INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians–
Universität
München

A Model-Theoretic Semantics of Integrity Constraints in Deductive Databases

Hendrik Decker

appeared in J. Dix, L. M. Pereira, T. Przymusinski (eds), *Proc. LPKR'97*,
Port Jefferson (New York), Univ. Koblenz, Institut f. Informatik, 1997
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1997-12, August 1997

A Model-Theoretic Semantics of Integrity Constraints in Deductive Databases

Hendrik Decker
Institut für Informatik
Universität München, Germany
hdecker@informatik.uni-muenchen.de

Abstract

We first skim some of the background problems associated to model-theoretic semantics and its relationship to proof theory. Then, we develop the notion of sustained models. Sustained models coincide with partial stable models. On the basis of sustained models, we develop a semantics for database integrity. Conventional semantics are two-valued, and hence do not explicitly consider states that neither satisfy nor violate integrity. In a sense, they are biased, by either paying more attention to satisfaction than to violation, or vice-versa, independent of any application. Our semantics is three-valued and not biased. As a refinement of the three-valued semantics, we specify at last a two-valued semantics which leaves it to the application to be more or less tolerant wrt satisfaction or violation of integrity.

Introduction

Proof theory and model theory of logic seem to have been in a continuous "love-hate" relationship; cf. the 'Historical Notes' in [Ro]. Robinson explains that a general theory of sets and hence of models is very intricate and demanding. It lacks the simplicity and certainty of evidence which is desirable for such a fundamental issue as semantics. On the other hand, what often spurs the interests of researchers in logic is its historical claim on truth and veracity. But it is this very claim which is put into question (and is ultimately renounced) by the school of logic formalists because of the shortcomings of model theory. On the other hand, logic as a naked formalism can be perceived as impoverished and degenerated to a mere glass bead game. For a critical assessment of model theory from a pragmatic point of view, cf. [Ko].

Trying to answer the question, "Why does it seem to be quite natural to think of models as the meaning of deductive databases?" is beyond the scope of this paper. However, a first response might be that relational databases (a special case of deductive databases which belongs to the roots of the field) are isomorphic to the (finite) models constituted by the database facts, which is simple and evident (cf. [Re1]). But things get more complicated when relational databases are enriched with more advanced concepts (e.g., recursive views, integrity constraints, indefinite knowledge, non-monotonic inference, or function symbols which may engender infinitely many terms). Still, models have remained to be of interest to the database community as a means to characterize semantics.

The current state of live-and-let-live between database logic and models is partly due to the following circumstances: First, almost all classes of models considered to be serious candidates for defining the semantics of deductive databases are Herbrand models, i.e. a distinguished kind of models whose universe does not exist as a (virtual or real) world outside of the database, but is constituted precisely of all ground terms ("the universe of discourse") in the formal language underlying the database. Second, the currently most successful model-theoretic (stable and well-founded) semantics can be seen as abstract proof theories, i.e. as non-constructive descriptions of (top-down or bottom-up-oriented) inference systems, and logic programming can be taken as one way to implement them.

Stable models [GL] are widely accepted as the intended semantics of those databases for which there is at least one stable model. For the common classes of definite, hierarchical, stratified [ABW] and locally stratified [Pr] databases, stable models coincide with the unique canonical (least Herbrand, standard, perfect) models of these classes. The main other contender for a model-theoretic semantics, well-founded models [VRS], can be characterized as a computationally interesting approximation of stable models (cf. [Ul]). In order to capture all databases that do not have a stable model, the partial stable model semantics has been proposed in [SZ]. It is shown in [KM2] to coincide with the argumentation-theoretic preferred extensions semantics [Du]. The latter has been applied also to sets of integrity constraints in [TK]. However, constraints are required to obey some syntactic restrictions in [TK], which often are not severe but occasionally are possibly disturbing. (For example, let r be a predicate the definition of which comprises a recursive cycle. Then, it is not clear which of the literals in the unique-key constraint $\leftarrow r(x, y) \,\&\, r(x, z) \,\&\, y \neq z$, expressing a functional dependency of the second argument of r on the first, should be designated as "retractible".)

The sustained model semantics presented in the first part of the paper is similar to the partial stable model semantics [SZ]. It is the basis for the definition of a new, three-valued semantics of integrity constraints, as presented in the second part of the paper. No syntactic restrictions which would go beyond the usual denial form requirement are imposed. An abductive logic programming procedure which soundly computes the sustained model semantics is described in [De5].

1 The sustained model semantics

We assume the reader understands what is meant by a Herbrand interpretation and a Herbrand model of a database, and how formulas are evaluated under a given interpretation (cf., e.g., [Ll]).

