References

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

[De] Decker: A model-theoretic semantics of integrity constraints in deductive databases, to appear in Proc. Workshop Logic Programming & Knowledge Representation (LPKR'97), Long Island, NY, October 1997.

[D2] Decker: One abductive logic programming procedure for two kinds of updates, to appear in Proc. Workshop DYNAMICS'97, Long Island, NY, October 1997.

[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.

[Fi] Fitting: A Kripke-Kleene semantics for logic programs, J. Logic Programming 2, 1985.

[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.

[Ku] Kunen: Negation in logic programming, J. Logic Programming 4, 1987.

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

[R1] Reiter: On closed world data bases, in Gallaire, Minker (eds), Logic and Data Bases, Plenum Press, 1978.

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

[SK] Sadri, Kowalski: A theorem-proving approach to database integrity, in Minker (ed): Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, 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.

[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.

If, for example, the designated model which determines the semantics of choice for a given application is the well-founded model, then, for $IC = \{\leftarrow p, \leftarrow q\}$, integrity is satisfied in $D_1 = \{p \leftarrow \sim q,\ q \leftarrow \sim p\}$, since both p and q are unknown in the well-founded model $\{\ \}$ of D_1. Even the set $IC' = \{\leftarrow p, \leftarrow \sim p\}$ (which would be violated in classical logic, or if the law of excluded middle $p \vee \sim p$ was enforced) is not violated in D_1 according to the well-founded model semantics. Notice that the constraint $\leftarrow p \,\&\, \sim p$, i.e., the law of contradiction, remains satisfied. If, however, the semantics is given by the partial stable models, then in each designated model of D_1, one of the constraints in IC and also one in IC' is violated (the partial stable models of D_1 are $\{p\}$ and $\{q\}$).

Proposition

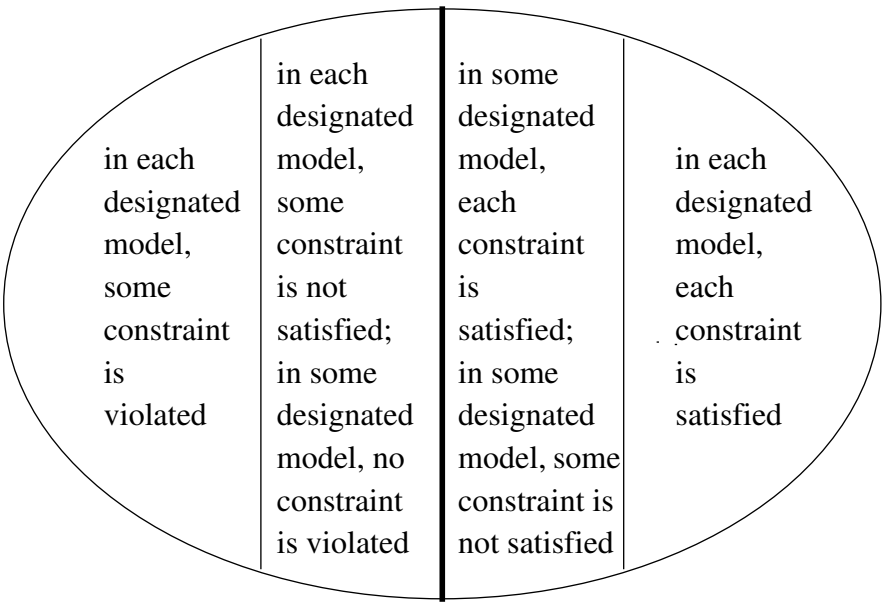
Let D be a database and I an integrity constraint. Then, I is unknown in D if and only if one of the following two points holds. (It can be shown that they are exclusive, i.e., both cannot hold at a time.)

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

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

The proposition above suggests that it makes sense to distinguish between two kinds of "unknown" integrity: One where there exists a designated 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 designated model (e.g., $\{\leftarrow p\}$ in $D_2 = \{p \leftarrow \sim p\}$).

