UPMAIL Technical Report No. 129
September 19, 1996
ISSN 1100-0686

Logic Databases and the Meaning of Change

Jorge Bocca

IDS Integral Decisions Systems SE GmbH
Munich, Germany;
University of Birmingham
England
jorge@ids.de

Hendrick Decker

Siemens ZFE T SE 4
D-81730 München
Germany
decker@werra.zfe.siemens.de

Michael Kifer

Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400
U.S.A.
kifer@cs.sunysb.edu

Andrei Voronkov

Computing Science Department
Uppsala University
Box 311, S 751 05 Uppsala,
Sweden
voronkov@csd.uu.se

Abstract

This technical report contains preliminary proceedings of the seminar "Logical Databases and the Meaning of Change" held in Dagstuhl, Germany, September 23–27, 1996.[1]

[1]This technical report and other technical reports in this series can be obtained at `http://www.csd.uu.se/reports.html` or at `ftp.csd.uu.se` in the directory `pub/papers/reports`. Some reports can be updated, check one of these addresses for the latest version.

Contents

View Updating by Abduction and Integrity Maintenance (Extended Abstract)

Hendrik Decker

Siemens ZFE T SE 4, D-81730 München, Germany
hendrik@zfe.siemens.de

Abduction is a key technique for advanced applications of logic databases and artificial intelligence. Besides deduction and induction, abduction has been identified by C.S.Peirce as one of three basic principles of formal reasoning. Abduction infers hypotheses. For an observation A and a given theory T, a set H of sentences is called a hypothetical explanation of A if the union of T and H is consistent and entails A as a logical consequence. Similarly, for an answer substitution θ to a query $\leftarrow Q$ in a database D that satisfies a given integrity theory IT, a set H of facts is called a hypothetical justification of θ if the union of D and IT and H is consistent and entails the universal closure of $Q\theta$. The similarity is obvious since $Q\theta$ may be taken as an observed fact and H then is taken as its explanation.

A more operational characterization of abduction which sets it off against deduction and induction follows. From a causal rule of the form $A \leftarrow B$ and the knowledge of B, deduction infers A. Induction hypothetically infers universally quantified generalizations of the form $A \leftarrow B$ from observing that instances A' of A hold whenever corresponding instances B' of B have been observed. From a given causal rule $A \leftarrow B$, abduction generates the hypothesis B to explain A. In database terms, abduction infers, from knowing a universally quantified deduction rule $A \leftarrow B$, that $B\theta$ is a hypothetical justification for the answer θ to the query $\leftarrow A$.

Deduction can simulate abduction. Hence, existing deductive logic programming procedures can be easily adapted to implement abductive reasoning.

Several forms of abductive reasoning have been developed in various fields of artificial intelligence. In logic databases, abductive logic programming has first been used for inferring negative answers and default answers, and more generally, for non-monotonic reasoning. Non-monotonic reasoning may involve default hypotheses as well as non-default hypotheses. A prominent database application of abduction requiring non-default hypotheses is view updating.

View updating means to satisfy an update request by modifying the data from which those that are requested to change are derived. In simple cases, that can be done by tracing attempts to derive the request, and drawing updates from the reasons of failure of the attempts, as e.g., in [To, D2, GL, KM1, KM2]. This approach is complicated by the presence of integrity constraints which may invalidate updates drawn from failed derivations, and also by negation in the condition of database clauses, as observed in [D2]. In [KM1, KM2], the abductive approach of processing negation by [EK] is extended to deal also with positive hypotheses, not just negative ones. The hypotheses correspond to the reasons of failure to derive the request and, when assumed to hold, contribute to make the request derivable.

We present and discuss SLDAI, an SLD-based proof procedure extended by Abduction and Integrity constraints, for view updating in logic databases. For a given update request in a database which satisfies some integrity theory, SLDAI computes solutions (i.e., hypothetical insertions and/or deletions of facts about base predicates) which satisfy the request while preserving integrity. Apart from filtering solutions by rejecting those that would violate integrity, SLDAI may optionally run attempts of corroborating solutions. That way, preferable solutions may be chosen out of several alternative ones. Redundancy of solutions is a filtering criterion for choosing solutions which may come to bear in various ways. One is that solutions often are required to satisfy some criteria of minimality (i.e., lack of redundancy). Another is that solutions may be corroborated by having consequences that are redundant, in the sense that such consequences already are known to hold, independent of any hypothesis in the solution. Moreover, SLDAI avoids redundant computations that are unnecessarily run in other procedures.

SLDAI combines two lines of development. One has led to SLIC, a Selection-driven Linear resolution procedure based on SLDNF, for Integrity Checking in deductive databases [DC]. The other line of development has started with Eshghi & Kowalski's work [EK] on casting the negation-as-failure rule of SLDNF in an abductive framework, and has led to the abductive procedure for database updating by Kakas & Mancarella [KM1, KM2], and beyond. While [EK] is an example of default reasoning, [KM1, KM2] is one for reasoning with non-default hypotheses.

An essential difference between SLDNF and procedures [KM1, KM2], SLIC, SLDAI is that clauses in denial form, by which integrity theories are represented, can (at least to a certain extent) be taken into account as candidate input clauses by the latter, while SLDNF only is able to deal with positive input clauses, i.e., normal database clauses with exactly one positive literal in the head. While SLDNF always reasons backward, denials may act as input clauses only if resolution proceeds forward.

However, [KM1, KM2] takes integrity constraints into account only to a limited extent. [KM1, KM2] proceeds forward from hypotheses only in singular resolution steps, never by two or more forward steps in a row. Hence, indirect consequences of hypotheses are not considered, even though they may be as harmful wrt integrity as immediate consequences. In fact, forward reasoning which resolves the literal in the head of a database clause in a derivation against a literal in the body of an input clause only involves a slight extension of the usual backward reasoning capabilities of SLD-like procedures, which is still far from the expensive generality of arbitrary first-order procedures such as SL resolution [KK] or SLY [CaD]. Such extensions have been first proposed in Sadri & Kowalski's procedure [SK] for integrity checking upon insertions in definite databases. [SK], together with other procedures proposed in [De] and [LT], is a forerunner of SLIC.)

Moreover, there is another, more intricate kind of forward reasoning which may complicate matters considerably. Complications arise if abduced hypotheses have to be propagated in forward direction through literals of opposed polarity, for generating negative hypothetical consequences. In SLIC and SLDAI, such complications are avoided by simple but effective steps, to be sketched later on. Before, let us consider this particular kind of forward propagation in some more detail.

Suppose some literal L is hypothesized (abduced) in the course of an SLDAI derivation. Further, suppose there is a clause $A \leftarrow B$ in the database, with a literal L' in B such that the atoms of L and L' unify but L and L' are of opposed polarity. Then, it may happen that instances of A which could be deduced in the absence of hypothesis L can no longer be deduced as soon as L is assumed to hold.

For instance, suppose the database contains the clause $p \leftarrow q \& r$ and the negative fact $\neg q$ is hypothesized. Possibly, p is derivable via that rule as long as $\neg q$ is not assumed to hold, viz if q and r hold. However, if $\neg q$ is hypothesized (expressing that q should be deleted for satisfying some update request), then p may no longer be derivable. Yet, it may still be derivable after the deletion of q, since there may be other clauses in the definition of p via which p could be derived, independent of the presence or absence of q. In general, we essentially always have one of the following four possible situations whenever a literal L is hypothesized and a literal of opposed polarity which unifies with L occurs in the body of a rule of form $A \leftarrow B$:

1. A is derivable if L is not hypothesized, but A no longer is derivable if L is hypothesized. In the example above, this is the case if q and r hold, no assumption about $\neg q$ is made, and if there is no other way to derive p after $\neg q$ is hypothesized.

2. A is derivable if L is not hypothesized and remains derivable if L is hypothesized. In the example, this is the case if q and r hold while no hypothesis $\neg q$ is assumed, and if there is some other way to derive p when $\neg q$ is hypothesized.

3. A is not derivable if L is not hypothesized and remains non-derivable if L is hypothesized. For instance, this is the case if r does not hold, independent of the absence or presence of hypothesis $\neg q$.

   4) A is not derivable via $A \leftarrow B$ as long as L is not hypothesized but becomes derivable as soon as L is assumed to hold. This, however, can happen only via some other clause, which then is captured by some other forward reasoning path. For instance, suppose the fact q is present in the current state of the database in the example, and besides $p \leftarrow q\&r$, there are also clauses $r \leftarrow \neg q$ and $p \leftarrow r$. Then, p is not derivable, but the hypothesis of $\neg q$ (which overrides the presence of q, suggesting that q should be deleted) makes p derivable.

Dealing head-on with the intricacies just outlined is responsible for lots of unpleasant complications encountered in procedures for integrity checking and view updating, e.g. [De, SK, DW, Ol] and others. As opposed to these procedures, SLIC and SLDAI adapt a much simpler kind of forward reasoning through literals of opposed polarity as proposed in [LT]. More precisely, for a hypothesis L and a clause $A \leftarrow B$ with literal L' in B such that the atoms of L and L' unify but L and L' are of opposed polarity, SLIC infers the clause $\neg A \leftarrow \neg A$ as a consequence. Intuitively, such a clause expresses that the negation of A (in the head) can be taken to hold if it is not possible to prove A (i.e., if the negation $\neg A$ in the body holds by default). In fact, this seems to be the easiest way to capture possible negative consequences of hypotheses, in the average case. ([CCD, CD] showed that, in some cases, it might be advantageous to spend some more effort on computing a more precise characterization of possible consequences of hypothetical updates, but [CD, DC] conclude that, usually, an approach as pursued in SLIC is more satisfying, because it is always simpler and most often more efficient than procedures which invest more effort in computing consequences of updates that "go through negation".)

Back to [KM1, KM2] and SLDAI. In general, the [KM1, KM2] abductive framework is simplified and extended by SLDAI. User-defined integrity is maintained by each SLDAI-computed solution for satisfying some update request. As opposed to other abductive approaches, SLDAI is able to take the presence or absence of facts about abducible base predicates in the current database state into account, thereby exploiting that the current state satisfies integrity. As opposed to that, other abductive procedures usually neglect the presence or absence of abducible facts. Thus, the efficiency of integrity checking is sacrificed and an unnecessary overhead of hypothesized facts is garnered, instead of taking advantage of the evidence of base facts in the current state.

Further, we show how to make computed solutions independent of the used selection function. (This independence should be interesting, since the computed solutions of almost all known abductive procedures suffer from the circumstance that their computed solutions depend on the employed selection function, and hence are incomplete, in the sense that preferable solutions may be missed. The procedure for computing view updates by Teniente & Olive' [TO] is, to our knowledge, the only one which does not depend on the selection function, but suffers from other drawbacks, not addressed in this abstract.)

Another feature of SLDAI which sets it apart from other abductive procedures: The latter may assume new hypotheses only one at a time, although all ground abducible literals in a goal to which an update request is reduced will need to be hypothesized at some point, for refuting the request. As opposed to that, SLDAI may eagerly hypothesize several literals at selection time in addition to a selected one, such that subsidiary computations can be significantly shortened by taking additional hypotheses into account. But special care must then be taken with additional hypotheses that are not yet being checked for consistency.

Comparison to other abductive approaches of integrity-maintaining view updating is facilitated by introducing a new semantic concept of integrity which distinguishes a weak and a strong form of constraint satisfaction and violation. It can be shown that SLDAI is sound wrt the strong notion of integrity satisfaction for classes that are broader than the class of stratified databases. With a transformation as described in [TK], SLDAI is sound wrt all normal databases and requests (i.e., no restrictions whatsoever need to be imposed).

A problem with many abductive procedures is their incompleteness, in the sense that the failure to compute an abductive explanation of an incomplete procedure does not justify to conclude that "no" would be a correct answer. In comparison, SLDAI does better wrt computing existing solutions than other abductive procedures. No extension for reasoning with arbitrary first-order theories are needed for that.

We briefly sketch how to use SLDAI for schema validation [DTU]. We argue that SLDAI has the potential for tackling abductive tasks which reach beyond a conventional database context.

Bibliography

[CCD] Celma, Casamayor, Decker. Improving integrity checking by compiling derivation paths. In *Proc. 4th Australian Database Conf.*, 1993.

[CaD] Casamayor, Decker. Hypothetical query answering in first-order databases. In *Proc. 5th SCAI*, 1995.

[CD] Celma, Decker. Integrity checking in deductive databases — the ultimate method? In *Proc. 5th Australasian Database Conf.*, 1994.

[DC] Decker, Celma. A slick procedure for integrity checking in deductive databases. In *Proc. 11th ICLP*, 1994.

[De] Decker. Integrity enforcement on deductive databases. In *Proc. Expert Database Systems*, 1987.

[D2] Decker. Drawing updates from derivations. In *Proc. 3rd ICDT*, 1990.

[DTU] Decker, Teniente, Urpi. How to tackle schema validation by view updating. In *Proc. 5th EDBT*, 1996.

[DW] Das, Williams. A path-finding method for checking integrity in deductive databases. *Data and Knowledge Engineering*, 4, 1989.

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

[GL] Guessoum, Lloyd. Updating knowledge bases I/II. *New Generation Computing*, 8/10, 1990/1991.

[KK] Kowalski, Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2, 1971.

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

[KM2] Kakas, Mancarella. Knowledge assimilation and abduction. In *Proc. ECAI workshop on Truth Maintenance*, LNAI, 1990.

[LT] Lloyd, Topor. A basis for deductive database systems II. *J. Logic Programming*, 3, 1986.

[Ol] Olive. Integrity constraints checking in deductive databases. In *Proc. 17th VLDB*, 1991.