1.1 Partial interpretations

Interpretations are usually represented by subsets of the Herbrand base of the underlying language. Intuitively, each atom in an interpretation is assigned the value true, and each atom in the complement is assigned false. Interpretations which model the intended meaning of a database are fairly easy to obtain in case there is no negation in the body of clauses (cf. [vK]). However, the intended meaning of databases containing clauses with non-monotonic negation in the body is more difficult to capture. A third truth value, unknown, is convenient in order to make sense, in general, out of sets of clauses with negation. For example, assigning either true or false to p in $p \leftarrow \sim p$ makes no sense, as long as that clause is the only one about p in the database and thus describes the only possible way to derive p. However, assigning unknown to p captures fairly accurately what such a database would know about p. In general, negation ~ swaps truth values true and false, and the negation of a ground atom with truth value unknown is also unknown.

Abductive logic programming procedures such as [EK] [KM1] [D5] and others treat ground negative literals as quasi-positive literals. That is, the ~ concatenated with the predicate symbol of an atom is taken as a distinguished new "negative" predicate symbol. For emphasizing this, negated literals of form ~p(t) (say) are sometimes replaced by positive literals with a fresh predicate symbol that is used only to substitute each occurrence of ~p. In this paper, we simply speak of positive and negative predicates, and keep on distinguishing terminologically and notationally between positive and negative literals.

For convenience, we define some notational conventions: For a literal L, let |L| denote the atom of L; the *complement* $\overline{L}$ of L be ~L if L is positive, and |L| if L is negative. For a set S of clauses, let $\underline{S}$ be the set of all ground instances of clauses in S. For convenience, we call the set of integrity constraints associated to a database D at schema design time the *integrity requisite* of D. Integrity requisites are always supposed to be finite sets. For a database D with integrity requisite IC and underlying language $\mathcal{L}$, we denote with $\mathcal{LL}$ the language that comprises all of $\mathcal{L}$, and for each predicate p in $\mathcal{L}$, also a negative predicate symbol ~p. We distinguish the Herbrand base $\mathcal{H}$ of $\mathcal{L}$ from the Herbrand base $\mathcal{HH}$ of $\mathcal{LL}$, such that, for each ground atom A in $\mathcal{H}$, there is also a negative ground atom ~A in $\mathcal{HH}$.

Definition [SZ]

For a database with underlying language $\mathcal{L}$, a *partial interpretation* of $\mathcal{L}$ is an interpretation of $\mathcal{LL}$, i.e., a subset M of the Herbrand base of $\mathcal{LL}$ such that, for each literal L in M, $\overline{\mathsf{L}}$ is not in M. For convenience, the set of positive literals in M be denoted by M^+ and the set of negative literals in M by M^-.

Intuitively, a partial interpretation M explicitly tells which atoms, positive or negative, are assigned true; the complements of literals assigned true are assigned false; each literal L such that neither L nor $\overline{\mathsf{L}}$ is in M is assigned unknown. For convenience, we say, for a literal L in M, that L *is true in* M and $\overline{\mathsf{L}}$ *is false in* M; if neither L nor $\overline{\mathsf{L}}$ is in M, then we say that L *is unknown in* M. If no literal is unknown in M, then M is called a *total interpretation*. As usual, we may simply speak of (partial or total) interpretations of a database, instead of mentioning the underlying language explicitly. In general, it is reasonable to assume that the language may contain more symbols that those that actually occur in the database or its integrity requisite. However, for simplicity, we tacitly associate to each database the language which precisely consists of symbols occurring in the database.

Note that each *total* interpretation of $\mathcal{L}$ (which, for *each* atom A, contains either A or ~A) is also a partial interpretation of $\mathcal{L}$. However, not every subset of $\mathcal{LL}$ is a partial interpretation of $\mathcal{L}$, because the law of contradiction ("for each literal L in M, $\overline{\mathsf{L}}$ is not in M") excludes literals with the same atom but with opposed polarity in a partial interpretation. The law of contradiction could also be expressed on the theory level, by adding the (possibly infinite) set of integrity constraints of form ← p(X) & ~p(X) (where p is a predicate in $\mathcal{L}$, X is a vector of n distinct variables, and n is the arity of p) to the integrity requisite of the database. But we do not do that, for complying with the definition of the procedure in [D5], which would otherwise have to operate with such sets as candidate input clauses.

1.2 Sustained interpretations

The following definition is the basis for defining sustained models, later on.

Definition 1

A partial interpretation M of a database D is a *sustained interpretation* of D if there is a mapping λ from the Herbrand base of D to the set of natural numbers such that the following two points hold:

(1) For each literal L in M^+, there is a clause L ← B in $\underline{\mathsf{D}}$ such that each literal in B is true in M, and for each atom A of any literal in B, λ(A) < λ(L).

(2) For each literal L in M^- and each clause $\overline{\mathsf{L}}$ ← B in $\underline{\mathsf{D}}$, there is a literal in B which is false in M.

