$$

which holds basically since  $[4, 8] \subseteq [3, 10]$ .

To show that here we do not have inconsistency tolerance, consider a case  $W' = \leftarrow l(4, 8) \wedge r(Z) \wedge 4 \leq Z \leq 8$  of  $W$ , a database  $D = \{l(3, 6), l(5, 10), r(7)\}$  and the same update  $U$  as before. Clearly,  $W$  is violated in  $D$  whereas  $W'$  is satisfied. Again, the method guarantees that  $U$  cannot violate integrity provided that  $D$  has integrity (for the same containment as before), i.e.,  $\mathcal{M}(D, W, U) = \textit{satisfied}$ , where  $\mathcal{M}$  refers to the method of [10]. However, satisfaction of  $W'$  is not preserved in  $D^U$ , therefore the method of [10] is not inconsistency-tolerant.

## 5 Related work

Simplification has been recognised by a large body of research as a necessary technique for optimisation of integrity

<sup>1</sup>We omit other restrictions, not relevant for the present discussion, included in [10] for technical reasons.

checking. For each specific update or update pattern, a specialised integrity theory can be derived that guarantees consistency of the updated state, provided that the old state is also consistent. We have discussed approaches based on post-tests [17, 13] and pre-tests [11, 18, 5]. More generally, integrity checking is a special case of materialised view maintenance: constraints are defined as views that must always remain empty for the database to be consistent [9, 7].

Approaches based on logic programming such as [19], unlike classical logic, do not exhibit any explosive behaviour in the presence of inconsistency. In this respect, they may have been characterised as paraconsistent in a procedural sense, as done in [12]. However, to the best of our knowledge, the declarative inconsistency tolerance of simplifications for improving integrity checking, as investigated in this paper, has never been studied beforehand. We actually reckon that all the mentioned approaches can be reconsidered in terms of this declarative understanding of inconsistency tolerance (although, as shown, not all methods turn out to have this property). In [6], we are also studying the related problem of using the potential of inconsistency tolerance for controlling uncertain data, which, as such, should not necessarily be removed or prevented, as it is not clear whether they are bad or good. There, we focus on the dual of integrity satisfaction, i.e., violation, which may be more relevant in certain scenarios, and discuss soundness conditions for these (typically, while prevalence of satisfaction is concerned about sufficient conditions for ensuring preservation of satisfaction, making sure that integrity is violated may require additional effort).

The related problems of restoring integrity of a database once inconsistencies are detected (tackled since [2] with the notion of *repair*) and of using active rules for much the same purpose [4, 3], certainly give way to inconsistency tolerance, but cannot be directly used to detect inconsistencies for integrity checking purposes.

## 6 Conclusion

Intuitively, it seems unrealistic to assume that integrity in databases is always fully satisfied. This, however, is exactly the premise for virtually all known approaches to integrity checking. More precisely, all correctness results in the literature about more efficient ways to check integrity by evaluating simplifications of integrity constraints upon each update fundamentally rely on the requirement that integrity must be satisfied in the old state before the update. The unease about this intuitive conflict has motivated us to have a closer look into some well-known methods for integrity checking, to see if the consistency requirement could be relaxed somehow. On the basis of the notion of a “case” of an integrity constraint, we were able to generalise the main results of [17], [14], and [19].

We also observe that all of the performance gains obtained by such methods are inherited by their inconsistency-tolerant counterparts. The practical significance of these results is further demonstrated by the fact that, as experiments have shown, simplified integrity constraints execute faster than the original constraints and their execution time does not depend on the number of inconsistencies in the database. Besides, by applying an inconsistency-tolerant integrity checking method, the percentage of the data in a relational database that participate in inconsistencies will necessarily decrease in the new state if the update consists only of insertions, since the number of inconsistent cases cannot grow, while the total number of tuples increases. Although the same conclusion cannot be drawn for deletions, if deletions are distributed proportionally over consistent and inconsistent cases, in the long run, integrity will improve also in percentage. This is especially advantageous for the federation of databases, where, initially, there will be a fair amount of inconsistency (e.g., in a business taking over a former competitor). Inconsistency-tolerant integrity checking will automatically help making the database consistent over time, by reducing the percentage of inconsistency.