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

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

[TO] Teniente, Olive. Updating knowledge bases while maintaining their consistency. *VLDB Journal*, 4, 1995.

[To] Tomasic. View update translation via deduction and annotation. In *Proc. 2nd ICDT*, 1988.

Adding information by taking topics into account

Laurence Cholvy

ONERA-CERT
2 avenue Ed Belin
31055 Toulouse, FRANCE
cholvy@cert.fr

In the present work, we contribute to define what becomes a database after the addition of a piece of information.

We show that the result of the addition trivially depends on the current state of the database, on the new information, but also depends on the topics of information. More precisely, it depends on the relative reliability, topics by topics, of the agents who store data in the database.

First of all, before answering the question "what is the meaning of an addition ?" in the context of a database, we must first precise what is a database.

What is a database ?

Generally[1], databases are tools for storing information representing facts which are supposed to be true in the real world.

We write "supposed to be true" instead of "true", because information stored in the databases are sent by agents who observe the real world (sensors, people...) and who may be unsafe [JD94], [CDJ94]. For instance, the tuple *cold(Paris)*, stored in a database, means that it is true in the real world, that it is cold in Paris, only if the sensor which sent this information is known to be safe. If it is not known to be safe, we can only say that it is supposed to be cold in Paris.

So, **a database is a set of beliefs some agents have about the real world**. These beliefs are true in the real world or not, depending on the safety of agents.

Notice that this point of view is different from the traditional ones [GMN78], [GMN84], [Rei84]. Indeed, more traditionnally, people consider that tuples stored in a database represent facts which are true in the real world. This comes to assume that agents who send information to the database always tell the truth. Furthermore, people often make the Closed World Assumption, considering that absent tuples represent facts which are false in the real world. This assumption comes to consider that the real world is entirely known. In our approach, we reject both assumptions.

What is an addition ?

Since a database is seen as a set of beliefs some agents have about the real world, adding some pieces of information in the database means adding some beliefs of some agents. For instance, inserting *Employee(John)* in a database means that there is an agent, say a, who thinks (believes) that John is an employee. Inserting *cold(London)* means that there is an agent, say b, who thinks that it is cold in London.

Defining the semantics of an addition in a database aims to decide what is the database state which results from that addition in the current state.

[1]We do not address here, the case when databases are used to store information expressing how the real world should be, i.e, when databases are used to store norms

Let DB denote the current state of the database, New denote the information to be added and DB + New denote the result of the addition of New into DB. The problem is to define DB + New from DB and New.

Classical database approach

Traditionnally, databases are also associated with a set IC of integrity constraints restricted to negative clauses and with a set IDB of deduction rules, restricted to definite Horn clauses. DB is generally restricted to a set of positive literals.

Let $\mathrm{DB}_{\mathrm{IDB}}$ denote the unique Herbrand minimal model of DB ∪ IDB.

The addition into DB, of a tuple (or a set of tuples) denoted New, is defined in the following way :

- If $(\mathrm{DB} \cup \mathrm{New})_{\mathrm{IDB}} \models \mathrm{IC}$ [2] then DB + New = BD ∪ New
- Else, DB + New = DB

This means that, given IDB and IC, if the new information do not contradict the current database state, then we just add it. Else, we reject it. It is a rather conservative attitude in which the current set of information is always preferred to the new information.

Example :

Consider a database used in an university for storing information about students and employees. In this university, there is a restaurant for the employees and a self-service for the students. Nobody is allowed to access both. This is expressed by two deduction rules and one integrity constraints, which are the following :
IDB = { $\forall$ *x student(x)* $\rightarrow$ *self(x),* $\forall$ *x employee(x)* $\rightarrow$ *restaurant(x)*}
IC = { $\forall$ *x* $\neg$ *restaurant(x)* $\vee$ $\neg$ *self(x)* }
Let us assume that the current state of the database is : DB = { student(john), self(philip)}
Consider the addition of the following information : New = { restaurant(john), employee(philip)}
One can notice that $(DB \cup New)_{IDB} \not\models IC$*, so according to this conservative approach, DB + New = DB.*

Belief revision approach

In the domain of belief revision, the addition of a piece of information has been carefully studied [FUV83], [AGM85], [MS86], [FKUV86], [Gar88], [Neb89], [KM91], [Som94], [GM92]. In this community, people do not distinguish between different types of beliefs. So they do not make appear sets like IC and IDB.

Most people agree on defining the addition in such a way that the new information belongs to the resulting state. I.e, they postulate that : New ∈ DB + New.

They of course agree on another postulate that states that DB + New is consistent.

- If BD ∪ New is consistent, then DB + New is defined by DB ∪ New. It satisfies the two previous postulates.
- Else, DB ∪ New cannot be candidate to be DB + New, because it does not satisfies the consistency postulate. So, because New must belong to DB + New, some data must be rejected from DB. This is the revision process

[2]Sometimes, this condition is replaced by the more general one : (DB ∪ New ∪ IDB ∪ IC) is consistent

There are different approaches for implementing revision we cannot detail here. Many of them are based on a minimal change postulate i.e, they only reject as few data as possible.

Let us only insist on the fact that the first postulate we mention comes to consider that the new information is preferred to the current set of information.

Example (continued)

Let us consider again the previous example.

Let DB denotes here DB $\cup$ IDB $\cup$ IC.

Since (DB $\cup$ New) is not consistent, the revision attitude consists in rejecting as many data in DB as necessary, in order to add New.

One solution could be : DB + New = IDB $\cup$ IC $\cup$ New.

I.e, the tuples student(John) and self(Philip) have been rejected.

Intuitive presentation of our proposal

The approach we want to suggest is intermediate between the two previous ones.

- We have showed that in classical databases, the current database state is always preferred to the new information ; in belief revision domain, the new information is always preferred to the current state. As for us, **we postulate that the politics of adding information may be different according to the considered addition**. For a given addition, it can be the case that the current state is preferred to the new information, but for another addition, it can be the contrary.

  The preference relation between DB and New, denoted >, may express the relative credibility that exists between the two agents whose beliefs are respectively stored in DB and in New.

  (DB > New) means that the agent whose beliefs are stored in DB is more reliable than the agent whose beliefs are expressed in New. In that case, the idea is to keep all the beliefs stored in DB, and add to them, as many beliefs stored in New as possible.

  (New > DB) means that the agent whose beliefs are stored in New is more reliable than the agent whose beliefs are expressed in DB. In that case, the idea is to keep all the beliefs stored in New, and add to them, as many beliefs stored in DB as possible.

- We immediately refine this previous idea in the following way :

  Considering an order between DB and New is not realistic : generally, we cannot say that an agent is more reliable than another one, for any piece of information it provides.

  We think that it is more realistic to order agents topic by topic : for a given topic, DB is consider to be more reliable than New, but for another one, New is more reliable than DB.

  The notion of topic has been introduced in order to characterize sets of sentence from the point of view of their meaning, independently of their truth values. In the context of Cooperative Answering, topics are used to extend an answer to other facts related to the same topics [CD88], [CD92]. In our work, topics are sets of literals, thus defining sub-languages. We postulate that if a literal belongs to a topic, its negation also belongs to this topic.

Example (continued)

Let us now consider two topics. T1 groups the information about John, T2 groups the information about Philip. So, student(John) $\in$ T1, restaurant(John) $\in$ T1, employee(Philip) $\in$ T2, self(Philip) $\in$ T2.

Let us consider again the addition of New to DB. We assume that (DB $>$ New) for topic T1 and that (New $>$ DB) for topics T2. In other terms, DB is considered as more reliable that New for any information which concerns John, and New is considered as more reliable than DB for any information which concerns Philip.

Here is the intuitive reasoning we want to formalize :

Student(John) is explicitely provided by agent DB. It contradicts the fact $\neg$ student(John), deduced by agent New from its explicit belief restaurant(John). However, as far as John is concerned (topic T1), DB $>$ New. So, student(John) is kept and we reject any proof of $\neg$ student(John), i.e, we reject restaurant(John).

In a similar way, employee(Philip) is explicitely believed by New. It contradicts $\neg$ employee(Philip) deduced by DB from its explicit belief self(Philip). But, as far as Philip is concerned (topic T2), New $>$ DB. So, employee(Philip) is kept and self(Philip) is rejected.

So, DB + New = $\{$ student(John), employee(Philip)$\}$.

However, orders expressed between agent DB and agent New, topic by topic, cannot be defined without taking the structure of topics into account. Indeed, **we will ask the orders to satisfy a compatibility condition with the topics**, which is an extension of the compatibility condition defined in [CD94]. Roughly speaking, as soon as two topics intersect or as soon as some rules of IDB relate two literals belonging to two different topics, the orders associated with these topics must be identical.

Example (continued)

Now, let us consider the two following topics : T1 groups any information about the status of the people in the university, T2 groups the information about the place where they go to eat.

So, student(John) $\in$ T1, employee(Philip) $\in$ T1, restaurant(John) $\in$ T2 and self(Philip) $\in$ T2.

Consider again that (BD $>$ New) for T1 and that (New $>$ BD) for topic T2. These two orders are different, although there is a relation, through the rules of IDB, between a literal in T1, for instance student(John) and a literal in T2, for instance self(John).

If we reason as previously, this leads to keep student(John) and restaurant(John), and to reject self(Philip) and employee(Philip).

So DB + New = $\{$ student(John), restaurant(John) $\}$.

However, this set cannot be adequate since it is not consistent with IDB.

Formalization

The work we will present can be considered as an extension of the different results we have presented in [Cho94], [Cho93] [CD94], [Cho95], [Cho96]. Our assumptions are the following :

- DB and New are sets of (positive or negative) literals
- There is a set of rules (deduction rules or integrity constraints) IDB (which are general clauses)
- DB $\cup$ IDB (resp, New $\cup$ IDB) is consistent and is equivalent to a set of literals
- There are m topics of information in the language

We will present a logic of belief, based on KD logic [HC72], whose modalities express the data which are explicitely stored in DB and in New, the data which are deduced from DB and IDB, or New and IDB and also the data which are deducible in the database resulting from the addition of New into DB, by taking into account the relative reliability of DB and New, topic by topic.

For instance, the formula $B_{exp_{DB}}l$ will mean that the literal l is explicitely stored in DB ; the formula $B_{DB\ DB}l$ will mean that the literal l is deducible from DB and IDB ; the formula $B_{DB>New\quad New>DB}l$ will mean that literal l is deducible from the database obtained by adding New to DB, assuming that DB>New for the first topic and that New>DB for the second. Notice that this modality is defined only if the orders $DB > New$ and $New > DB$ satisfy the condition of compatibility with the two topics.

Let us end this extended abstract by presenting the axiomatic of the logic. We present a general case where there are several databases to be merged. The addition of New to DB can be seen as a particular case.

Let O_k be an order between several databases (two in the particular case of addition).

Let i be the name of a database (In the particular case of addition, i is DB or New).

$O_k > i$ denotes the order which extends O_k with i by inserting i as minimal element.

$O_k + i$ denotes any order which extends O_k with i, i.e, i is inserted at any place in the order O_k.

- (A0) Axioms of the propositional logic.
- (A1) $B_{O_1 \ldots O_m}$ ¬ F → ¬ $B_{O_1 \ldots O_m}$ F
- (A2) $B_{O_1 \ldots O_m}$ F ∧ $B_{O_1 \ldots O_m}$ (F → G) → $B_{O_1 \ldots O_m}$ G
- (A3) $B_{O_1 \ldots O_m}$ l → $B_{O_1+i \ldots O_k>i \ldots O_m+i}$ l if l is a literal belonging to the topic T_k.
- (A4) $\mathrm{B}_{\mathrm{exp}_\mathrm{i}}$ l ∧ ¬ $B_{O_1 \ldots O_m}$ ¬ l → $B_{O_1+i \ldots O_k>i \ldots O_m+i}$ l if l is a literal belonging to topic T_k.
- (A5) $B_{O_1+i \ldots O_k>i \ldots O_m+i}$ l → $B_{O_1 \ldots O_k \ldots O_m}$ l ∨ $B_{i \ldots i}$ l if l is a literal belonging to topic T_k.
- (A6) $\mathrm{B}_{\mathrm{exp}_\mathrm{i}}$ l → $B_{i \ldots i}$ l
- (A7) $B_{O_1 \ldots O_m}$ (l1 ∨ … ∨ lp) → $B_{O_1 \ldots O_m}$ l1 ∨ … ∨ $B_{O_1 \ldots O_m}$ lp where li's are non complementary literals.

Inferences rules are :

- (Nec) ⊢ F ⟹ ⊢ $B_{O_1 \ldots O_m}$ F (if F is a propositional formula)
- (MP) ⊢ F and ⊢ (F → G) ⟹ ⊢ G

(A3) expresses that, if a literal l, belonging to topic T_k, is deducible from a database obtained by merging several databases according to some orders, then it remains deducible if we merge a new database i, considered as least reliable on topic T_k.

(A4) expresses that, if it is the case that a literal, belonging to topic T_k, belongs to the database i and if its negation is not deducible in the base obtained by merging several database with some orders, then it remains deducible if we merge the new database i considered as least reliable on topic T_k.

(A5) expresses that, if l is a literal of a given topic T_k, which is deducible in a base obtained by merging several bases with another base i, considered as least reliable on topic T_k, then either l is deducible in the merging of these bases or l is deducible in database i.

Bibliography

[AGM85] C. Alchourron, P Gardenfors, and D. Makinson. On the logic of theory change : Partial meet contraction and revision functions. *The journal of symbolic logic*, 50(2), 1985.

[CD88] F. Cuppens and R. Demolombe. Cooperative Answering: a methodology to provide intelligent access to Databases. In *Proc of Expert Database Systems*, 1988.