(1) is the key condition used for defining sustained models of disjunctive databases in [DC]. The first part of (1) is shaped after the definition of supported models in [ABW]. The mapping λ, which is only used in (1), is inspired by the level mappings in [Cv] (for defining acyclicity and also for redefining local stratifiability). Here, it ensures that the literal under consideration is derivable by a finite sequence of backward (top-down) reasoning steps from the literal to database facts. That corresponds to the minimality of supported models as required in [ABW]. In [SZ], minimality of derivable positive literals is achieved by a bottom-up-oriented foundedness condition. (2) rephrases the condition in the definition of partial models [SZ], i.e. partial interpretations that satisfy (2) are equivalent to the class of partial models. Moreover, the definitions of founded partial models [SZ] and sustained interpretations are equivalent.

Theorem 1

Let D be a database and M a partial interpretation of D.

a) M is a partial model of D if and only if (iff) M satisfies (2) above.

b) M is a sustained interpretation of D iff M is a founded partial model of D.

c) The well-founded model of D is a sustained interpretation of D.

Proof *(sketch)*

a) holds by definition, *c)* follows from *b)* and the result in [SZ] that well-founded models are founded partial models. (The converse does not hold: e.g., {p, ~q} is a sustained interpretation of $D_1 = \{p \leftarrow \sim q,\ q \leftarrow p\}$, but it is not well-founded.)

The idea for proving *b)* is that M^- can be equivalently defined by either (2) or a corresponding condition for partial models in [SZ], and M^+ can be equivalently defined by either the top-down-oriented definition of the level mapping in (1) or the bottom-up-oriented definition of the immediate consequence operator T applied to the "positive instantiation" of D [SZ]. A sketch of an elaboration of this idea follows:

only if: Let M be founded, let L be a positive literal in M. Suppose that, for each clause L<– B in $\underline{D}$, B was not true in M. Then show by induction on the least possible value of $\lambda(L)$ that L cannot be in the least fixpoint $T\uparrow\omega$ of T, which contradicts the assumption that M is founded.

if: Let M be a sustained interpretation. Suppose $T\uparrow\omega \neq M^+$. Then there is a literal L such that either L is in $T\uparrow\omega$ but not in M^+, or L is not in $T\uparrow\omega$ but is in M^+. Show by induction on either the fixpoint iteration or the value of $\lambda(L)$ that each of the two alternatives leads to a contradiction.

❏

Definition 2

Let M be a partial interpretation of a database D. M is a *sustained model* of D if M is a maximal sustained interpretation of D, i.e., there is no sustained interpretation M' of D which would properly contain M.

As stated in the theorem below, sustained models coincide with partial stable models. Therefore, we recall here that each database has at least one partial stable model, and hence a sustained model. But, in general, there can be several sustained models of a database, reflecting an ambiguity of predicate definitions in the database. Also, sustained models may not be total interpretations, reflecting that the database does not know the truth value of those literals that are assigned unknown. On the other hand, even if there are several sustained models which mutually contradict each other, these models may be total interpretations. For example, the sustained models of the database $D_1 = \{ p \leftarrow \sim q,\ q \leftarrow \sim p \}$ are $\{p, \sim q\}$ and $\{q, \sim p\}$. The unique sustained model of $D_2 = \{ p \leftarrow \sim p \}$ is $\{ \}$.

It can be shown that there never is a conflict between requirements (1) and (2) of sustained interpretations when they are maximized for obtaining a sustained model. That is, for a sustained interpretation M which is maximal wrt either (1) or (2), there is always a sustained model containing M. A maximality condition similar to the one above is also used in [SZ] for defining partial stable models. It can be shown that, for stratified databases, sustained models are minimal supported models; for locally stratified databases, sustained models are perfect models. Moreover, the following results hold. With theorem 1 and the main result in [KM2], proofs are straightforward.

Theorem 2

Let D be a database and M a partial interpretation of D.

a) M is a sustained model of D iff M is a partial stable model of D.

b) M is a sustained model of D iff M is a preferred extension of D.

With the preceding results, the question arises why we did not adopt the partial stable model semantics, instead of inventing an equivalent one. The reasons are rather subjective than objective: We have developed the beginnings of sustained models independently [DC], and in conjunction with a new approach to the semantics of database integrity (which is not dealt with in [SZ]), as presented in the next section. We also think that having developed a different way to define (an equivalent version of) partial stable models adds to their standing, rather than competing against it.

2 Integrity constraints

We first give an informal characterization of integrity constraints and define their syntax. Then we cast the well-known theoremhood and consistency views of database integrity (cf. [SK]) into the framework of sustained models. Both views have originally been defined in terms of the database completion semantics [Cl] [Ll], which unfortunately is not always consistent. As far as this author is aware, no analogue of theoremhood and consistency view in terms of a model-theoretic semantics has yet been studied in the literature. Our reformulation of both views in terms of sustained models serves as a reference for developing new refined views of integrity, as outlined in the abstract.

2.1 The purpose and the syntax of integrity constraints

Informally, (static) integrity constraints express properties required to hold of each database state. Procedurally speaking, integrity constraints help to prevent updates that would lead to semantically inconsistent states. Thus, integrity constraints are invariants across state changes. They limit the space of possible states that the database can ever be in. They may also be used to narrow the search space of possible answers, as known from semantic query optimization [CGM].