According to point (**), it seems quite reasonable to relax the demands on integrity satisfaction, and consider a set of integrity constraints satisfied if there is one designated model in which each of its elements is satisfied. We should think that, for many applications, taking sides in this manner is fair, since the idea of integrity constraints is to constrain the space of admissible states of a database. For some applications, the demands on satisfaction can possibly be relaxed even further, according to point (*): If there is a designated model 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]. The diagram below corresponds to such a differentiated view of integrity satisfaction and violation.

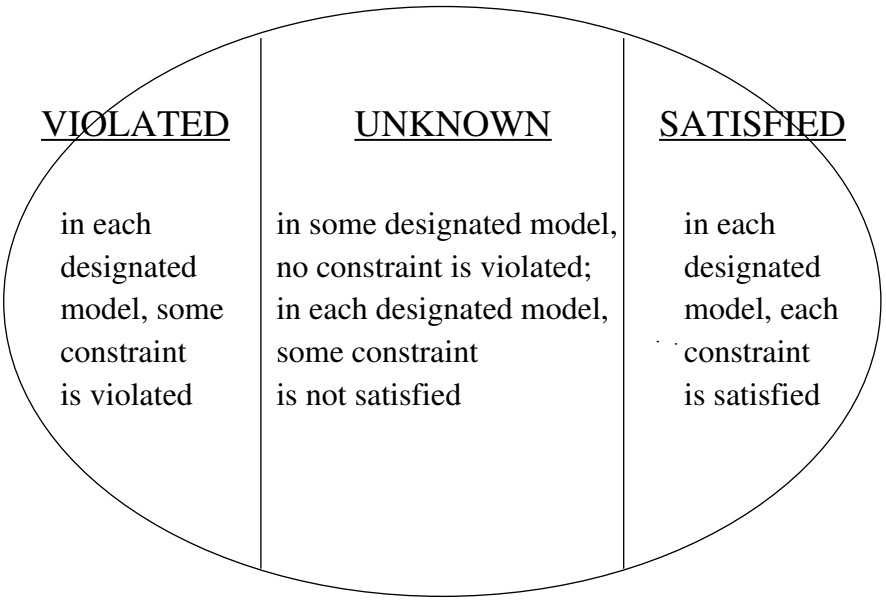


Although the notions of satisfaction and violation in the definition above are of opposed polarity, they are obviously not complementary. Satisfaction is defined in the spirit of the theoremhood view, and violation in the spirit of the consistency view. Since the notions are not complementary, they leave a non-empty, "unknown" space between satisfaction and violation of integrity. For example, if well-founded or partial stable models designate the semantics, then 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 partial stable model $\{q, \sim p\}$ (which, in this case, is even a stable model) 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 partial stable model $\{p, \sim q\}$ (the other stable model of D_1). Similarly, also the integrity constraint $\leftarrow q$ is unknown in D_1, and so is $\leftarrow \sim q$, and also $\leftarrow \sim p$. On the other hand, $\{\leftarrow p, \leftarrow q\}$ is violated in D_1, and also $\{\leftarrow p, \leftarrow \sim p\}$ is violated in D_1, in terms of partial stable models; each of these constraints is unknown in D_1 in terms of well-founded models.

The following diagram illustrates the definition above.



The following result can be shown by applying the definition.

with a three-valued semantics may return three kinds of answers: Positive ones, negative ones and those that return a u-value. Thus, there seems to be no reason to not go all the way and also consider databases in which the state of integrity is unknown because one of the constraints evaluates to u, or is assigned u by the semantics of choice.