[CD92] S. Cazalens and R. Demolombe. Intelligent access to data and knowledge bases via users' topics of interest. In *Proceedings of IFIP Conference*, pages 245–251, 1992.

[CD94] L. Cholvy and R. Demolombe. Reasoning with information sources ordered by topics. In *Proceedings of Artificial Intelligence : Methods, Systems and Applications (AIMSA)*. World Scientific, Sofia, september 1994.

[CDJ94] L. Cholvy, R. Demolombe, and A. Jones. Reasoning about the safety of information : from logical formalization to operational definition. In *Lecture notes in Artificial Intelligence*, number 869. Springer-Verlag, 1994.

[Cho93] L. Cholvy. Proving theorems in a multi-sources environment. In *Proceedings of IJCAI*, pages 66–71, 1993.

[Cho94] L. Cholvy. A logical approach to multi-sources reasoning. In *Lecture notes in Artificial Intelligence*, number 808. Springer-Verlag, 1994.

[Cho95] L. Cholvy. Automated reasoning with merged contradictory information whose reliability depends on topics. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, Fribourg, July 1995.

[Cho96] L. Cholvy. Answering queries addressed to a group of deductive databases. In *Workshop on flexible query-answering*. Roskilde University, 1996.

[FKUV86] R. Fagin, G. Kupper, J. Ullman, and M. Vardi. Updating logical databases. *Advances in Computing Research*, 3, 1986.

[FUV83] R. Fagin, J.D. Ullman, and M. Vardi. On the semantics of updates in databases. In *ACM TODS*, pages 352–365, 1983.

[Gar88] P. Gardenfors. *Knowledge in Flux : Modeling the Dynamics of Epistemic States*. The MIT Press, 1988.

[GM92] P. Gardenfors and D. Makinson. Nonmonotonic inference based on expectations. *Artificial Intelligence*, to appear, 1992.

[GMN78] H. Gallaire, J. Minker, and J.M. Nicolas. *An overview and introduction to logic and database*. Plenum, 1978.

[GMN84] H. Gallaire, J. Minker, and J. M. Nicolas. Logic and databases : a deductive approach. *ACM Surveys*, 16(2), 1984.

[HC72] G. E. Hughes and M. J. Cresswell. *An introduction to modal logic*. Methren London and New York, 1972.

[JD94] A. Jones and R. Demolombe. Deriving answers to safety queries. In R. Demolombe and T. Imielinski, editors, *Non standard queries and answers*. Oxford University Press, 1994.

[KM91] H. Katsuno and A. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52, 1991.

[MS86] J. P. Martins and S. C. Shapiro. Theoretical foundations for belief revision. In *Reasoning about knowledge*, 1986.

[Neb89] B. Nebel. A knowledge level analysis of belief revision. In *Proc of KR'89*, 1989.

[Rei84] R. Reiter. *Towards a logical reconstruction of relational database theory*, chapter 8. Springer Verlag, 1984.

[Som94] Léa Sombé. A glance at revision and updating in knowledge bases. *International Journal of Intelligent Systems*, 9(1), 1994.

A Formal and Computational Account of Database Transactions

Raymond Reiter

Department of Computer Science
University of Toronto
and
The Canadian Institute for Advanced Research
`http://www.cs.toronto.edu/~cogrobo/`

Motivation and Background

- We consider only *transaction centred* databases, i.e. databases which evolve over time only by virtue of prespecified primitive *transactions*, whose purpose is to update the database with new information.
- Example: An educational database might have a primitive transaction specifically designed to change a student's grade.
  - This would normally be a procedure which, when invoked on a specific student and grade, first checks that the database satisfies certain preconditions (e.g., that there is a record for the student, and that the new grade differs from the old), and if so, records the new grade.
- This is a *procedural* notion. Transactions also *physically modify* the database (database materialization, progression).
- Ideally, we want a purely logical, non procedural *specification* of the entire evolution of a database. Why?
  - Clarity.
  - Generality, e.g. for databases with incomplete information like null values.
  - Proving properties of the database and its transactions (correctness, integrity constraint satisfaction, etc).

This Talk

- Propose to specify transaction-centred databases by appealing to various ideas from theories of actions in AI:
  - The situation calculus as a first order language for representing actions and their effects.
  - The frame problem and its solution for primitive actions.
  - Goal regression for planning.

- Specify transaction-centred databases in the situation calculus and how to deal with the frame problem.
- Database logs.
- Regression for query evaluation wrt a database log. Complexity of query evaluation.
- Mathematical induction for the situation calculus, and integrity constraints as inductive entailments of a database.
- Complex transactions and GOLOG, a new, situation calculus-based logic programming language for transactions.

Theory of database updates $\subseteq$ *high level robot control.*

Related Publications

The frame problem.

R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz (ed.) *AI and Math. Theory of Computation: Papers in Honor of John McCarthy*, Academic Press, 1991.

Formalizing databases in the sitcalc.

R. Reiter. On specifying database updates. *J. Logic Programming*, **25**, Oct. 1995.

Database materialization in the sitcalc.

F. Lin and R. Reiter. How to progress a database. *J. of Artificial Intelligence.* To appear.

GOLOG

H. Levesque, R. Reiter, Y. Lespérance, F. Lin and R. Scherl. GOLOG: a logic programming language for dynamic domains. *J. Logic Programming.* To appear.

Y. Lespérance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. A Logical Approach to High-Level Robot Programming – A Progress Report. *Control of the Physical World by Intelligent Systems*, Working Notes of the 1994 AAAI Fall Symposium.

Induction and integrity constraints.

R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence* **64**, 1993, pp. 337–351.

http://www.cs.toronto.edu/~cogrobo/

Specialised Integrity Checking by Combining Conjunctive Partial Deduction and Abstract Interpretation

(Abstract)

Michael Leuschel[1] and Danny De Schreye[2]

Departement Computerwetenschappen
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
{michael,dannyd}@cs.kuleuven.ac.be

A meta-program is a program which takes another program, the *object* program, as one of its inputs. An important issue in meta-programming is the representation of the object program. One approach is the *ground representation*, which encodes variables at the object level as ground terms. A lot of meta-programming tasks can only be written declaratively using the ground representation. This is e.g. the case for the application in [7, 8], where a simplification procedure for integrity constraints in recursive deductive databases is written as a meta-program. The goal is to obtain a pre-compilation of the integrity checking via partial deduction of the meta-interpreter. However, contrary to what one might expect, partial deduction is then unable to perform interesting specialisation and no pre-compilation can be obtained.

The crucial problem in [8] boils down to a lack of information propagation at the object level. The ground representation entails the use of an explicit unification algorithm at the object level. For the application of [8] we are interested in deriving properties of the result `R` of calculating `unify(A,B,S),apply(H,S,R)` where `S` is a substitution (the *mgu*) and `A,B,H` are (representations of) partially known atoms. In a concrete example we might have `A = status(X,student,Age)`, `B = status(ID,E,A)`, `H = category(ID,E)` and we would like to derive that `R` must be an instance of `category(ID',student)`. However it turns out that, when using an explicit unification algorithm, the substitutions have a very complex structure and current abstract interpretation methods, as well as current partial deduction methods alone, fail to derive the desired information.

We will present an elegant and powerful solution to this problem by combining two existing analysis schemes, each underlying a specific specialisation technique: the one of *conjunctive partial deduction* [6, 2] and the one of *more specific programs* [9]. By this approach we will be able to obtain the required specialisation, as well as an analysis which surpasses the precision of current abstract interpretation techniques (like e.g. [1], [3], [4],[9], [10]).

Bibliography

[1] J. Gallagher and D. A. de Waal. Fast and precise regular approximations of logic programs. In P. Van Hentenryck, editor, *Proceedings of the Eleventh International Conference on Logic Programming*, pages 599–613. The MIT Press, 1994.

[2] R. Glück, J. Jørgensen, B. Martens, and M. Sørensen. Controlling conjunctive partial deduction of definite logic programs. In *Proceedings of the International Symposium on Programming*

[1]Supported by the Belgian GOA "Non-Standard Applications of Abstract Interpretation"
[2]Senior Research Associate of the Belgian Fund for Scientific Research

Languages, Implementations, Logics and Programs (PLILP'96), LNCS, Aachen, Germany, September 1996.

[3] N. Heintze. Practical aspects of set based analysis. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 765–779, Washington D.C., 1992. MIT Press.

[4] G. Janssens and M. Bruynooghe. Deriving descriptions of possible values of program variables by means of abstract interpretation. *The Journal of Logic Programming*, 13(2 & 3):205–258, 1992.

[5] M. Leuschel and D. De Schreye. Towards creating specialised integrity checks through partial evaluation of meta-interpreters. In *Proceedings of PEPM'95, the ACM Sigplan Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 253–263, La Jolla, California, June 1995. ACM Press.

[6] M. Leuschel, D. De Schreye, and A. de Waal. A conceptual embedding of folding into partial deduction: Towards a maximal integration. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming JICSLP'96*, pages 319–332, Bonn, Germany, September 1996. MIT Press. Extended version as Technical Report CW 225, K.U. Leuven. Accessible via `http://www.cs.kuleuven.ac.be/~lpai`.

[7] M. Leuschel and B. Martens. Obtaining specialised update procedures through partial deduction of the ground representation. In H. Decker, U. Geske, T. Kakas, C. Sakama, D. Seipel, and T. Urpi, editors, *Proceedings of the ICLP'95 Joint Workshop on Deductive Databases and Logic Programming and Abduction in Deductive Databases and Knowledge Based Systems*, GMD-Studien Nr. 266, pages 81–95, Kanagawa, Japan, June 1995.

[8] M. Leuschel and B. Martens. Partial deduction of the ground representation and its application to integrity checking. In J. Lloyd, editor, *Proceedings of ILPS'95, the International Logic Programming Symposium*, pages 495–509, Portland, USA, December 1995. MIT Press. Extended version as Technical Report CW 210, K.U. Leuven. Accessible via `http://www.cs.kuleuven.ac.be/~lpai`.

[9] K. Marriott, L. Naish, and J.-L. Lassez. Most specific logic programs. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 909–923, Seattle, 1988. IEEE, MIT Press.

[10] A. Mulkers, W. Winsborough, and M. Bruynooghe. Live-structure data-flow analysis for prolog. *ACM Transactions on Programming Languages and Systems*, 16(2):205–258, 1994.

Classification and Specification of Update Problems[1] (abstract)

Ernest Teniente and Toni Urpí

Universitat Politècnica de Catalunya
Facultat d'Informàtica
Pau Gargallo 5
08028 Barcelona — Catalonia
{teniente/urpi}@lsi.upc.es

Deductive databases are based on deductive rules (views) which define the common knowledge shared by different users by allowing to deduce new facts (derived or view facts) from base facts explicitly stored in the database. Deductive databases use logic programming as a base language and generalize relational databases by overcoming the limitations of relational languages in the definition of complex views and constraints. This extension of the concept of relational database provides several advantages like facilitating the development of applications, preventing multiple development of programs with functional similarities or providing reasoning capabilities [GV89, GM92].

Among other components, deductive databases include an update processing system that provides the users with a uniform interface in which they can request different kinds of updates, i.e. updates of base facts, updates of derived facts, updates of deductive rules and updates of integrity constraints.

Several problems may arise when updating a deductive database [Abi88]. During last years much research has been devoted to different database updating problems like view updating [KM90, LS91, GL91, TO95, CHM95, Dec96], materialized view maintenance [CW91, HD92, UO92, GMS93, GM95], integrity constraints checking [BDM88, SK88, DW89, Küc91, Oli91, GCM+94], integrity constraints maintenance [CW90, ML91, Wüt93, TO95, Dec96] or condition monitoring [RCB+89, HCK+90, QW91].

Up to now, the general approach of the research related to deductive database updating problems has been to provide specific methods for solving particular problems. However, most of these methods are explicitly or implicitly based on a set of rules that define the changes that occur in a transition from an old state of a database to a new one. Therefore, these rules provide the basis of a framework for classifying and specifying these problems. We propose to use the event rules [Oli91], which explicitly define the insertions and deletions induced by an update, for such a basis.

We define two interpretations of the event rules: the upward interpretation and the downward one, which allow us to classify deductive database updating problems in two different types: upward problems and downward ones. The upward interpretation defines the changes on derived predicates induced by a transaction which consists of a set of changes on base facts. On the other hand, the downward interpretation defines the changes on base predicates needed to satisfy a given set of changes on derived predicates. The following figure illustrates this idea:

[1]This work has been partially supported by the PRONTIC CICYT program project TIC94-0512.

$$\begin{tikzcd}[row sep=large] \text{Derived predicates} \arrow[d, shift left=2, "\text{Downward interpretation}"] \\ \text{Base predicates} \arrow[u, shift left=2, "\text{Upward interpretation}"] \end{tikzcd}$$

For classifying and specifying deductive database updating problems in terms of the interpretations of the event rules, we need to endow a derived predicate with a concrete semantics. Many authors [DW89, RCB+89, Küc91] have proposed to express an integrity constraint, a view (materialized or not) and a condition to be monitored as a derived predicate. In all cases, the definition of a constraint, a view and a condition has the same form than a deductive rule, but an specific interpretation.

Thus, for instance, a derived predicate P defined by means of the rule $P(x) \leftarrow Q(x) \wedge \neg R(x)$, could be expressed as:

$$\begin{array}{l} \mathrm{Ic}(x) \leftarrow Q(x) \wedge \neg R(x) \\ \mathrm{View}(x) \leftarrow Q(x) \wedge \neg R(x) \\ \mathrm{Cond}(x) \leftarrow Q(x) \wedge \neg R(x) \end{array}$$

where Ic, View and Cond must be interpreted accordingly to their concrete semantics.