Rather than contributing positively to the definition of predicates, integrity constraints express what must *not* hold of them. Hence, we follow the usual convention to represent integrity constraints by *denials*, i.e. clauses with an empty head, signalling constraint violation in case the body is satisfied. For that and other purposes, the empty head is often replaced by a distinguished atom, e.g. false, violated, alarm, inconsistent etc, for indicating that something is wrong.

A more general syntax is conceivable. However, it is always possible to represent an arbitrary first-order sentence Σ in denial form: Σ can be represented by $\leftarrow neg_\Sigma$ (say), where neg_Σ is a fresh 0-ary predicate, defined by the "general clause" $neg_\Sigma \leftarrow \sim\Sigma$. That then is transformed into a set of normal database clauses for defining neg_Σ, according to the equivalence-preserving transformation in [LT]. This definition of neg_Σ is at last added to the database. (In many cases, the [LT] transformation yields floundering queries, but it never does if Σ is range-restricted or satisfies some even less limiting properties [De4], and if the [LT] transformation is applied to the range form of Σ [D1,3], instead of the original form of Σ; the range form of Σ is equivalent to Σ but renders it evaluable; cf. also [VT].)

Dynamic integrity constraints are requisites which limit the admissibility of states relative to other states. They can be transformed to static ones by amalgamating database states into the object language (cf. [BK]). However, for simplicity, dynamic integrity constraints are not dealt with in this paper, and amalgamation of meta-data with object data is simply taken for granted.

2.2 Two-valued semantics of integrity constraints

All semantics for database integrity in the literature of which this author is aware are two-valued, in the sense that constraints may either be satisfied or violated, but nothing else. Roughly, the usual intuition is that an integrity constraint is satisfied if it is true of the database, and violated if it is false.

Somewhat subtler definitions are used in the theoremhood view semantics in [LST], and in the consistency view of [SK]. An integrity constraint is satisfied in [LST] if it is true of the database completion, and violated otherwise, which includes the possibility that it is neither true nor false. Conversely, a constraint is violated in [SK] if its body is true of the completion, and satisfied otherwise, which again includes the possibility that it is neither true nor false. Thus, as observed in [D2] [Re2], the same integrity constraint may have a radically different meaning in the same database by different semantics. For instance, in the database $D_1 = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$, the constraint $\leftarrow p$ is satisfied according to consistency, but violated according to theoremhood. An example where the semantics for integrity is neither well-defined in the theoremhood nor in the consistency view, in as far as the database completion is inconsistent, is $D_2 = \{p \leftarrow {\sim}p\}$ with requisite $\{\leftarrow p\}$.

For further reference to the distinction of theoremhood and consistency view, we now recast these notions by replacing "truth in all models of the completion" with "truth in all sustained models". However, no matter if the completion or sustained models are adopted as the underlying semantics, we can observe: In general, the notions of integrity satisfaction according to theoremhood and violation according to consistency are not complementary. They are if and only if there is a unique two-valued interpretation which models the semantics of the database.

A significant difference of theoremhood and consistency view is that, effectively, the former pays more attention to satisfaction, the latter to violation. The respective opposite notion is defined only indirectly, as the negation of the class of cases which fall under the definition of the notion to which higher attention is paid. Thus, it can be argued that each of these propensities corresponds in its own way to the well-known prevalence of the positive over the negative in logic.

In general, each two-valued approach to define the semantics of database integrity in the literature either favors satisfaction or violation, by being definitive on satisfaction or, resp., violation, while merely subsuming the opposite notion as the complement. However, it seems one-sided to take a quasi constructive stand on what it means that integrity is violated (or satisfied), on one side, without being equally differentiating wrt the rest. Rather, it should be left to the application to pay more or less attention to either satisfaction or violation.

In general, it seems hard to avoid difficulties of assigning an appropriate meaning to integrity requisites, as long as the semantics remains two-valued and takes sides wrt either satisfaction or violation. For instance, consider $D_2 = \{p \leftarrow {\sim}p\}$

with requisite $IC = \{\leftarrow p, \leftarrow \sim p\}$. (The non-stratified D_2 may seem contrived. However, predicate definitions which recursively go through negation may have a practical meaning; e.g., the clause of the so-called vanilla interpreter for interpreting negative ground literals, $demo(\sim A) \leftarrow \sim demo(A)$, is of that form; other examples are in [To].) None of the constraints in IC is violated, since neither p nor ~p is true in D_2. However, it seems daring to call integrity satisfied in this example, as a violation-prevalent semantics such as [TK] would. To call integrity satisfied in this example is questionable also because IC might easily be considered unsatisfiable, independent of the database (cf. [BDM] [DTU]). Anyway, the sustained model { } of D_2 gives an acceptable meaning to D_2 by assigning unknown to each literal, which suggest that also each of the constraints in IC should be considered unknown, i.e. neither satisfied nor violated.