In [D2], a three-valued semantics of integrity constraints has been proposed. It is based on sustaied models, which, for DDBs without integrity constraints, are equivalent to the partial stable model semantics. In this paper, we also are going to consider u as a possible third truth value of integrity constraints. However, we stay on a more abstract level and are less specific as to which particular three-valued semantics of DDBs is applied not only to the database, but also to its constraints. Besides partial stable models, candidates are, e.g., well-founded models, a Kunen-Fitting kind of semantics, etc. In the remainder, we are going to speak of designated models as those which determine the truth values of answers to closed queries and constraints. Thus, the designated models may be either the well-founded or the partial stable models, the Fitting or the Kunen models, etc.

With the following definition, we can be more precise.

Definition

Let D be a database, $I = \leftarrow B$ an integrity constraint and IC a set of integrity constraints.

a) I is *satisfied in* D if, for each designated 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 designated model M of D, each constraint in IC is satisfied in M.

b) I is *violated in* D if, for each designated 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 designated 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, i.e., if there is a designated model M of D such that no literal in B is false in M and at least one literal in B is unknown in M.

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

A weaker, more tolerant notion of integrity (which is analogous to the *consistency view* in [SK]) only requires the existence of a single partial stable model such that each constraint is true in it. For instance, the database $D_1 = \{p \leftarrow \sim q,\ q \leftarrow \sim p\}$ satisfies the integrity constraint $I = \leftarrow \sim p$ (which denies that $\sim p$ could hold) according to the consistency view, since $\{p, \sim q\}$ is a stable model of D_1 in which I is true; however, I is not true in the second stable model $\{q, \sim p\}$ of D_1, hence I is not satisfied in D_1 in terms of theoremhood.

In general, though, our analogons of theoremhood and consistency view coincide in the (frequent) case that there is a unique stable model of the database. However, whenever there are several stable models or several partial stable models, the possibility arises that some constraint may be neither satisfied nor violated in some or all of the models, because they evaluate to u.

Hence, an even weaker, thus even more tolerant notion concedes satisfaction of integrity already if none of the constraints evaluates to false in any of the partial stable models. For example, consider $D_2 = \{p \leftarrow \sim p\}$ (cf. footnote), which is not inconsistent in the sense that the partial stable model { } (the empty set of ground literals of positive and negative polarity) of D_2 considers both p and $\sim p$ unknown, and $I = \leftarrow p$ is neither true nor false in { }. Thus, I can be considered not violated and hence satisfied as long as nothing more precise about p is known in D_2. Similarly, also $I' = \leftarrow \sim p$ is neither true nor false in { }. *(*An instance of a database of form D_2 above is the well-known barber's paradox, which can be expressed by the clause $\text{shaves(barber, x)} \leftarrow \sim\text{shaves(x, x)}$, i.e., "the barber shaves each individual x who does not shave himself".)

One may hesitate to tolerate both constraints I and I' at a time, since $I'' = \{\leftarrow p, \leftarrow \sim p\}$ appears to be self-contradictory. However, it is arguable to consider I'' not violated as long as nothing more precise is known about p, in terms of partial stable models of the database. Thus, I'' would not be violated in $\{p \leftarrow \sim p\}$, but it would be in the empty database Ø, since $\sim p$ is true in the preferred extension of Ø and hence violates the constraint $\leftarrow \sim p$ in I''.

A three-valued semantics of integrity constraints

As opposed to the widespread use of a third truth value for defining the semantics of DDBs, the meaning of integrity constraints is usually defined two-valued. Either an integrity constraint is satisfied or it is violated. In most definitions, there is no in-between, i.e. there is no notion of a database state in which an integrity constraint is neither satisfied nor violated. That is worrisome, because the syntax of integrity constraints is that of queries. Moreover, integrity constraints are usually evaluated as closed queries with a *yes/no* answer, such that one answer signalizes satisfaction, the other violation. On the other hand, queries in deductive databases

The purpose of integrity constraints

From a technical point of view, integrity constraints in DDBs serve to constrain the inferability of positive conclusions and to check whether updates of the theory introduce inconsistency. In other words, integrity constraints serve to narrow the search space of admissible solutions to problems posed as queries or update requests. From a conceptual point of view, the world modeled by a Horn theory can be described more accurately by making use of constraints. More precisely, an integrity constraint is a closed formula in denial form which is required to be *satisfied* in each state of the associated database. If any one of the constraints of a database is not satisfied, then integrity is said to be *violated*.