The upward and downward interpretation of the event rules corresponding to Ic, View and Cond allow us to classify the deductive database updating problems that have been addressed in the past years. Moreover, they allow us to identify some deductive database updating problems that, to our knowledge, have not been addressed up to now. This is summarized in the following table, where ιP refers to an insertion of predicate P, δP refers to a deletion of predicate P and T refers to a transaction consisting of a set of base fact updates.

		View	**Ic**	**Cond**
Upward Interpretation	ιP	Materialized view maintenance	IC checking	Condition monitoring
	δP		IC violation removal checking	
Downward Interpretation	ιP	View updating	Redundancy of ICs	Enforcing condition activation
	δP	View validation	Repairing inconsist. DB IC satisfiability	Condition validation
	$T, \neg\iota P$	Preventing side effects	IC maintenance	Preventing condition activation
	$T, \neg\delta P$		Maintaining DB inconsistency	

A more detailed explanation of the upward and the downward interpretations and of the concrete specifications of the above database updating problems in terms of these interpretations can be found in [TU95].

It is worth to mention that upward problems can be combined among them. All of them share a common starting-point (a transaction which consists of a set of base fact updates) and aim at the same goal (to define the changes on derived predicates induced by this transaction). The same reasons allow the combination of downward problems among them. Therefore, we can specify more complex upward and downward problems by considering possible combinations of the problems specified in the previous table.

Moreover, we could also combine upward and downward problems among them. Note that the result of the downward interpretation is the same than the starting-point of the upward interpretation, that is, a transaction which consists of a set of base fact updates. Therefore, we could first deal with downward problems and, immediately after, use the obtained result for dealing with the upward ones.

By considering only a unique set of rules for specifying all these problems, we want to show that it is possible to provide general methods able to deal with all these problems as a whole. Therefore, we could uniformly integrate view updating, materialized view maintenance, integrity constraints checking, integrity constraints maintenance, condition monitoring and other deductive database updating problems into an update processing system.

Finally, it is important to point out that we are not proposing a new method for change computation. What we propose is to use the event rules and their interpretations as a common framework for classifying and specifying deductive database updating problems. A particular implementation of these interpretations would produce a particular method for change computation.

Bibliography

[Abi88] S. Abiteboul. Updates, a new frontier. In *Proc. ICDT'88*, pages 1–18. Springer Verlag, 1988.

[BDM88] F. Bry, H. Decker, and R. Manthey. A uniform approach to constraints satisfaction and constraints satisfiability in deductive databases. In *Int. Conf. on Extending Database Technology (EDBT'88)*, pages 488–505, Venezia, 1988.

[CHM95] I.A. Chen, R. Hull, and D. McLeod. An execution model for limited ambiguity rules and its application to derived data update. *ACM Transactions on Database Systems*, 20(4):365–413, December 1995.

[CW90] S. Ceri, G. Gottlob, and L. Tanka. *Logic Programming and Databases.* Surveys in Computer Science. Springer Verlag, 1990.

[CW91] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proc. of the 17th VLDB Conference*, pages 577–589, Barcelona, 1991.

[DW89] S. Das and H. Williams. A path finding method for constraint checking in deductive databases. *Data and Knowledge Engineering*, 4:223–224, 1989.

[Dec96] H. Decker. An extension of SLD by abduction and integrity maintenance for view updating in deductive databases. In *Joint International Conference and Symposium on Logic Programming (JICSLP'96)*, Bonn, Germany, 1996. MIT Press.

[GL91] A. Guessoum and J.W. Lloyd. Updating knowledge bases II. *New Generation Computing*, 10:73–100, 1991.

[GCM+94] C García, M. Celma, L. Mota, and H. Decker. Comparing and synthesizing integrity checking methods for deductive databases. In *Proc. of the 10th ICDE*, pages 214–222, Houston, USA, 1994.

[GM92] J. Grant and J. Minker. The impact of logic programming on databases. *Communications of the ACM*, 35(3):66–81, 1992.

[GM95] A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques and applications. *Data Engineering*, 16(2):3–18, 1995.

[GMS93] A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In *Int. Conf. on Management of Data (SIGMOD)*, pages 157–166, 1993.

[GV89] G. Gardarin and P. Valduriez. *Relational Databases and Knowledge Bases.* Addison-Wesley, 1989.

[HCK+90] E.N. Hanson, M. Chaabouni, C. Kim, and Y. Wang. A predicate matching algorithm for database rule systems. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 271–280, Atlantic City, 1990.

[HD92] J.V. Harrison and S. Dietrich. Maintenance of materialized views in a deductive database: an update propagation approach. In *Workshop on Deductive Databases, JICSLP*, pages 56–65, Washington, D.C., 1992.

[KM90] A. Kakas and P. Mancarella. Database updates through abduction. In *Proc. of the 16th VLDB Conference*, pages 650–661, Brisbane, Australia, 1990.

[Küc91] V. Küchenhoff. On the efficient computation of the difference between consecutive database states. In *Deductive and Object Oriented Databases. Second International Conference DOOD'91*, volume 566 of *Lecture Notes in Computer Science*, pages 487–502, 1991.

[LS91] J. Larson and A. Sheth. Updating relational views using knowledge at view definition and view update time. *Information Systems*, 16(2):145–168, 1991.

[ML91] G. Moerkotte and P.C. Lockemann. Reactive consistency control in deductive databases. *ACM Transactions on Database Systems*, 16(4):670–702, 1991.

[Oli91] A. Olivé. Integrity checking in deductive databases. In *Proc. of the 17th VLDB Conference*, pages 513–523, Barcelona, Catalonia, 1991.

[QW91] X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. *IEEE Trans. on Knowledge and Data Engineering*, 3(3):337–341, September 1991.

[RCB+89] A. Rosenthal, S. Chakravarthy, B. Blaustein, and J. Blakeley. Situation monitoring for active databases. In *Proc. of the 15th VLDB Conference*, pages 455–464, Amsterdam, 1989.

[SK88] 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, 1988.

[TO95] E. Teniente and Olivé. A. Updating knowledge bases while maintaining their consistency. *The VLDB Journal*, 4(2):193–214, 1995.

[TU95] E. Teniente and T. Urpí. A common framework for classifying and specifying deductive database updating. In *Int. Conf. on Data Engineering (ICDE)*, pages 173–183, Taipei (Taiwan), 1995.

[UO92] T. Urpí and A. Olivé. A method for change computation in deductive databases. In *Proc. of the 18th VLDB Conference*, pages 225–237, Vancouver, Canada, 1992.

[UO94] T. Urpí and A. Olivé. Semantic change computation optimization in active databases. In *Proc. of the 4th Int. Workshop on Research Issues on Data Engineering — Active Database Systems (RIDE -ADS'94)*, pages 19–27, Houston, USA, 1994.

[Wüt93] B. Wüthrich. On updates and inconsistency repairing in knowledge bases. In *Int. Conf. on Data Engineering*, pages 608–615, Vienna, 1993.

A Minimalist's Approach to the Meaning of Database Updates and Active Rules

Carlo Zaniolo

Computer Science Department
UCLA

We pursue a simple framework for capturing database updates and Event-Condition-Action rules of active databases. The main idea is to treat the database history as the extensional database and view the current database as a set of intentional relations derived via frame-axioms rules. Reasoning on histories can be accomplished using Datalog$_{1S}$ programs having a syntactic structure that ensures (i) the existence of stable models, and (ii) efficient bottom up computations of these models. Different active rule semantics, including immediate, deferred and durable-change activation modes can be modeled effectively with this approach.

The result is a unified and simplified framework to reasoning with non-monotonic knowledge, non-deterministic constructs, database updates, active rules and a database histories.

Languages for Event-Based Reasoning in Active Databases and Time Series

Iakovos Motakis and Carlo Zaniolo

Department of Computer Science,
University of California,
Los Angeles, CA 90024, USA.
`lastname@cs.ucla.edu`

A powerful feature provided by new active database prototypes is the support for rules triggered by complex patterns of composite temporal events.

We proposes a logic-oriented approach to the definition of languages for active databases where rules can be triggered by composite events. The method is demonstrated via the design of the TREPL language, which improves on previous prototypes, such as ODE, Snoop and SAMOS, in several respects, including semantics and temporal aggregation.

We define the meaning of TREPL rules by mapping them into $Datalog_{1S}$, whose formal semantics is used to define the meaning of the original event expressions. A similar approach is then used in the definition of temporal aggregates. TREPL constructs for temporal aggregation are more powerful than those used by other active database languages; in fact, their expressive power matches that of time-series languages.

Nested Transactions in a Logical Language for Active Rules[1]

Bertram Ludäscher Wolfgang May[2] Georg Lausen

Institut für Informatik
Universität Freiburg
Am Flughafen 17, 79110 Freiburg
Germany

The need for a logically defined and intuitive semantics has been recognized as one of the major theoretical problems in the area of active databases. The active database manifesto, for example, requires as an essential feature that *"… rule execution must have a clear semantics, ie must define when, how, and on what database state conditions are evaluated and actions executed"* [DGG95]. Nevertheless, researchers continue to complain about the unpredictable behavior of active rules and the lack of a uniform and clear semantics.[3]

To overcome these difficulties, it has been suggested to use the logical foundations of deductive databases – with certain extensions – as a declarative semantics for active rules [Zan93, ZS94, Zan95, LL94, LHL95]. The main benefits of this approach are better understanding, maintainability and reasoning about rules when compared to the usual implementation-dependent operational semantics. However, as we will be shown, the existing "mergers" of active and deductive rules are not sufficient to model complex (trans)actions in a natural way, since they (i) lack structuring capabilities, and (ii) do not encapsulate the effect of semantically related rules. In particular, they neglect the fact that complex database transactions can be adequately modeled by *nested transactions* where the parent transaction may consult the outcome of subtransactions in order to perform its own complex tasks. As a solution, we propose the extension of our declarative framework for active rules Statelog [LHL95] by the concept of update *procedures*. Procedures execute as (closed) *nested transactions* whereas previous rule based approaches were limited to flat transactions. A Statelog procedure consists of a set of ECA-style Datalog rules each of which defines either

- a non-state-changing *query*, ie a (potentially recursive) view, or
- an *action*, ie
  - a *primitive update* request (insert, delete, modify),
  - a *complex update* request (procedure call), or
  - an *external action* to be issued by the database system, or
- a transaction control predicate.

System-immanent *frame* and *procedure rules* provide a declarative specification of state transitions and integrity preserving policies within the logical language without bothering the user with those problems.

The idea of using state terms to refer to different states in logical rules has come up several times, e.g. in *XY-Datalog* [Zan93, ZAO93], to allow a unified semantics for active and deductive rules, and in [KLS92, LL94] as a means to specify updates in a declarative way. Flat Statelog [LHL95], XY-Datalog, and the temporal query languages $Datalog_{1S}$ and *Templog* [Cho90, AM89, Bau95] are closely related, since they all extend Datalog by a linear state space. In contrast, our present

[3] *"The unstructured, unpredictable, and often nondeterministic behavior of rule processing can become a nightmare for the database rule programmer"* [AWH95]. See also [Wid94, WC96, PCFW95, DHW95, FT96].

approach uses a branching hierarchical state space (similar to that of $Datalog_{nS}$ [CI93] which does not deal with active rules and procedures, however). The presented Kripke semantics extends that of [LS93] which is now a special case restricted to flat sequential computations.

[Zan95] proposes a "transaction-conscious" stable model semantics to cope with the problem that occurs when *ephemeral changes* (changes whose effect is undone within the same transaction) trigger active rules. Thus, to avoid unintended behavior, only durable changes should be visible to active rules. In our approach this problem is solved in a different way by the concept of "atomically executing" procedures which encapsulate their changes until the end of transaction. Thus, only the net effect of a subtransaction may trigger rules in the calling transaction.

Transaction Logic $\mathcal{T}_{\mathcal{R}}$ [BK93, BK94] deals, on a high level of abstraction, with the phenomenon of state changes in logic databases and employs a powerful model theory and proof theory. Primitive updates (so-called *elementary transitions*) are not part of $\mathcal{T}_{\mathcal{R}}$, but a parameter which is supplied by a transition oracle. In contrast, Statelog semantics provides a complete specification of changes from primitive updates to complex transactions and has a standard logic programming and Kripke-style semantics. Both languages can be combined by "plugging in" Statelog procedures in the transition oracle of $\mathcal{T}_{\mathcal{R}}$.

The concept of nested transactions in Statelog is similar to that of *HiPAC* [DBB$^+$88, DBC96]. Statelog declarations *initial, always* and *final* allow execution of rules at specific points within a transaction and thus can be used in a similar way as *coupling modes* in HiPAC. E.g. integrity maintenance may be deferred until EOT by declaring the corresponding rules *final*.

The idea to structure rule sets using procedures or modules has already been introduced in the area of logic programming. E.g. [BMPT94, BT94] develop a modular design for logic programs including union, intersection, and encapsulation. However, they do not deal with active rules and state change, so their concept does not cover sequential composition, transactions etc.

[FT96] proposes *Extended ECA* rules as a common framework for a large number of existing active database systems and prototypes. In existing systems, the semantics of programs depends on the implicitly given operational semantics. These implicit assumptions are made apparent by encoding them in user-readable EECA rules. *Heraclitus[Alg,C]* [GHJ$^+$93, GHJ96], is an extension of C which incorporates the relational algebra and elevates deltas to be "first-class citizens" of the database programming language. It allows to combine deltas and to express hypothetical updates, however no logical semantics is given.