2.3 A three-valued view of integrity

In the framework of a general theory of deductive databases, it is desirable to have a general, application-independent semantics of integrity constraints. The discussion in the preceding subsection suggests that such a semantics should not be biased, but should be equally sharp wrt satisfaction and violation. Thus, it is convenient to acknowledge the existence of a third truth value (or a third property between the derivability of a formula and of its negation) also for integrity constraints. Intuitively, we want to say that a constraint I is *unknown* if I is neither satisfied nor violated in the database. With the sustained model semantics, we can be more precise:

Definition 3

Let D be a database, IC its integrity requisite and $I = \leftarrow B$ an integrity constraint.

a) I is *satisfied in* D if, for each sustained model M of D, there is a literal in B which is false in M. We then also say that I *is satisfied in* M.

IC is *satisfied in* D if, for each sustained model M of D, each constraint in IC is satisfied in M.

b) I is *violated in* D if, for each sustained model M of D, each literal in B is true in M. We then also say that I *is violated in* M.

IC is *violated in* D if, for each sustained model M of D, there is a constraint in IC which is violated in M.

c) I is *unknown in* D if I is neither satisfied nor violated in D.

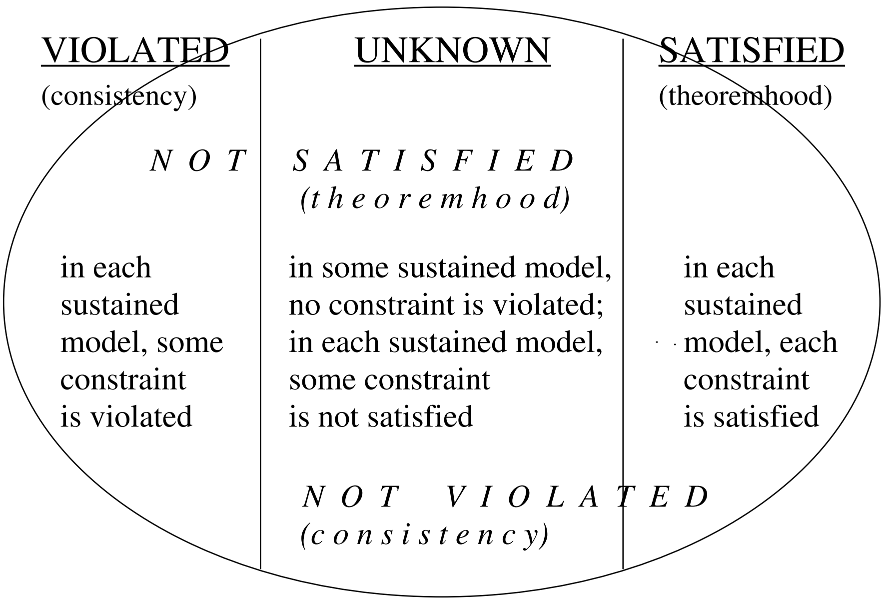
IC is *unknown in* D if there is a constraint in IC which is unknown in D.

Although the notions of satisfaction and violation in definition 3 are of opposed polarity, they are obviously not complementary. Satisfaction is defined in the spirit of the theoremhood view, and violation in the spirit of the consistency view. Since the notions are not complementary, they leave a non-empty, "unkown" space between satisfaction and violation of integrity. For example, each constraint in $\{\leftarrow p, \leftarrow \sim p\}$ and hence integrity as a whole is unknown in $D_2 = \{p \leftarrow \sim p\}$. Also, $\{\leftarrow p\}$ is unknown in

$$D_1 = \{p \leftarrow \sim q,\ q \leftarrow \sim p\}.$$

The sustained model $\{q, \sim p\}$ (which, in this case, is one of the two stable models) of D_1 prevents the constraint $\leftarrow p$ from firing, and hence it is not violated. Nor is $\leftarrow p$ satisfied, because it is violated in the sustained model $\{p, \sim q\}$ (the other stable model of D_1). Similarly, also the requisite $\{\leftarrow q\}$ is unknown in D_1, and so is $\{\leftarrow \sim q\}$, and also $\{\leftarrow \sim p\}$. However, $\{\leftarrow p, \leftarrow q\}$ is violated in D_1, and also $\{\leftarrow p, \leftarrow \sim p\}$ is violated in D_1.

The relationship between theoremhood, consistency and the three-valued view of integrity can be pictured as shown below. Constraint satisfaction and violation is understood according to definition 3. That is, theoremhood and consistency view are cast into the framework of sustained models, rather than confining them to the completion semantics, their original basis. Theoremhood view (on the right hand side) and consistency view (on the left) conflict in the "unknown" field between violation according to consistency and satisfaction according to theoremhood.



2.4 A two-valued refinement of the three-valued view

The following theorem can be shown by applying definition 3.

Theorem 3

Let D be a database and I an integrity constraint. Then, I is unknown in D if and only if one of the following two points holds. (It can be shown that they are exclusive, i.e., both cannot hold at a time.)

(*) There is a sustained model M of D such that I is neither satisfied nor violated in M, or

(**) there are two sustained models M, M' of D such that I is satisfied in M and violated in M'.