In [EK] [KM1] [Du] [KKT] [De], also the consistency of abductively inferred hypotheses is defined on the level of associated integrity constraints, by denials of the form $\leftarrow p(x1,...,xn)$ & $\sim p(x1,...,xn)$, for each predicate p in the language, where $n \geq 0$ is the arity of p, & is conjunction, ~ is negation and $x1,...,xn$ are distinguished variable symbols. Such denials express the classical law of contradiction. In [EK] [KM1] but not in [Du] nor [De], also the law of excluded middle is implicitly assumed as a set of integrity constraints of form $p(x1,...,xn) \vee \sim p(x1,...,xn)$. Because of the non-classical use of "$\leftarrow$" in LP, the latter are not equivalent to the constraints for expressing the law of contradiction.

Because of the parconsistent behavior of NLP and ALP inference rules, there is no inconsistency in any database state if there are no integrity constraints. Thus, the purpose of integrity constraints is to model explicitly what should and what should not be considered an inconistent database state. In the remainder, we are going to see that the third truth value of the semantics of DDBs allows to capture various degrees of inconsistency, in terms of the satisfaction or violation of integrity constraints.

The semantics of integrity constraints

There is an ongoing debate about the adequacy of various concepts of what it could or should mean that a database satisfies or violates its integrity constraints (cf., e.g., [R2]). According to the *theoremhood view* (cf. [SK]), integrity constraints are supposed to state properties that must be true (i.e., logical consequences) of the theory embodied by the database. In order to have a declarative understanding of the meaning of a database theory which is independent of the (usually incomplete) way that truth is inferred from the database, a non-procedural semantics is desirable. The declarative semantics of DDBs (pure logic programs without integrity constraints, except the law of contradiction) has been described in terms of "preferred extensions" [Du], or, equivalently, as partial stable models (cf. [KM2]). Analogous to the theoremhood view, integrity is then considered satisfied if and only if each constraint is true in each partial stable model of the database.

line [Fi, Ku] has generalized *comp* by assigning a paraconsistent semantics to databases, the completion of which had heretofore been considered inconsistent. While stable models (more generally, partial stable models [SZ]) strive to capture what is *intended*, *comp* reflects what is *computed* by SLDNF. Well-founded models can be computed by common database query answering procedures, and have come to be understood as tractable approximations of (partial) stable models.

Both lines of development make use of a third truth value, undefined or unknown (u), in order to capture situations where the usual two-valued interpretations fail to work consistently. The meaning of the u-value of model-oriented and procedural semantics differ. The u-value of model-oriented semantics expresses that particular database facts are not well-defined and can neither be answered *yes* nor *no* when queried. The u-value in the semantics of [Fi] [Ku] reflects that SLDNF never terminates properly to search for a *yes*/*no* answer when querying a fact with truth value u; however, the intended meaning of the definition of that fact may well be one of the two standard truth values true and false.

A more recent approach to the semantics of DDBs is related to an abductive proof procedure [EK] which reasons hypothetically with negated ("abducible") facts. It improves SLDNF by terminating for larger classes of queries and databases, as well as by providing explanations for computed answers. The procedure of [EK] computes the partial stable model semantics, which has been described in argumentation-theoretic terms by [Du]. An elaboration of [EK] in [KM1] reasons with a larger class of abducibles, including both positive and negative database facts as hypotheses. It computes answers to queries and hypothetical explanations for justifying the answers. Each explanation of an answer to a query can easily be translated to database updates by which the query, interpreted as an update request, can be satisfied. In more general terms, this can be seen as a form of computed belief revision. Possible improvements of [KM1] interms of effectiveness and efficiency have been described in [De].