Related to our work are approaches dealing with updates in deductive databases. Often, the rule semantics depends on a certain evaluation strategy, e.g. [Abi88, AV91, SK96] (bottom-up), or [MW88, Che95] (top-down), whereas e.g. [MBM95] is – like Statelog – independent of a certain strategy. However, these works do not cover the ECA-rule paradigm of active databases or the concept of nested transactions. Although Statelog allows a very intuitive "bottom-up reading" of rules, evaluation may also be done top-down due to the presence of explicit state terms $[S]$ and $[S + 1]$. This is in contrast to approaches like [Abi88, AV91] or [MW88, Che95], which refer to different states only *implicitly.* Thus their semantics is more tied to *either* bottom-up *or* top-down evaluation, respectively.

Bibliography

[Abi88] S. Abiteboul. Updates, a new frontier. In *ICDT*, Springer LNCS 326, pp. 1–18, 1988.

[AM89] M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Comp.*, 8(3), 1989.

[AV91] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *JCSS*, 43, 1991.

[AWH92] A. Aiken, J. Widom, and J. M. Hellerstein. Behavior of database production rules: Termination, confluence, and observable determinism. In *SIGMOD*, 1992.

[AWH95] A. Aiken, J. Widom, and J. M. Hellerstein. Static analysis techniques for predicting the behavior of active database rules. *TODS*, 20(1):3–41, 1995.

[Bau95] M. Baudinet. On the expressiveness of temporal logic programming. *Information and Computation*, 117(2), 1995.

[BCP95] E. Baralis, S. Ceri, and S. Paraboschi. Run-time detection of non-terminating active rule systems. In Ling et al. [LMV95].

[BK93] A. J. Bonner and M. Kifer. Transaction logic programming. In D. S. Warren, editor, *ICLP*. MIT Press, 1993.

[BK94] A. J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Comp. Sci.*, 133, 1994.

[BMPT94] A. Brogi, P. Mancarella, D. Pedreschi, and F. Turini. Modular logic programming. *ACM TOPLAS*, 16(4):1361–1398, July 1994.

[BT94] A. Brogi and F. Turini. Semantics of meta-logic in an algebra of programs. In *LICS*, pp. 262–270, Paris, France, July 1994.

[Che95] W. Chen. Programming with logical queries, bulk updates and hypothetical reasoning. In B. Thalheim, ed., *Proc. of the Workshop Semantics in Databases*, Prague, 1995. TU Cottbus.

[Cho90] J. Chomicki. Polynomial time query processing in temporal deductive databases. *PODS*, 1990.

[CI93] J. Chomicki and T. Imieliński. Finite representation of infinite query answers. *TODS* 18(2), 1993.

[DBB$^+$88] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M. J. Carey, M. Livny, and R. Jauhari. The HiPAC project: Combining Active Databases and Timing Constraints. In *SIGMOD*, 1988.

[DBC96] U. Dayal, A. Buchmann, and S. Chakravarthy. The HiPAC Project. In J. Widom and S. Ceri, editors, *Active Database Systems: Triggers and Rules for Advanced Database Processing*, Morgan Kaufmann, 1996.

[DGG95] K. R. Dittrich, S. Gatziu, and A. Geppert. The active database management system manifesto: A rulebase of adbms features. In Sellis [Sel95].

[DHW95] U. Dayal, E. Hanson, and J. Widom. Active database systems. In W. Kim, ed., *Modern Database Systems: The Object Model, Interoperability, and Beyond*, Ch. 21. ACM Press, 1995.

[FT96] P. Fraternali and L. Tanca. A structured approach for the definition of the semantics of active databases. *TODS*, 1996. to appear.

[GHJ+93] S. Ghandeharizadeh, R. Hull, D. Jacobs *et al.* On implementing a language for specifying active database execution models. In *VLDB*, 1993.

[GHJ96] S. Ghandeharizadeh, R. Hull, and D. Jacobs. Heraclitus: Elevating deltas to be first-class citizens in a database programming language. *TODS*, 1996. To appear.

[KLS92] M. Kramer, G. Lausen, and G. Saake. Updates in a rule-based language for objects. *VLDB*, 1992.

[KU96] A. P. Karadimce and S. D. Urban. Refined triggering graphs: A logic-based approach to termination analysis in an active object-oriented database. In *12th ICDE*, 1996.

[LHL95] B. Ludäscher, U. Hamann, and G. Lausen. A logical framework for active rules. In *Proc. 7th Intl. Conf. on Management of Data (COMAD)*, Pune, 1995. Tata McGraw-Hill. ftp://ftp.informatik.uni-freiburg.de/documents/reports/report78/report78.ps.gz.

[LL94] G. Lausen and B. Ludäscher. Updates by reasoning about states. In *2nd Intl. East-West Database Workshop*, Workshops in Computing, Klagenfurt, Austria, 1994. Springer.

[LMV95] T. W. Ling, A. O. Mendelzon, and L. Vieille, editors. *DOOD*, Springer LNCS 1013, 1995.

[LS93] G. Lausen and G. Saake. A possible world semantics for updates by versioning. In *Proc. of 4th Workshop on Modelling Database Dynamics*, Volkse, 1993. Springer.

[MBM95] D. Montesi, E. Bertino, and M. Martelli. Transactions and updates in deductive databases. Technical Report 2, Dipartimento di Scienze dell'Informazione, Università di Milano, 1995.

[MW88] S. Manchanda and D. S. Warren. A logic-based language for database updates. In J. Minker, ed., *Foundations of Deductive Databases and Logic Programming*, pp. 363–394. 1988.

[MZ95] I. Motakis and C. Zaniolo. Composite temporal events in active database rules: A logic-oriented approach. In *4th DOOD*, LNCS 1013, 1995.

[PCFW95] N. W. Paton, J. Campin, A. A. A. Fernandes, and M. H. Williams. Formal specification of active database functionality: A survey. In Sellis [Sel95].

[Sel95] T. K. Sellis, editor. *Proc. of the 2nd Intl. Workshop on Rules in Database Systems (RIDS)*, Athens, Greece, 1995, Springer LNCS 985.

[SK96] E. Simon and J. Kiernan. The a-rdl system. In Widom and Ceri [WC96], Chapter 5.

[WC96] J. Widom and S. Ceri, editors. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann, 1996.

[Wid94] J. Widom. Active databases. In M. Stonebraker, ed. *Readings in Database Systems, 2nd edition.* Morgan Kaufmann, 1994. Introduction to Chapter 4.

[Zan93] C. Zaniolo. A unified semantics for active and deductive databases. *Proc. of the 1st Intl. Workshop on Rules in Database Systems (RIDS)*, Edinburgh, 1993. Springer.

[Zan95] C. Zaniolo. Active database rules with transaction conscious stable model semantics. In Ling et al. [LMV95].

[ZAO93] C. Zaniolo, N. Arni, and K. Ong. Negation and Aggregates in Recursive Rules: the $\mathcal{LDL}++$ Approach. In S. Ceri, K. Tanaka, and S. Tsur, eds., *DOOD*, Springer LNCS 760, 1993.

[ZS94] C. Zaniolo and R. Sadri. A simple model for active rules and their behavior in deductive databases. *Proc. 2nd ICLP Workshop on Deductive Databases and Logic Programming*, Santa Margherita Ligure, Italy, 1994.

From Hypothetical Reasoning to Reasoning about Actions in Modal Logic Programming

Laura Giordano

Dipartimento di Informatica - Università di Torino
C.so Svizzera 185 - 10149 Torino, ITALY
laura@di.unito.it

In [9] we have proposed a logic programming language *CondLP*$^+$, that is an extension of *N_Prolog* [7] which provides capabilities for hypothetical and counterfactual reasoning. In *CondLP*$^+$, a database containing constraints can be hypothetically updated by adding new facts. A revision mechanism allows to deal with the inconsistencies that may arise by removing the information responsible for them.

In this language, as in *N_Prolog*, hypothetical implications $D \Rightarrow G$ (with G a goal and D a set of hypotheses) may occur both in goals and in clause bodies. For an implication $D \Rightarrow G$ to succeed from a program P, G must succeed from the new program obtained by hypothetically updating P with D. As a difference with N_Prolog, in *CondLP*$^+$ a program may contain integrity constraints, and, hence, inconsistencies may arise, when updates are performed.

In *CondLP*$^+$ hypothetical updates are sets of atoms. Moreover, a program consists of a protected part Δ, containing a set of clauses and integrity constraints, and a removable part, consisting of a set of atomic formulas partitioned in different priority classes.

When an inconsistency arises, a *revision* of the program is needed. Revision does not affect the protected part, which is permanent. Instead, it modifies its removable part, by overruling the atoms which are responsible for the inconsistency and have the least preference. Such atoms are not removed from the database, but they are temporarily made non visible.

We represent the revisable part of the program as a list of sets of atoms $L = S_1, \ldots, S_n$, assuming that atoms in S_i have higher priority than atoms in S_j, for all $j < i$.

As an example of program in our language, consider the following one, adapted from [7], containing the set Δ of clauses and constraints:

$$\begin{aligned} &british\_citizen(X,T) \leftarrow born\_in\_UK(X) \wedge father(X,Y) \\ &\qquad\qquad \wedge\ (alive(Y,T) \Rightarrow british\_citizen(Y,T)) \\ &british\_citizen(Z,T_2) \leftarrow british\_citizen(Z,T_1) \wedge less\_than(T_1,T_2) \\ &\qquad\qquad \wedge\ alive(Z,T_2) \\ &dead(X,T) \wedge\ alive(X,T) \rightarrow \bot, \end{aligned}$$

and the list L containing a single set of facts S:

$$\begin{aligned} S = \{&born\_in\_UK(bob), father(bob,tom), british\_citizen(tom,1950), \\ &dead(tom,1984), less\_than(1950,1984)\}. \end{aligned}$$

The reading of the first rule is "One is a British citizen at a certain time if (a) (s)he is born in UK and (b) if his/her father were alive at that time, then he would be a British citizen." The reading of the second rule and of the integrity constraint is obvious.
Consider the goal

$$G = british\_citizen(bob, 1984).$$

G succeeds from the program P consisting of the set of clauses and constraints Δ above and the list of facts L. In fact, the subgoals $born\_in\_UK(bob)$ and $father(bob, tom)$ succeed from P. The subgoal

$$alive(tom, 1984) \Rightarrow british\_citizen(tom, 1984)$$

succeeds from P since $british\_citizen(tom, 1984)$ succeeds from the revised program obtained from P by adding $alive(tom, 1984)$. Notice that the query

$$alive(tom, 1984) \Rightarrow dead(tom, 1984)$$

fails from P, since $dead(tom, 1984)$ is overridden by the addition of *alive(tom, 1984)*.

Since the removable part of the program may contain several atoms with the same priority, in general, more than one revision of the program can be obtained. Hence it is quite natural to make use of abduction to characterize the alternative revisions through different abductive solutions.

For this language a logical characterization has been defined by making use of a (preaty standard) modal logic, in which hypothetical updates are represented by modalities. In fact, modal logic allows very naturally to represent state changes as transition, through the accessibility relation of Kripke structures. In particular, updates are represented by modalities and an hypothetical implication $D \Rightarrow G$, where D is a set of atoms, is represented by the modal formula $[D]G$.

Moreover, to capture the non-monotonic behaviour of the language. an abductive construction is introduced. More specifically, persistency axioms are introduced to allow persistence of assumptions, if they are not inconsistent with other assumptions with higher priority. This construction has strong similarities with Eshghi and Kowalski's abductive semantics for negation as failure [5] (and it can be regarded as a generalization of the stable models semantics [10] to the modal setting).

For this language a goal directed proof procedure has been defined within the argumentation framework [4], which provides a nice setting in which an abductive semantics and a procedure for computing it can be defined, since it is parametric with respect to both the logic and the abducible symbols adopted.

Besides providing update capabilities, by giving the highest preference more recent updates, the language also allows to deal with a statically defined preference ordering among assumptions. Hence, $CondLP^+$ can be used to perform several types of defeasible reasoning, and, in particular to deal with exeptions. Though the language does not explicitely provide negation as failure, normal logic programs with negation as failure can be easily represented within it, as long as priorized normal default and, to some extent, also non-normal defaults.

In defining $CondLP^+$ we have chosen a specific policy to deal with updates. In particular, more recent updates are given higher priority with respect to the previous ones. This has been motivated by some applications to defeasible and counterfactual reasoning. In principle, however, other policies might be used, to perform different types of reasoning, and the language can be regarded as being parametric with respect to this choice.

The problem of dealing with an ordered revisable database, is strongly related to the problem of reasoning with multiple, possibly contradicting, sources of information, ordered according to their reliability. Such a problem has been tackled by several authors (see, for instance, [1, 2]). In particular, in [1] the problem of combining first order theories and resolving inconsistencies among them has been studied, and two different approaches are devised: the *bottom-up* approach and the *top-down* approach.

The revision policy adopted in CondLP^+ is an example of the top-down approach: rather than removing an inconsistency as soon as it arises, the proof procedure maintains the consistency of logical consequences of the database by making use of the priority among facts. The proof procedure keeps track of the (possibly inconsistent) sequence of updates that have been performed, and it reasons on that to determine the facts which hold in the current situation. In essence, CondLP^+ resolves the inconsistencies by considering the assumptions from those with the highest preference to those with the lowest preference (top-down). A different (bottom-up) approach, that has been followed in *CondLP* [8], is that of removing facts responsible for an inconsistency each time the inconsistency is detected. As observed in [1], the top-down approach is more informative then the bottom-up one.