Theorem 3 suggests that it might make sense to distinguish between two kinds of "unknown" integrity: One where there exists a sustained model in which integrity is satisfied (e.g., $\{\leftarrow p\}$ in $D_1 = \{p \leftarrow \sim q,\ q \leftarrow \sim p\}$), and the other where there does not exist such a sustained model (e.g., $\{\leftarrow p\}$ in $D_2 = \{p \leftarrow \sim p\}$).

According to point (**) of theorem 3, it seems quite reasonable to relax the demands on integrity satisfaction, and consider a requisite weakly satisfied if there is one sustained model in which each of its constraints is satisfied. We should think that, for many applications, taking sides in this manner is fair, since the idea of integrity requisites is to constrain the space of admissible states (interpretations) of a database. For some applications, the demands on satisfaction can possibly be relaxed even further, according to point (*) above: If there is any admissible interpretation which does not violate integrity, then such an interpretation should be accepted as possibly intended, rather than be rejected, as by the theoremhood view. This relaxed point of view toward integrity satisfaction is taken, e.g., in [TK].

However, such a point of view might be too liberal, for applications where integrity requisites are meant to be strong. In the case that no sustained model which satisfies each constraint exists, it may seem more appropriate to relax the notion of violation (rather than satisfaction), by considering a requisite weakly violated if there is no sustained model to satisfy it. This motivates the following definition.

Definition 4

Let D be a database, IC its integrity requisite and I an integrity constraint.

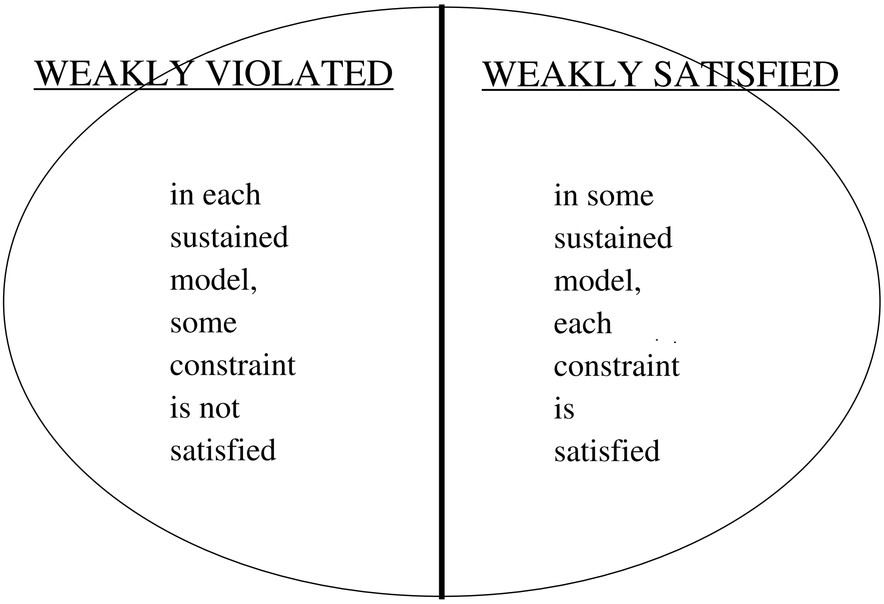
a) I is *weakly satisfied in* D if there is a sustained model M of D such that I is satisfied in M.

IC is *weakly satisfied in* D if there is a sustained model M of D such that each constraint in IC is satisfied in M.

b) I is *weakly violated in* D if, for each sustained model M of D, I is not satisfied in M.

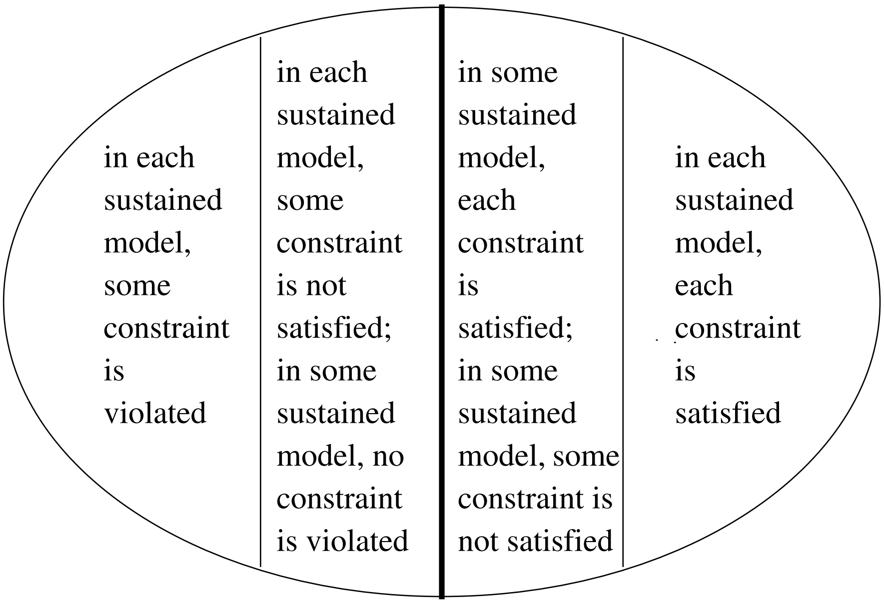
IC is *weakly violated in* D if, for each sustained model M of D, there is a constraint in IC which is not satisfied in M.