Future work will investigate in which sense also other methods for simplified integrity checking are inconsistency-tolerant, such as [15]. However, similar relaxations may be more elusive for methods that do not directly or indirectly employ a notion of case or substitution. Furthermore, as we have seen, inconsistency tolerance is not necessarily enjoyed by all methods and needs to be studied case by case, although we expect most methods to have this property.

**Acknowledgements** H. Decker is supported by Spanish grant TIC2003-09420-C02. D. Martinenghi is supported by the TONES IST project financed by the EU under contract number FP6-7603.

## References

- [1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [2] M. Arenas, L. E. Bertossi, and J. Chomicki. Querying database repairs using logic programs with exceptions. In *FQAS 2000*, pages 27–41. Physica-Verlag Heidelberg New York, A Springer-Verlag Company, 2000.
- [3] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Automatic generation of production rules for integrity maintenance. *ACM Transactions on Database Systems (TODS)*, 19(3):367–422, 1994.
- [4] S. Ceri and J. Widom. Deriving production rules for constraint maintainance. In D. McLeod, R. Sacks-Davis, and H.-J. Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 566–577. Morgan Kaufmann, 1990.
- [5] H. Christiansen and D. Martinenghi. On simplification of database integrity constraints. *Fundamenta Informaticae*, 2006. To appear. See [15].
- [6] H. Decker and D. Martinenghi. Integrity checking in inconsistent databases is not uncertain. *Submitted to The Second Twente Data Management Workshop - TDM'06 Uncertainty in Databases*.
- [7] G. Dong and J. Su. Incremental Maintenance of Recursive Views Using Relational Calculus/SQL. *SIGMOD Record*, 29(1):44–51, 2000.
- [8] J. Grant and J. Minker. Integrity constraints in knowledge based systems. In H. Adeli, editor, *Knowledge Engineering Vol II, Applications*, pages 1–25. McGraw-Hill, 1990.
- [9] A. Gupta and I. S. Mumick, editors. *Materialized views: techniques, implementations, and applications*. MIT Press, 1999.
- [10] A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 24-26, 1994, Minneapolis, Minnesota*, pages 45–55. ACM Press, 1994.
- [11] A. Hsu and T. Imielinski. Integrity checking for multiple updates. In S. B. Navathe, editor, *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 28-31, 1985*, pages 152–168. ACM Press, 1985.
- [12] R. A. Kowalski. *Logic for Problem Solving*. Elsevier, 1979.
- [13] M. Leuschel and D. de Schreye. Creating specialised integrity checks through partial evaluation of meta-interpreters. *Journal of Logic Programming*, 36(2):149–193, 1998.
- [14] J. W. Lloyd, L. Sonenberg, and R. W. Topor. Integrity constraint checking in stratified databases. *Journal of Logic Programming*, 4(4):331–343, 1987.
- [15] D. Martinenghi. *Advanced Techniques for Efficient Data Integrity Checking*. PhD thesis, Roskilde University, Denmark, in *Datalogiske Skrifter*, 105, <http://www.ruc.dk/dat/forskning/skrifter/DS105.pdf>, 2005.
- [16] D. Martinenghi, H. Christiansen, and H. Decker. Integrity checking and maintenance in relational and deductive databases, and beyond. In Z. Ma, editor, *Intelligent Databases: Technologies and Applications*, page to appear. Idea Group Publishing, 2006.
- [17] J.-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18:227–253, 1982.
- [18] X. Qian. An effective method for integrity constraint simplification. In *Proceedings of the Fourth International Conference on Data Engineering, February 1-5, 1988, Los Angeles, California, USA*, pages 338–345. IEEE Computer Society, 1988.
- [19] F. Sadri and R. Kowalski. A theorem-proving approach to database integrity. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 313–362. Morgan Kaufmann, Los Altos, CA, 1988.