The procedures in [EK] [KM1] [De] tolerate definitions of facts in terms of their own negation, e.g. fact ← ~fact, while that is considered inconsistent by the [Cl] completion. Also SLDNF tolerates such definitions, in the sense that it does not use efq for deriving arbitrary consequences from the if-and-only-if completion fact ↔ ~fact, which is inconsistent, in terms of classical two-valued logic. However, SLDNF loops when trying to answer the query ← fact, while the abductive procedures properly terminate with failure to find any explanation of answering the query positively; also, querying the negation ← ~fact is answered that way. In general, finite failures to explain both a fact and its negation indicate that the truth value of the fact is unknown, while one-sided finite failure to explain either a fact or its negation but not both justifies a *yes*/*no* answer.

To sumarize this section in one sentence, it can be said that, with a third truth value in the semantics of DDBs, a certain amount of paraconisteny is captured.

An abductive interpretation of NaF is described in [EK]. More generally, SLD and NLP can be conveniently extended by abductive inference rules in order to enable automated reasoning with constraints in *dynamic* theories, in which the set of axioms, or at least a subset thereof, is treated as an updatable set of beliefs (cf. [KM1] [KKT] [De]). DDBs are typical representatives of such dynamic theories the beliefs of which may change over time. Rather than applying efq or giving up in the presence of inconsistency, such theories either try to restore consistency by committing to hypothetical changes which restore consistency, or at least to live with inconsistency without becoming trivial. However, the paraconsistency potential of abductive extensions of DDBs has, to the knowledge of the author, never been addressed in print to a large extent.

In the remainder, we first address some further aspects of paraconsistency in the semantics of DDBs. Then, we argue why the use of a third truth value in the semantics of DDBs with integrity constraints lends itselves naturally toward paraconsistency. Finally, we develop an abstract three-valued semantics of DDBs with integrity constraints which allows for several degrees of paraconsistency.

Paraconsistency in the semantics of deductive databases

DDBs do not support the inference of arbitrary consequences from inconsistent components, such as contradictory definitions of database relations (predicates) or violated integrity constraints. Moreover, computed answers in deductive databases typically make sense, almost no matter to which degree consistency or integrity is corrupted. That is because query answering is goal-oriented, i.e., it focuses on input clauses that are relevant for refutations by reducing the proof of goals to the proof of antecedents of clauses the conclusion of which is the goal. Thus, computed answers are correct wrt a consistent subset of the database, which is given by the formulas that are used as input clauses in the query answering refutations.

The field of DDBs ("pure logic programming") provides a solid semantic foundation which is unparalleled by other programming paradigms. In particular, the semantics of the definite case (pure Horn clause theories), described denotationally as the least fixpoint of the immediate consequence operator [vK], is unanimously agreed upon. For the non-monotonic inference of negative information from a definite database, there are two competing semantics: The closed world assumption (*cwa*) [R1] and the database completion (*comp*) [Cl]. For the semantics of normal DDBs (which generalize definite databases by admitting clauses with negative literals as premises), a model-oriented and a proof-procedure-oriented line of development can be distinguished. In a sense, they generalize *cwa* and *comp*, respectively. The model-oriented line has brought forward proposals such as the (credulous) stable [GL] and the (skeptical) well-founded model semantics [VRS], which coincide for fairly general and common classes of cases. The procedural

The Robinson resolution rule is known to be complete. More precisely, resolution is refutation-complete and subsumption-complete. *Refutation completeness* means that, from each inconsistent set T of clauses, the empty clause can be inferred (abbr. T |– [ ]) by a finite sequence of resolution steps. A refutation of a clause G in T shows that T ∪ {G} is inconsistent and hence the negation ~G can be inferred. Thus, resolution also uses the *reductio ad absurdum* inference rule, which however is strictly weaker than efq.