In accordance with the diagram above, the weakened notions of satisfaction and violation of integrity can be pictured as follows.



For example, $\{\leftarrow p\}$ is weakly satisfied in $D_1 = \{p \leftarrow \sim q,\ q \leftarrow \sim p\}$ and weakly violated in $D_2 = \{p \leftarrow \sim p\}$. In general, it is easy to show that weak satisfaction and weak violation of integrity generalizes satisfaction and, resp., violation as in definition 3. Notice that weak satisfaction and weak violation are symmetric and complementary. Thus, definition 4 yields a two-valued semantics for integrity which does not take sides. Rather, it splits up the class of cases called "unknown", before, into two symmetric, complementary parts. One of them enlarges the class of cases termed satisfied, the other enlarges the class of cases termed violated.

Note that weak satisfaction and weak violation do not contradict the three-valued view of integrity according to definition 3. Rather, they just refine it, by differentiating the class of databases and requisites where integrity is unknown. The refinement can be pictured by blending the previous two diagrams, as follows.



The following theorem can be shown by applying definitions 3 and 4.

Theorem 4

Let D be a database, IC its integrity requisite and I an integrity constraint. Then, the following points hold.

(*i*) I is weakly satisfied in D if I is satisfied in D.

(*ii*) IC is weakly satisfied in D if IC is satisfied in D.

(*iii*) I is weakly violated in D if I is violated in D.

(*i*v) IC is weakly violated in D if IC is violated in D.

(*v*) I is weakly satisfied in D iff I is not weakly violated in D.

(*vi*) IC is weakly satisfied in D iff IC is not weakly violated in D.

The diagram above illustrates that our refined view of integrity captures different degrees of violation or satisfaction of integrity. Thus, a more or less permissive stand on violation or satisfaction can be taken, according to the needs of a given application. The same decision procedure can be used for implementing the different points of view. The decisions to be taken are whether a constraint is satisfied or violated in some or each sustained model of a database. For a query or an update request, the procedure in [De5] computes answers and sets of assumptions which explain the answers. Each computed set of assumptions that explains a computed answer is true in some sustained model in which no constraint is violated.

Conclusion

We have developed the sustained model semantics for query answering and integrity checking in deductive databases. Several points have motivated this endeavor.

First, it appeared as disturbing that, while most of today's semantics for query answering are three-valued, most semantics in the literature for answering the query if integrity is satisfied or violated are two-valued. After all, the answer "unknown" may make as much sense to the query about integrity as to any other query. Hence, we have developed a three-valued view of integrity which concedes that integrity may be neither violated nor satisfied in certain cases.

Second, we found that most semantics for integrity are, in a sense, one-sided, by paying a higher attention to either satisfaction or violation. We have argued that an application-independent semantics for integrity should not be one-sided wrt conditions that ensure satisfaction (or, resp., violation) while being less concerned wrt the opposite. Rather, it should be left to the particular requirements of a given application to relax positions, for taking sides wrt weaker or stronger notions of satisfaction or violation of integrity, if necessary. This led to a refinement of our three-valued view, resulting in a non-biased two-valued semantics. The four possible cases that an integrity constraint is either satisfied by each sustained model, or only weakly satisfied by some sustained model, or that it is neither satisfied nor violated (hence weakly violated), or that it is violated in all sustained models, might even suggest to investigate a four-valued semantics for integrity.

Third, there was a need to clarify the relationships between the traditional theoremhood and consistency views of integrity [SK] and more recent proposals concerning database integrity in abductive logic programming [KM1] [KKT] [TK]. That need arose in the context of ongoing work on using abductive logic programming for knowledge assimilation in deductive databases [De5−8].

To repeat, the starting point of this paper was a somehow uneasy feeling wrt the subtle diversity of existing proposals for capturing the meaning of integrity constraints. This paper is not the first in this regard. Similar observations have been made in [Re2]. However, the conclusions drawn in [Re2] lead into a direction which is different from ours. Reiter proposes to make the epistemic meta-level character of integrity constraints explicit by using a first-order calculus enhanced with modal operators. As opposed to that, my goal is to see how far model theory together with abductive logic programming can carry. A further refinement of our notions of integrity satisfaction and violation is proposed in [D7].

Acknowledgements

François Bry hosted me as a visitor at the computer science department of the University of München, where this paper was written. I also appreciate the feedback received from reviewers.

References

[ABW] Apt, Blair, Walker: Towards a theory of declarative knowledge, in [Mi], 1988.

[BDM] Bry, Decker, Manthey: A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases, Proc. EDBT, Springer, 1988.