The top-down approach can be suitably used to perform default reasoning, and in particular, to deal with exceptions. However, there are other applications for which the bottom-up approach is better suited. In fact, the bottom-up approach allows to model more naturally the dynamic change from one state to another one. This is true for instance, when we have to deal with the problem of "reasoning about actions".

In the language CondLP^+, a modality $[S]$ which represents an update by a given set of facts S, can be ragarded as a special kind of action, the action of adding S to the database. This actions is implicitely defined within the language, through some logical axioms.

In order to deal with more general kinds of actions, the language must be extended by allowing actions definitions (action laws) of the form:

$$Preconditions \rightarrow [Action]Postcondition,$$

where each action is represented by a modalities. In this context, a sequence of modalities $[a_1]\ldots[a_n]$ represents the state which can be reached by executing actions $a_1, \ldots, a_n$. Moreover, we can represent the fact that an observation Obs holds in the state $[a_1]\ldots[a_n]$ through the formula $[a_1]\ldots[a_n]Obs$

Example. *Shooting problem.* We define a domain description containing the following set of action laws and observations on the initial state:

$$\begin{array}{lll} \Pi : & \Box(true \rightarrow [load]loaded) & \Box(loaded \rightarrow [shoot]\neg alive) \\ & \Box(true \rightarrow [shoot]\neg loaded) & \\ Obs : & alive & \neg loaded \end{array}$$

The adoption of the same approach followed in the definition of the language CondLP^+ leads to the definition of an abductive semantics for reasoning about actions in which abductive assumptions are used to deal with persistency.

The adoption of a modal language is common with other proposals for modeling actions [3, 6, 12], which are based on Dynamic Logic [11].

Bibliography

[1] C. Baral, S. Kraus, J. Minker, V. S. Subrahmanian. Combining knowledge bases consisting of first-order theories. *J.Automated Reasoning*, vol.8, n.1, pp. 45–71, 1992.

[2] L. Cholvy. Proving theorems in a multi-source environment. In *Proc. IJCAI-93*, pages 66–71, Chambery, 1993.

[3] G. De Giacomo, M. Lenzerini. PDL-based framework for reasoning about actions. In *LNAI 992*, pages 103–114, 1995.

[4] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming. In *Proc. IJCAI93*, pages 852–857, 1993.

[5] K. Eshghi and R. Kowalski. Abduction compared with negation by failure. In *Proc. 6th ICLP*, pages 234–254, Lisbon, 1989.

[6] L. Fariñas del Cerro and A. Herzig. Interference logic = conditional logic + frame axiom. In *Int. J. of Intelligent Systems*, 9(1):119–130, 1994.

[7] D. M. Gabbay and N. Reyle. NProlog: An extension of Prolog with hypothetical implications.I. *Journal of Logic Programming*, (4):319–355, 1984.

[8] D. Gabbay, L. Giordano, A. Martelli, and N. Olivetti. Conditional logic programming. In *Proc. 11th ICLP*, Santa Margherita Ligure, pages 272–289, 1994.

[9] Dov Gabbay, Laura Giordano, Alberto Martelli, and Nicola Olivetti. Hypothetical updates, priority and inconsistency in a logic programming language. In *Proc. LPNMR-95*, LNAI 928, pages 203–216, 1995.

[10] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proc. ICLP-90*, pages 579–597, Jerusalem, 1990.

[11] D. Harel. First order dynamic logic in *Extensions of Classical Logic, Handbook of Philosophical Logic II*, pp. 497–604, 1984.

[12] C. Schwind. A Logic Based Framework for Action Theories. In *Proc. TSLLC*, Jean-Jacques Levy and Zurab Khasidashvil (eds.), to appear in CSLI-series Stanford,USA, 1996.

(A)Mnesia — An Industrial Deductive DBMS with Transactions and Distribution

Hans Nilsson and Claes Wikström

Computer Science Laboratory
Ericsson Telecom AB
S-125 26 Stockholm, Sweden
{hans,klacke}@erix.ericsson.se

1 Introduction

Mnesia is a multiuser Distributed DBMS specially made for industrial telecommunications applications written in the symbolic language ERLANG [1].

In telecom applications there are needs different from the features provided by the DBMS we have seen so far. The applications we now see implemented in the ERLANG language[1] needs:

1. fast soft real-time key-value lookup
2. complicated non-realtime queries mainly for operation and maintenance
3. distributed data due to distributed applications
4. high fault-tolerance
5. dynamic re-configuration

In the current first version of Mnesia we mainly use well-known algorithms. In following versions we will start experimenting with alternative solutions. The most important by now is to broaden the group of real users as a basis for further experiments with real applications.

The rest of this paper is organized as follows: Chapter 2 is an ERLANG introduction and chapte 3 is an overview of Mnesia. The query evaluation is outlined in chapter 4. Some usage of Mnesia is given in chapter 5 and some notes of the future and conclusions are given in chapters 6 and 7.

2 Erlang

ERLANG is a symbolic, functional language intended for large real-time applications mainly in the telecommunication area.

Functions may have many "clauses" and the correct one is selected by pattern matching. The visibility is controlled with explicit export declarations in modules.

Processes are lightweight and created explicitly. They communicate by asynchronous message passing. Messages are received selectively using pattern matching. When running on UNIX there is only one UNIX process namely the ERLANG node. This node has its own internal process handling, so the ERLANG processes should not be mistaken for UNIX processes.

Erlang nodes can be run on different types of machines and operating systems and the Erlang processes can communicate across a network with processes on other nodes. This does not influence the program writing, except if the message passing time is critical. No extra protocol specifications are necessary. The message passing is competitive with other network communication methods[7].

There are two kind of support for error recovery in the language: exceptions and special exit messages propagated to linked processes when a process dies. Those messages can be caught by the other processes to activate some recovery plan, for instance restart the dying process.

Erlang programs communicate with hardware and programs written in other languages by *ports*. The ports behave similar to processes in that they can receive messages from processes and send messages back to them.

3 Mnesia

The DBMS Mnesia[4] provides:

- primary memory storage for fast lookup with optional disk backup
- a deductive query language
- a fast program interface for very simple queries
- distributed transactions and query evaluation
- crash recovery
- replication of data on separate nodes
- location transparency: applications are written without knowledge where and how the tables are located

Data are organized in tables. A table has a set of properties, for example:

- on which node it is located
- where it is replicated (= a list of zero or more nodes)
- persistence (if it is only in RAM or is replicated on disk)
- if the system should maintain one or more indices

The table definitions are collected in a schema which may be altered in runtime.

Operations on a database are grouped in transactions which are distributed over the network. All completed transactions are logged as well as the 'dirty' operations where a time-critical application can bypass the transactions.

The query construction is integrated with the Erlang language by a construction called list comprehension. This removes the impedance mismatch that a solution like Embedded SQL would give with a language like Erlang .

There are also query shells for interactive quering by humans. Presently there is only one with Prolog-like syntax. The same syntax is used for rule definitions.

All of Mnesia is written in Erlang . However there is some low level support in C for efficient table hashing.

4 Query Evaluation

The majority of our users queries are non-recursive. Therefore we have chosen to handle them separately with an old but efficient relational DBMS technique called operators[3]. Those queries are compiled and optimized using standard methods like partial evaluation and goal reordering guided by statistics about the database.

The operator technique maps extremely well onto the ERLANG processes and messages — as a spin-off effect queries are automatically parallelized when run on a multi-cpu computer.

Recursive queries or rather recursive subgraphs of the query graph are evaluated by SLG[6, 5, 2] resolution. This is implemented by a naive straight-forward interpreter. No optimization at all is yet performed on those parts.

One simple comparison with Oracle showed that Mnesia is only slightly slower than Oracle for an typical office-application query. No systematic measurements are yet performed.

5 Projects

Mnesia is currently used in a large development project involving more than hundred persons. Unfortunatly they are not advanced users but it gives valuable feedback anyway. In another small project Mnesia is used in an advanced project with several processors for performance and fault tolerence.

6 The Future

Primarily we will evaluate Mnesia with the applications that are now being written. Performance comparisons with commercial DBMS as well with modern research DBMS will be done.

In the area of recursive processing there are two obvious continuations namely making an *efficient* SLG implementation and to optimize recursive queries.

We will try other transaction and locking algorithms, specially real-time algorithms.

Mnesia has no integrity constraints today, so this is an extension that we will do.

Some people ask for an SQL-interface. This is needed and probably also an ODBMC interface to be able to access the database from external PC-applications.

7 Experiences and conclusions

It is possible to write a full industrial DBMS in the symbolic high level language ERLANG with only a fraction of the manpower normally required.

Bibliography

[1] Joe Armstrong, Robert Virding, , Wikström Claes, and Mike Williams. *Concurrent Programming in ERLANG.* Prentice Hall, second edition, 1995.

[2] A. W. Chen and D. S. Warren. Query evaluation under the well-founded semantics. In *Proc. ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys.*, page 168, Washington, DC, May 1993.

[3] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[4] Hans Nilsson, Torbjörn Törnkvist, and Claes Wikström. Amnesia — a distributed real-time primary memory DBMS with a deductive query language (poster). In *Proceedings of the Twelfth Conference on Logic Programming*, 1995.

[5] D. S. Warren W. Chen, T. Swift. Efficient computation of queries under the well-founded semantics. Technical Report 93-CSE-33, Southern Methodist University, 1993.

[6] David S Warren and Weidong Chen. Towards effective evaluation of general logic programs. Technical report, Computer Science Department (SUNY) at Stony Brook, 1993.

[7] Claes Wikström. Distributed computing in Erlang. In *First International symposium on Parallel Symbolic computation*, Sep 1994.

Logic Programming: Modularity and Revisions (Work in Progress)

Thomas Eiter   Georg Gottlob   Helmut Veith

Information Systems Department
Technical University of Vienna
Paniglgasse 16, A-1040 Wien
Austria
`(eiter|gottlob|veith)@dbai.tuwien.ac.at`

In the programming methodology of logic programming it is important to have the possiblity of building program modules, which can be used to develop a complex program. These program modules act like subprograms, and are used to compute intermediate results by the main program, which calls the subprograms on specific input data.

While it has been realized that modular programming is important [1, 2, 3], the investigations on modularities have focussed so far on the possibility to split a logic program LP into smaller programs LP_1, LP_2 in a stratified manner, where $LP1$ is a kind of subprogram, so that inference from P can be done by first doing inference on LP_1 and then, utilizing the results obtained, do inference on LP_2, cf. [1]; similar, that generating a model of LP can be done by first generating a model of LP_1, which is extended to a model of LP_2 in order to obtain a model of LP cf. [2, 3].

In our talk we argue that logic subprograms can be seen as a certain kind of *generalized quantifiers*, and propose a semantics based on stable models which allows for reusable logic subprograms.

We generalize the notion of dependency graph to the case of modular programs, and identify syntactic properties of modular logic programs akin to stratification (*semistratified, call recursion free, call independent programs*).

For those systems, we investigate the role of different call modalities, the nesting depth of modules, and the notion of *monotone subprograms*.

Building on results from finite model theory, we show normal forms for several of those classes, and relate the syntactic classes to well-known complexity classes from the polynomial hierarchy. As an immediate consequence, we see that disjunctive logic programming is just as expressive as modular programming with nesting depth 2.

A natural application stems from revision programming: In a recent series of papers [4, 5, 6], a concept of revision programming which is closely related to the stable model semantics was developed. We investigate the use of revision programs as subprograms, as well as the evaluation of subjunctive queries in the proposed framework of modular logic programming.

Bibliography

[1] J. Schlipf. The Expressive Powers of Logic Programming Semantics. *Journal of Computer and System Sciences*, 51(1):64–86, 1995. Abstract in Proc. PODS 90, pp. 196–204.

[2] T. Eiter, G. Gottlob, and H. Mannila. Adding Disjunction to Datalog. In *Proceedings of the Thirteenth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-94)*, pages 267–278, May 1994.

[3] V. Lifschitz and H. Turner. Splitting a Logic Program. In *Proceedings ICLP-94*, pages 23–38, Santa Margherita Ligure, Italy, June 1994. MIT-Press.

[4] V.W. Marek and M. Truszczyński. Revision Programming, database updates and integrity constraints. In *Proceedings of the ICDT'95*, LNCS 893, pages 368–382, 1995.

[5] V.W. Marek and M. Truszczyński. Revision specifications by means of revision programs. In *Logics in AI. Proceedings of JELIA'94*, LNAI 1994.

[6] T.C. Przymusiński and H. Turner. Update by means of inference rules. InProceedings of LPNMR'95, LNCS 928, pages 156–174, 1995.

Logical Specification of Bulk Updates and Sequential Updates (Extended Abstract)

Carl-Alexander Wichert and Burkhard Freitag

Universität Passau
Fakultät für Mathematik und Informatik
D-94030 Passau, Germany
{wichert,freitag}@fmi.uni-passau.de
http://www.uni-passau.de/~freitag/

1 Main Ideas

In the field of deductive databases the static semantics and the evaluation of queries are well understood. However, there is no consense how to handle dynamic behaviour appropriately. Although there exist many different concepts integrating updates into the logical languages, these approaches – regarded in isolation – cannot capture the semantics used in the database community.

We present[1] a specification language for complex updates which allows to combine procedural update methods with set-oriented updates, such that the conventional database update operations are subsumed. To generalize the syntax and semantics of deductive rules we allow basic update requests to appear in a rule body (goal). In this case, the rule does not define a part of a view but a part of a transaction, the latter being named by the head predicate.