Subsumption completeness means that for each logical consequence C of T (abbr. T |= C), there is a clause C' such that (C' → C) and T |– C hold. For example, in the theory {p}, there is no sequence of resolution inference steps by which the disjunction (p ∨ q) could be inferred, but both {p} |– p and (p → (p ∨ q)) hold. Moreover, ([ ] → F) holds for each formula F, i.e., each formula is subsumed by the empty clause, and the latter is derivable in an inconsistent theory. Strictly speaking, however, efq does not apply in resolution, in the sense that, in general, there is no sequence of resolution steps by which an arbitrary formula could be explicitly inferred from any finite inconistent theory.

Logic programming and paraconsistency

The well-known selection-driven linear resolution inference rule of SLD resolution (shortly, SLD) in logic programming (LP) is complete and also paraconsistent for definite Horn theories and queries in a similar sense as outlined above for resolution in general. A *Horn theory* is a set of clauses of form A ← B, where either the *head* A or the *body* B or both may be empty. If not empty, A is an atom and B is a conjunction of atoms. A Horn theory is definite if it contains no *denial*, i.e. no clause of form ← B (B not empty). Apart from the empty clause, SLD only infers positive conclusions from Horn theories. Queries to Horn theories are expressed as denials to be refuted. Otherwise, denial clauses in Horn theories are called *constraints*. A typical point in case is the use of integrity constraints in deductive databases. However, the essential point in this section is the observation that SLD is paraconsistent because, apart from [ ] and the denials in the theory, no negated formula would ever be inferable.

In LP and particularly in deductive databases (abbr.: DDBs), the introduction of the *Negation as Failure to prove* rule (*NaF*), by which negative concusions and answers can be inferred, was found convenient [Cl]. The next step then was to extend the Horn clause syntax by admitting negated atoms in the body of so-called *general* (also, *normal*) clauses, and to integrate NaF into SLD, resulting in SLDNF [Cl] [Ll]. However, even though the amount of inferable information is extended considerably by NaF, the basic feature that there is no way of applying efq remains, essentially for the same reasons as argued before.

Some Aspects of Paraconsistency in Deductive Databases

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

Abstract

Resolution in general, logic programming in particular, and even more so abductive logic programming and deductive databases are paraconsistent. Deductive databases are theory systems which change over time. They do not support the inference of arbitrary consequences from inconsistent components, such as contradictory definitions of database relations (predicates) or violated integrity constraints. Because of the goal-orientedness of query answering which focuses on consistent subsets of relevant database clauses, computed answers in deductive databases typically make sense, almost no matter to which degree consistency or integrity is corrupted. Consistency requirements for the knowledge embodied by a database is expressed by an associated set of integrity constraints which, taken on its own, may also vary in its degree of (in-)consistency. Deductive databases evolve via sequences of state changes, effected by updates that can be seen as belief revision steps. Instead of rejecting update requests because of inconsistency, i.e., integrity violation, a certain amount of paraconsistency can be tolerated sometimes, in order to accommodate such requests. We discuss to which extent deductive databases can or do tolerate various degrees of paraconsistency. We do not discuss consistency nor integrity in terms of compatibility of replicated or distributed data or in terms of data security.

Resolution and paraconsistency

In this and the next section, we are going to argue why it can be said that resolution and particularly SLD resolution in Horn theories and their usual extensions in normal and abductive logic programming (abbreviated NLP, ALP) are paraconsistent.

A set R of inference rules is taken to be *paraconsistent* if the *ex falso quodlibet* inferene rule (*efq*) does not apply. That is, for some theory T (i.e., a set of closed formulae called axioms) which is inconsistent (i.e., which has no model), there is a formula which is not derivable in T by R.

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

Ludwig
Maximilians–
Universität
München
LMU

Some Aspects of Paraconsistency in Deductive Databases

Hendrik Decker

(submitted)
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1997-17, October 1997