[BK] Bowen, Kowalski: Amalgamating language and metalanguage in logic programming, in Clark, Tärnlund (eds), Logic Programming, Academic Press, 1982.

[CGM] Chakravarthy, Grant, Minker: Logic-based approach to semantic query optimization, ACM Transactions on Database Systems, 1990.

[Cl] Clark: Negation as failure, in Gallaire, Minker (eds): Logic and Data Bases, Plenum Press, 1978.

[Cv] Cavedon: Acyclic logic programs and the completeness of SLDNF resolution, Theoretical Computer Science 86, 1991.

[D1] Decker: Integrity enforcement on deductive databases, in Kerschberg (ed): Expert Database Systems, Morgan Kaufmann, 1987.

[D2] Decker: Three views on integrity checking, internal note, ECRC, 1987, revised 1989.

[D3] Decker: The range form of databases and queries, or: How to avoid floundering, Proc. 5. ÖGAI, Springer Informatik-Fachberichte 208, 1989.

[D4] Decker: On generalized cover axioms, Proc. 8th ICLP, MIT Press, 1991.

[D5] Decker: An extension of SLD by abduction and integrity maintenance for view updating in deductive databases, Proc. JICSLP'96, MIT Press, 1996.

[D6] Decker: On knowledge assimilation in deductive databases, Draft, http://www.pms.informatik.uni-muenchen.de, 1997.

[D7] Decker: Toward a paraconsistent semantics of database integrity, http://www.pms.informatik.uni-muenchen.de, presented at the 1st World Congress on Paraconsistency, Ghent (Belgium), 30 July – 2 August 1997.

[D8] Decker: Abduction for knowledge assimlation in deductive databases, to appear in Proc. 17th Int'l Conf. of the Chilean Computer Society, IEEE Press, November 1997.

[DC] Decker, Casamayor: Sustained models and sustained answers in first-order databases, Proc. Joint Conference GULP-PRODE'94 on Declarative Programming, Vol. 2, Univ. Politec. Valencia (Spain), 1994.

[DTU] Decker, Teniente, Urpí: How to tackle schema validation by view updating, Proc. 5th EDBT, Springer, 1996.

[Du] Dung: An argumentation-theoretic foundation for logic programming, J. Logic Programming 22, 1995.

[EK] Eshghi, Kowalski: Abduction compared with negation by failure, Proc. 7th ICLP, MIT Press, 1989.

[GL] Gelfond, Lifschitz: The stable model semantics for logic programming, Proc. 5th ICLP, MIT Press, 1988.

[KKT] Kakas, Kowalski, Toni: The role of abduction in logic programming, in Handbook of Logic in Artificial Intelligence and Logic Programming, Oxford University Press, 1995.

[KM1] Kakas, Mancarella: Database updates through abduction, Proc. 16th VLDB, 1990;

[KM2] Kakas, Mancarella: Preferred extensions are partial stable models, J. Logic Programming 14, 1992.

[Ko] Kowalski: Logic without models, Imperial College, 1993. http://laotzu.doc.ic.ac.uk/UserPages/staff/rak/rak.html

[Ll] Lloyd: Foundations of Logic Programming, Springer, 1987.

[LST] Lloyd, Sonenberg, Topor: Integrity constraint checking in stratified databases, J. Logic Programming 4, 1987.

[LT] Lloyd, Topor: Making Prolog more expressive, J. Logic Programming 3, 1984.

[Mi] Minker (ed): Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, 1988.

[Pr] Przymusinski: On the declarative semantics of deductive databases and logic programs; in [Mi], 1988.

[Re1] Reiter: Towards a logical reconstruction of relational database theory, in Brodie, Mylopoulos, Schmidt (eds), On Conceptual Modelling, Springer, 1984.

[Re2] Reiter: What should a database know?, J. Logic Programming 14, 1992.

[Ro] Robinson: Logic: Form and Function, The Mechanization of Deductive Reasoning, Edinburgh University Press, 1979.

[SK] Sadri, Kowalski: A theorem-proving approach to database integrity, in [Mi], 1988.

[SZ] Saccà, Zaniolo: Stable models and non-determinism in logic programs with negation, Proc. PoDS'90, 205-217, 1990.

[TK] Toni, Kowalski: Reduction of abductive logic programs to normal logic programs, Proc. 12th ICLP, 1995.

[To] Torres: Negation as failure to support, Proc. 2nd Int'l Workshop Logic Programming and Non-monotonic Reasoning, MIT Press, 1993.

[Ul] Ullman: Assigning an appropriate meaning to database logic with negation, Stanford University, 1996.

[vK] van Emden, Kowalski: The semantics of predicate logic as a programming language, J.ACM 23, 733-742, 1976.

[VRS] Van Gelder, Ross, Schlipf: Unfounded sets and well-founded semantics for general logic programs. Proc. PoDS, 1988.

[VT] Van Gelder, Topor: Safety and translation of relational calculus queries, ACM Transactions on Database Systems 16, 1991.