For example, suppose a base relation *Sal* (addressed by a predicate *sal*) is given which contains the salaries of several persons. One can easily define an operation $raise\_sal\_10$ performing a complex salary update for a specified person P, e.g. increasing the salary by 10 percent:

$$\begin{array}{l} raise\_sal\_10(P) :- \\ \quad sal(P,X),\ Y = X * 1.1, \\ \quad DEL\ sal(P,X),\ INS\ sal(P,Y). \end{array}$$

To actually invoke an update operation, the corresponding (sub-)query must be asked. The following query will apply the complex update operation $raise\_sal\_10$ on the employee *miller*, i.e. it will initiate a transaction:

$$:-\ raise\_sal\_10(miller).$$

We have voted for the updates-as-subgoals paradigm which is also used in Transaction Logic [4, 5] and U-Datalog [3], because complex update operations can be implemented modularly and hierarchically. This is also helpful for integrating object-orientation or operation hiding. Furthermore, the updates-as-subgoals concept is suitable for the integration with nested transactions models [1, 9, 10].

The paradigm where basic updates appear in the rule body unfortunately has some own disadvantages that we address in our work. As goal evaluation is logically based on the resolution tree, the natural extension of the conventional proof procedure to update operations performs *tuple-oriented updates*. Every path of the augmented resolution tree yields a substitution of the free query variables and defines a transition to a new database state. It is easy to collect all answers

[1] A full paper describing our concepts has been published [11] and is also available in the WWW.

of a query, but it is not at all clear how to produce a global database transition depending on a set of existing resolution paths. Thus *set-oriented updates (bulk updates)* which are often needed in classical databases and which are expressable in SQL are not specifiable without adequate modifications. In our approach we define a non-immediate update semantics in which update requests are handled as terms. During the *evaluation phase*, the basic updates which are suitable to perform the (abstract) operation (called as a query) are computed, in a subsequent *update phase* the collected basic update requests are actually performed and change the database state. In this setting, bulk updates can be achieved by aggregation over successful resolution paths. The bulk quantor # is used to specify, for instance, a salary update for all known persons:

$$raise\_sal\_of\_all \text{ :- } \# \; P \; [\; raise\_sal\_10(P) \;].$$

The computation of a global bulk update $\# \; X \; [\; G \;]$ from the updates of successful local resolution paths is performed in analogy to the computation of an aggregate value (sum, average, etc.) from a subquery. The aggregation merges update requests of one successful path for each successful binding of X in the subtree w.r.t. the interior goal G.

If we do not specify a bulk update using the quantification construct, multiple resolution paths will lead to non-determinism. This fits with the logical consense that auxiliary variables in a rule body are implicitly existentially quantified and it is sufficient to find one solution. For example, omitting the bulk quantor in the previous example we would specify that a non-deterministically chosen person gets her salary updated:

$$raise\_sal\_of\_one \text{ :- } raise\_sal\_10(P).$$

The second main problem, which we address in our work, is the description of *sequential update processes*. Within one transaction, a query or another update operation may want to see the effects of a previous update. Adopting the Prolog-like semantics, where rules are resolved in a left-to-right fashion and updates are immediately performed, results in an adhoc approach for procedural updates with a significant loss of declaritivity and optimization potential. Thus a more sophisticated treatment of intermediate transitions has to be developed. In our approach we explicitly denote sequences of operations by a construct $[\; G_1 : G_2 \;]$, which means intuitively that the data access of goal G_2 is to be done after the one of goal G_1. In contrast, the classical conjunction written as G_1, G_2 has the semantics that both goals are to be evaluated in the same state and must be compatible [2]. However, in neither case the instance of the extensional database is modified immediately. Because there may exist different hypothetical states, we handle them by hypothetical reasoning, until we commit the transaction.

The following query raises the salary of *miller* twice. The second occurrence of the subgoal $raise\_sal\_10(miller)$ is evaluated in a hypothetical state resulting from the former one:

$$\text{:- } [\; raise\_sal\_10(miller) \;:\; raise\_sal\_10(miller) \;].$$

Due to our non-immediate update semantics, in each hypothetical state we have to keep a set of basic update requests representing the transition from the original state to the current one. Whenever the resolution of a (sub-)goal leads to a new state, the corresponding update requests have to be adequately merged with the update requests collected before. As known from transaction theory, we require that sets of update requests are consistent, i.e. the update requests are compatible. For instance, not both existence and non-existence of a tuple must be contained in the same set of update requests. With explicit sequentialization, however, this is not a problem, since the new assertion, e.g. deletion of a tuple, will invalidate a former conflicting assertion, e.g. the insertion of the same tuple.

It should be mentioned that the semantics of our update concept guarantees the atomicity of transactions. The database state is only changed, after a consistent set of basic update requests has been computed. These updates will always lead to the final state of the transaction. The atomicity of the update phase can easily be achieved by classical transaction techniques on the extensional database.

2 Implementation

The implementation of the update concept relies on a meta-level approach using the deductive database system *LOLA* [7]. The extended logic programs (including the queries) are transformed into pure logic programs. Due to the declarative semantics the new programs can be evaluated in a bottom-up style after magic set transformation. It is also worth noticing that *LOLA*'s deductive database engine is used for the transaction processing overhead, i.e. the determination of the appropriate sets of update requests, as well as for the computation of classical intensional relations. The resulting updates are performed by a built-in operation having a side effect.

In one transformation step, rules defining hypothetical predicates are generated. The main idea is that the extensional relations are *not* changed during the evaluation process, and, as a consequence, references to intermediate states are handled by *hypothetical reasoning*. For each (intensional or update) predicate p that might be evaluated in an updated context, a hypothetical predicate *hyp_p* with augmented arity is produced. The new position serves for handling an input parameter referring to a list of basic update requests that characterize the transition from the original to the current hypothetical state.

In another step, the collection and merging of basic update requests is explicitly encoded into the program. All update predicates (also the hypothetical ones) get an augmented arity. This time, the new position serves for the result of the transaction. To handle update composition (sequential and normal conjunction) and basic update atoms adequately, built-in subgoals for list manipulation and consistency checking are integrated. The bulk update construct is computed using an aggregation function.

3 Future Work

The underlying resolution semantics of our concept is not suitable for set-oriented evaluation. Successive bulk updates are only expressible using bulk quantification, but this quantification makes it impossible to pass values to other subgoals. Further, the description of bulk updates is not very intuitive, as it requires an explicit construct (see e.g. $raise\_sal\_of\_all$). It would rather be acceptable for a user to *specify* where she allows non-determinism than to get it per default.

The consistency constraints are another point of discussion. Up to now, we require compatibility, which means that no complementary basic update requests, i.e. insert and delete of the same tuple, must occur on one single path or on aggregated paths. This seems reasonable for updates belonging to one single path, as this fits in with the logical conjunction. In the path aggregation case, it may be argued that this point of view is too restrictive. One future research topic will be to consider all update specifications as transaction specifications which are modularly built from subtransactions and to exploit the classical and advanced theories of serializability [2, 10] to find weaker constraints than compatibility.

Transaction properties must be integrated into the concept, that is, concurrency control, persistence, and recovery are necessary. The update requests and their hypothetical execution have to be managed efficiently. The scheduling of update operations should be made more flexible, e.g. by incorporating the concepts of open nested transactions [10]. The programmer should explicitly specify begin and end of transactions and provide a specification of the commutativity relation of higher-level operations.

The formal semantics of our approach gives an inductive characterization of valid facts and their side effects. Maybe, it is useful to reformulate the semantics based on a well-known temporal or dynamic logic [6, 8].

Bibliography

[1] C. Beeri, H. J. Schek, and G. Weikum. Multi-level transaction management, theoretical art or practical need? In *Proc. Int. Conf. on Extending Database Technology, Venice, Italy*, pages 134–154, 1988.

[2] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publishing Company, 1987.

[3] E. Bertino, G. Guerrini, and D. Montesi. Towards deductive object databases. *Theory and Practice of Object Systems*, 1(1):19–39, 1995.

[4] A. J. Bonner and M. Kifer. An overview of Transaction Logic. *Theoretical Computer Science*, 133:205–265, 1994.

[5] A. J. Bonner and M. Kifer. Concurrency and communication in Transaction Logic. In *Proc. Joint Int. Conf. and Symp. on Logic Programming, Bonn, Germany*, pages 142–156, 1996.

[6] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.

[7] B. Freitag, H. Schütz, and G. Specht. *LOLA* – a logic language for deductive databases and its implementation. In *Proc. 2nd Int. Symp. on Database Systems For Advanced Applications, Tokyo, Japan*, pages 216–225, 1991.

[8] D. Kozen and J. Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 789–840. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.

[9] J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing.* MIT Press, 1985.

[10] G. Weikum and H. J. Schek. Concepts and applications of multilevel transactions and open nested transactions. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[11] C. A. Wichert and B. Freitag. Logical specification of bulk updates and sequential updates. In *Proc. 4th Workshop on Deductive Databases and Logic Programming in conjunction with the Joint Int. Conf. and Symp. on Logic Programming, Bonn, Germany*, 1996.

Inference Techniques for Revising and Updating Logic Databases (Abstract)

Knut Hinkelmann

DFKI GmbH – German Research Center for Artificial Intelligence
Postfach 2080, 67608 Kaiserslautern
Germany
hinkelma@dfki.uni-kl.de

In a logic database the knowledge is composed of the extensional database (EDB) – a set of ground facts on base relations – and the intensional database (IDB) – a set of rules. Changing a logic databases can affect either the EDB, the IDB, or both. Sombé distinguishes between *revision* and *update* of logic databases [4]. An update helps a knowledge base to keep up-to-date with an evolving world while a revision is a belief change where information is added or removed about a static situation of which not everything is known.

In [2] it is assumed that "the rules are generally regarded to be complete and non-defeasible and thus should not be changed, whereas the EDB is regarded as containing incomplete and defeasible knowledge about the world." Consequently, *update* operations should only affect the base relation in the EDB (cf. [1]).

However, there are applications where this assumption does not necessarily hold. This is particularly true for logic databases containing rules that are detected by data mining. For instance, Inductive Logic Programming (ILP [3]) can be applied to learn rules from data. Since the rules depend on the training data it cannot be assumed that the learned rules are complete and non-defeasible. Thus, *revision* of a logic database can affect both the EDB and the IDB.

In my talk I will consider the problem of revising a logic database where a rule or a set of rules is recognized to be too general. First, I will present a transformation approach for integrity checking that suggests hypotheses for revision operations in order to satisfy a violated integrity theory (this transformation was developed in our project by Michael Sintek). Then I will show that ILP techniques can be applied not only for learning new rules but also for specializing already existing rules. I will present some possibilities for improving ILP, e.g. modifying the specialization operators or avoiding redundant derviation steps for satisfiability and consistency tests by reusing intermediate results.

Bibliography

[1] Hendrik Decker. An extension of SLD by abduction and integrity maintenance for view updating in deductive databases. In *International Joint Conference and Symposium on Logic Programming, Proceedings*, 1996.

[2] A. C. Kakas and P. Mancarella. Database updates through abduction. In *Proceedings of the 16th International Conference on Very Large Databases (VLDB)*, 1990.

[3] N. Lavrač and S. Džeroski. *Inductive Logical Programming: Techniques and Applications*. Ellis Horwood Limited, 1994.

[4] Léa Sombé. A glance at revision and updating in knowledge bases. *International Journal of Intelligent Systems*, 9:1–27, 1994.

Hypothetical Query Processing for Workflow Management (Extended Abstract)

Kazumasa Yokota[1] Takeo Kunishima[2]

yokota@kuis.kyoto-u.ac.jp kunishi@is.aist-nara.ac.jp

Hideyuki Nakanishi

nuka@kuis.kyoto-u.ac.jp

1 Introduction

One of the major features of data and knowledge in knowledge information processing is *partiality of information*. That is, in many cases, a database schema (or a predicate with a fixed number of arguments) cannot be defined in advance. Attributes have only indefinite values, such as constraints, and the data itself might be ambiguous or inconsistent. Partiality of information means that no one individual can specify *all* important information in sufficient detail in advance, and that there may be inconsistency among information sources. Especially, we cannot imagine a global schema of all information sources in wide-are network. Under environments such as knowledge information processing, a means of representing partial information and reasoning such partiality is indispensable.

We proposed thinking-experimental environments for partial information databases by using a deductive object-oriented database language $\mathcal{QUIXOTE}$ with facilities of conditional query processing and abductive reasoning[5]. Such features are very effective for debugging databases and knowledge-bases in applications with incomplete information such as natural language processing[4] and legal reasoning[3]. Furthermore, we have designed and developed a heterogeneous, distributed, cooperative problem solving system, called *Helios*[1], which can be considered as an extension of $\mathcal{QUIXOTE}$ in a distributed environment: that is, *Helios* allows to define a $\mathcal{QUIXOTE}$ database as an agent and supports cooperation among agents.

In this paper, we extend this approach for workflow management in distributed wide-area networks as follows:

1) To migrate heterogeneous information sources related to a specific workflow, we take step-by-step migration as in trial-and-error type processing.
2) To represent a global schema of related workflow databases, we employ a rule-based language consisting of production rules and definite clauses.
3) To extract necessary information, we apply conditional and abductive query processing.

In this paper, we discuss their extended thinking-experimental environments.

[1]Department of Information Science, Kyoto University, Yoshida-Honmachi, Sakyo, Kyoto 606-01, JAPAN

[2]Graduate School of Information Science, Advanced Institute for Science and Technology, Takayama, Ikoma, Nara 630-01, JAPAN

2 Thinking-Experimental Environments in $\mathcal{QUIXOTE}$

$\mathcal{QUIXOTE}$ is a constraint logic programming language with subsumption constraints, by which various object-orientation concepts such as object identity, type hierarchy, property inheritance, and method are embedded. A $\mathcal{QUIXOTE}$ database (or a program) is defined as a triple (S, M, R), where S, M, R correspond to the definitions of subsumption relations, submodule relations, and rules, respectively. S defines object subsumption hierarchy in the form of lattice, by which property inheritance is defined, M defines inter-module relation, by which rue inheritance is defined, and R defines a set of facts (extensional objects), a set of rules (intensional objects), and a set of modules.

