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

Ludwig-
Maximilians-
Universität
München

On Paraconsistency in Deductive Databases

Hendrik Decker

presented at *First World Congress on Paraconsistency*, University of Ghent (Belgium), 1997
preliminary abstract at http://allserv.rug.ac.be/ jvmeirve/wcp97/
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1997-11, August 1997

On Paraconsistency in Deductive Databases

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

Abstract

Deductive databases are paraconsistent 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. Moreover, computed answers in deductive databases typically make sense, almost no matter to which degree consistency or integrity is corrupted. Integrity of the knowledge embodied by a database is expressed by an associated set of database 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 or integrity violation, a certain amount of paraconsistency can be tolerated sometimes, in order to accommodate such requests. We recapitulate various forms of inconsistency and integrity violation in deductive databases. We investigate to which extent deductive databases can or do tolerate various degrees of paraconsistency. We do not discuss consistency in terms of compatibility of replicated or distributed data, nor do we discuss integrity in terms of security.

Paraconsistency in the semantics of databases

The field of deductive databases ("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 deductive databases (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 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 (a common logic programming proof procedure). 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 deductive databases 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.

The procedures in [EK] and [KM1] tolerate definitions of facts in terms of their own negation, e.g. $\mathsf{fact} \leftarrow \sim\mathsf{fact}$, while that is considered inconsistent by the [Cl] completion. Also SLDNF tolerates such definitions, in the sense that it does not use the classical *ex contradictione sequitur quodlibet* rule for deriving arbitrary consequences from the if-and-only-if completion $\mathsf{fact} \leftrightarrow \sim\mathsf{fact}$, which is inconsistent, in terms of classical two-valued logic. However, SLDNF loops when trying to answer the query $\leftarrow \mathsf{fact}$, while the abductive procedures properly terminate with failure to find any explanation of answering the query positively; also, querying the negation $\leftarrow \sim\mathsf{fact}$ 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.

Paraconsistency in the semantics of integrity constraints

In [D1], a procedure called SLDAI is proposed which is an improvement over [KM1] in terms of effectiveness and efficiency. Several restrictions which limit the [KM1] procedure, in particular concerning the reasoning with integrity constraints and basic facts, are relaxed in SLDAI. Moreover, we conjecture that SLDAI never computes updates that would increase the current degree of consistency beyond what is required as a necessary and sufficient condition of integrity in [KM1]. In other words, one could say that updates computed by SLDAI are conjectured to never increase the amount of paraconsistency in the database. Progressing degrees of paraconsistency are sketched in the remainder of this paper. They are investigated in more detail in the companion paper [D2].

An integrity constraint is a closed formula 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 the framework of [EK] [KM1] [D1], also the consistency of the database itself is defined on the level of the associated integrity constraints, by denials of the form $\leftarrow p(x1,...,xn)$ & $\sim p(x1,...,xn)$, for each relation (predicate) p in the underlying 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], also the law of excluded middle is implicitly assumed as an integrity constraint. In general, integrity constraints serve to narrow the search space of admissible solutions to queries and update requests. However, it seems that practically no kind of inconsistency between database and constraints, or amongst the constraints themselves, can invalidate computed results.

There is an ongoing debate about the adequacy of various ontologic and epistemic concepts of what it could or should mean that a database satisfies or violates its integrity constraints (cf., e.g., [R2]). According to the popular *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 deductive databases (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 may then be considered satisfied if and only if each of the constraints is true in each of the partial stable models of the database.

A weaker, i.e. 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 $D = \{p \leftarrow \sim q, q \leftarrow \sim p\}$ satisfies the integrity constraint $I = \leftarrow \sim p$ (which denies that ~p could hold) according to the consistency view, since {p, ~q} is a stable model of D in which I is true; however, I is not true in the second stable model {q, ~p} of D, hence I is not satisfied in D 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.

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 = {p ← ~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 considers both p and ~p unknown, and I = ← 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. Similarly, also I' = ← ~p is neither true nor false in { }.

However, one may hesitate to tolerate both constraints I and I' at a time, since I" = {← p, ← ~p} appears to be self-contradictory. On the other hand, 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 ← ~p}, but it would be in the empty database Ø, since ~p is true in the preferred extension of Ø and hence violates the constraint ← ~p in I".

In the appendix of [D2], we present a semantics of databases and integrity constraints which is based on the unique "open model" of a database. That semantics is even more tolerant in terms of permitting high degrees of paraconsistency than what has been addressed above. The open model assigns true precisely to the logical consequences of the database, and assigns false to a literal if and only if its complement is true. In particular, no negative literal is assigned true. Thus, the law of contradiction is never violated. Moreover, integrity is considered not violated as long as no constraint is definitely violated under the interpretation of the open model. For example, the empty database does not violate I" = {← p, ← ~p}.

As a topic of ongoing research, we investigate the conjecture that the reliability of what is computed by SLDAI is never derogated by any degree of paraconsistency, in terms of the spectrum sketched above. In other words, we want to convince ourselves that SLDAI *always* behaves in an arguably sensible way.

Footnote: An instance of a database of form D above is the well-known barber's paradox, which can be expressed by the clause shaves(barber, x) ← ~shaves(x, x), i.e., "the barber shaves each individual x who does not shave himself". Another instance, win(x) ← ~win(y) is known from game theory. Yet another example has been provided by [To].

References

[Cl] Clark: Negation as failure, in [GM], 1978.

[D1] Decker: An extension of SLD by abduction and integrity maintenance for view updating in deductive databases, Proc. JICSLP, 1996.

[D2] Decker: Toward a paraconsistent semantics of integrity in deductive databases,

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

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

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

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

[GM] Gallaire, Minker (eds): Logic and Data Bases, Plenum Press, 1978.

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

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

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

[R1] Reiter: On closed world databases, in [GM], 1978.

[R2] Reiter: On asking what a database knows, in J. Lloyd (ed): Computational Logic, Springer, 1990.

[SK] Sadri, Kowalski: A theorem-proving approach to database integrity, in Minker (ed): Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, 1988.

[SZ] Sacca, Zaniolo: Stable models and non-determinism for logic programs with negation, Proc. ACM Symp. Principles of Database Systems, 1990.

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

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

[VRS] Van Gelder, Ross, Schlipf: Unfounded sets and well-founded semantics for general logic programs, Proc. ACM Symp. Principles of Database Systems, 1988.

Acknowledgements

Francois Bry hosted the author as a visitor at the computer science department of the LMU München, where this paper was written. A preliminary version was presented at the First World Congress on Paraconsistency, Ghent University (Belgium), 30 July – 2 August 1997.