2.1 What is Partial?

There are many points to be considered regarding a partial information database. First, we must recognize that database information may be partial or incomplete:

1) Necessary definitions may be lacking for any element of (S, M, R).
2) Facts or rules may be ambiguous or indefinite. Mutually inconsistent hypotheses may arise.
3) A $\mathcal{QUIXOTE}$ object may be incompletely defined: each of indefinite numbers of extrinsic properties may be specified as a constraint.

For example, the analogy between legal precedents (with incomplete descriptions) is indispensable for legal reasoning, while hidden knowledge (undescribed knowledge) is important in natural language processing. In such areas, complete description cannot be expected.

A second point that must be considered, is the special query processing requirements when dealing with this kind of partial information database:

1) What will be returned in response to this query if a user inserts some information (candidates for lacking information) as hypotheses?
2) What information, required to successfully answer this query, is missing from this database?
3) When mutually inconsistent data or knowledge is stored separately in different modules within a database, which is better for this application?
4) When hypotheses, generated during query processing, are used by successive queries, how are they controlled and where are they stored?
5) Why is this answer (the constraint) returned as an answer to this query? What knowledge is used for the answer?

A query in $\mathcal{QUIXOTE}$ is of the form, "*query* if *hypotheses*", where *hypotheses* corresponds to 1). Such hypotheses may be incrementally inserted into the database for 4). A *query* may be issued to different modules for 3). An answer in $\mathcal{QUIXOTE}$ is of the form, "if *assumptions* then *answer* because *explanation*", where *assumptions* corresponds to 2), *explanation* corresponds to 5), and, as in CLP, *answer* is a set of subsumption constraints. These two kinds queries are related as follows:

$\texttt{query}_1$:	?-*query*.	
$\texttt{answer}_1$:	if *assum* then *answer*.	
$\texttt{query}_2$:	?-*query* if *assum*.	%*assum* in $\texttt{query}_1$ is used as hypotheses of $\texttt{query}_2$.
$\texttt{answer}_2$:	(unconditionally) *answer*.	% The same answer is returned without *assum*.

2.2 Conditional Query Processing and Abductive Reasoning

In general, conditional query processing in a database DB is defined as reasoning in a database $DB \cup H$, where H is a set of hypotheses. Just as a $\mathcal{QUIXOTE}$ database is defined as (S, M, R), a hypothesis also consists of a triple (H_S, H_M, H_R), where H_S, H_M, and H_R are a set of hypotheses for S, M, and R. A query Q with a hypothesis (H_S, H_M, H_R) to a database (S, M, R) is equivalent to a query Q without hypotheses to a database $(S \cup H_S, M \cup H_M, R \cup H_R)$. Hypothesis insertion is done before processing query Q. Even if such insertion requires the reconstruction of the lattice of object terms and the submodule graph of modules, no logical problems arise. The only possible penalty is in terms of performance efficiency.

In the sequence of queries, such hypotheses are incrementally inserted into a database. To control such insertions, *nested transaction* is introduced into $\mathcal{QUIXOTE}$: that is, even if a database is reorganized by hypotheses, the original image is recovered by *rollback* operations.

First, consider a simple database consisting of a rule and a fact:

$$john/[age = A] \Leftarrow john/[twin\_brother = paul], paul/[age = A]^3;;$$
$$paul/[age = 30].$$

For a query ?-$john/[age = X]$, what answer is expected? Although *paul*'s age is specified, there is no fact stating that *john* and *paul* are *twin_brother*. Without making any assumptions, the query fails. However, as we focus on the partiality of the information, the lack of information suggests an assumption be taken. So, in $\mathcal{QUIXOTE}$, the answer is that if $john.twin\_brother \cong paul$ then X is 30, that is, unsatisfied constraints of other objects' extrinsic properties in bodies are assumed.

In logic programming, finding a lack of information or unsatisfiable subgoals corresponds to *abduction*, that is, hypothesis or explanation generation. A rule in $\mathcal{QUIXOTE}$ can be represented in the form of $o_0|C_0 \Leftarrow o_1, \cdots, o_n \mid\mid A \cup C$, where object terms, $o_1, \cdots, o_n$, are considered existence checks for corresponding objects, C_0, a set of dotted constraints of o_0, regarded as being assertional constraints, and C, a set of variable constraints, considered as being a set of constraints to be satisfied. A is considered as being the constraints of other objects' extrinsic properties. In $\mathcal{QUIXOTE}$, only A is taken as an assumption, that is, even if body constraints about dotted terms are not satisfied, they are regarded as being a conditional part of an answer. Although A and C are disjoint, when variables in C are bound by dotted terms during query processing, constraints with the variables in C are moved into A. If the subsumption relation between object terms is taken as an assumption, it might destroy the soundness of the derivation because it affects property inheritance and does not guarantee results in the former derivation. Even if there is insufficient data in a database, $\mathcal{QUIXOTE}$ explicates hidden knowledge.

Additionally, we might obtain different answers from a single query if that query is put to different modules. In such cases, the derivation process of an answer is returned including an *explanation*, if necessary. By referring to this explanation, users can verify which rules have been used for the answer.

3 Framework of Workflow Management

Here we take workflow management as an example for hypothetical query processing in a distributed environment. *Workflow management system*, one of computer supported cooperative work (CSCW), is proposed to support a stereotyped work flow by computers systems such as e-mail and Web browser. In the systems, there are many information sources such as definitions of workflows, their progress reports, intermediate resources, and various pre- and post-conditions of works. To manage them efficiently, database management systems are indispensable, although most works are based on repository systems. Here we propose a workflow data model[2] and discuss hypothetical query processing on them in next section.

3.1 Workflow Model

In our workflow model[2], there are several important concepts: activity objects, workflow templates and workflow instances, an execution model consisting of P-box and B-box, and workflow base.

An *activity object*, corresponding to a unit of work, is recursively defined as follows:

$$a = (I, O, P, S), \text{ where } I = \{i_1, \cdots, i_n\}\ (n \geq 1), O = \{o_1, \cdots, o_m\}\ (m \geq 1), P = \text{string}, S = \text{WFT}$$

Intuitively, I is an input of a, O is an output of a, P is an agent of a, who is responsible for this work, and S is a set of sub-activity objects (workflow template) to support a, to each of which a sub-work is dispatched by P.

A *workflow template* (WFT) is a set of activity objects $a_1, \cdots, a_n$ $(n \geq 0)$: $W = \{a_1, \cdots, a_n\}$ or $(W, \{a_1, \cdots, a_n\})$ where W is the identifier. A WFT must satisfy two conditions, *closeness* and *consistency*, to define some workflows:

1) Closed WFT: $a_i \in W$ for any $a = (I, O, P, S) \in W$ and any $a_i \in S$, .
2) Consistency: we define two kinds ordering:

   (a) $a_1 \preceq_1 a_2 \overset{def}{=} O_1 \supseteq I_2$.

   (b) $a_1 \preceq_2 a_2 \overset{def}{=} a_2 = (I, O, P, S) \wedge a_1 \in S$.

   In both orderings, the transitive law holds. The consistency of a WFT is defined as $\forall a.\ a \not\preceq_1 a \wedge a \not\preceq_2 a$.

There are two kinds of flows in a WFT:

1) Horizontal "$\Longrightarrow$": $a_1 \Longrightarrow a_2 \overset{def}{=} O_1 \supseteq I_2$

   $\{a_1, a_2, \cdots, a_n\} \Longrightarrow a \overset{def}{=} (\sum_{i=1}^{n} O_i \supseteq I) \wedge \forall j \neg (\bigcup_{i=1}^{n} O_i \Leftrightarrow O_j \supseteq I)$

2) Vertical flow "$\Leftrightarrow\!\!\!\rightarrow$": $a \Leftrightarrow\!\!\!\rightarrow a_i \overset{def}{=} a_i \in S$ where $a = (I, O, P, S)$

If $a_1 \Leftrightarrow\!\!\!\rightarrow a_2$, then a_2 is a *child* of a_1 and a_1 is a *parent* of a_2.

For example, given four activity objects a_1, a_2, a_3, a, where $O_1 = \{o_1, o_2\}, O_2 = \{o_2, o_3\}, O_3 = \{o_3, o_1\}$, $I = \{o_1, o_2, o_3\}$, then the following workflows are defined:

$$\{a_1, a_2\} \Longrightarrow a,\ \{a_2, a_3\} \Longrightarrow a,\ \{a_3, a_1\} \Longrightarrow a$$

Generally, a WFT defines multiple workflows, where a workflow which is nor included in another, is called *maximal*.

The execution model of a WFT consists of two models, *P-box* and *B-box*, each of which corresponds to horizontal and vertical flows.

A horizontal flow defines the following production rules:

1) If $a_1 \Longrightarrow a_2$, then $a_2 \Leftarrow a_1$.
2) If $a_1 \Longrightarrow a_3$ and $a_2 \Rightarrow a_3$, then $a_3 \Leftarrow a_1, a_2$.
3) If $a_1 \Longrightarrow a_2$ and $a_1 \Rightarrow a_3$, then $a_2 \Leftarrow a_1$ and $a_3 \Leftarrow a_1$.

The right hand side of $\Leftarrow$ is condition pats, while the left hand side is an action part. This rule is evaluated forwardly: if all end conditions in the condition part are received, then the action part is activated. Such rules are stored in *P-box.*

On the other hand, a vertical flow is represented as a set of definite clauses: if an activity object a generates child objects, $a_1, a_2, \cdots, a_n$ dynamically, then the execution model is represented as in $a \Leftarrow a_1, a_2, \cdots, a_n$, which is also dynamically generated. Intuitively, if all constraint and binding information are propagated from a to $a_1, a_2, \cdots, a_n$, and all child objects executed successfully, then a succeeds. Such rules are stored in *B-box.*

A workflow is executed as combination of P-box and B-box and controlled by its interpretor.

3.2 Workflow Base

We must instantiate all arguments to make WFT effective for real works. Such instantiation corresponds to an assignment based on subsumption hierarchy of message objects and agents. An instantiated WFT is called its *workflow instance* (WFI), which is also a WFT. Subsumption relation between $W = (I, O, P, S)$ and $W' = (I', O', P', S')$ is defined by an assignment θ as follows:

$$W\theta = W' \quad \Leftrightarrow \quad \theta = \{I/I', O/O', P/P', S/S'\} \wedge I' \sqsubseteq_S I \wedge O' \sqsubseteq_S O \wedge P' \sqsubseteq_S P \wedge S' \sqsubseteq_S S$$

where "$\sqsubseteq_S$" corresponds to Smyth order.

All WFTs (and WFIs) are stored in a *workflow base* (WFB), which is defined as follows:

definition:	$W = (W', \theta, \{a_1, \cdots, a_m\})$
	$a_1 = (I_1, O_1, P_1, S_1), \cdots, a_n = (I_n, O_n, P_n, S_n)$
execution model:	P-box: $\cdots$
	B-box: $\cdots$

where there are two kinds of integrity constraints:

- Consistency of ordering among WFT are kept based on ordering among message objects and agents.
- All WFTs are closed and consistent.

Operations to a WFB are defined as an extension of ordinary relational algebra, although the closure property is not kept.

- `select` a set of activity objects by some conditions from a specified WFT or a WFB.
- `get` a set of maximal `workflow` from a set of activity objects in a specified WFT or a WFB.
- get `ancestors` or `descendants` (a set of activity objects) from a specified activity object.
- get `general` or `special` workflows of a specified workflow.
- generalize or `specialize` a WFT by a specified assignment.

Furthermore, we prepare several operations of multiple WFTs, such as `grouping`, `flatten`, `split`, and `concat`.

4 Discussions on Views for Workflow Management

Workflow management is usually considered over networks. Even if it is designed for a stereotyped work, we must consider its applicability to another work, its similarity with others, that is, possibility of their merging, and update according to changes of environments, including agents, system configuration, and various constraints. Further multiple WFBs over networks had been sometimes referred for improvement of a work: for example, if a workflow of review process for some conference could be abstracted, then it can be applied to another conference. New assignment may be generated from multiple WFTs for assigning better agents. Discussions about partiality in Section 2.1 can be also applied to this application: conditional query processing and abductive reasoning are indispensable as knowledge information processing in $\mathcal{QUIXOTE}$.

As in ordinary databases, we can define views, corresponding to the set of operations, from a WFB. However, under discussions of partial information databases, hypotheses and lacking information should be included in the definition of views: that is, the target may be the union of a given database and additional information such as hypotheses. Further, as we can get generalized or specialized WFTs from a WFB, they are over conventional concepts of views.

Considering merging or fusing of multiple WFBs, we can consider better situation than the cases in other heterogeneous databases, because the number of objects are restricted and each WFT/WFI is considered to be relatively well designed and not so frequently updated, although there might be much differences between clever and fool agents.

Specific features of workflow management is summarized: 1) support of dynamic assignment and generation of activity objects, 2) comprehensive management of various information sources including intermediate precesses, and 3) Treatment of many possible exception occurred in workflows.

Hypothetical query processing designed for advanced applications, based a deductive object-orientation paradigm for advanced applications, is effective also for new applications such as workflow management and there are many possibilities for their extensions to support specific features for each application.

Bibliography

[1] A. Aiba, K. Yokota, and H. Tsuda, "Heterogeneous Distributed Cooperative Problem Solving System Helios and Its Cooperation Mechanisms", *Int. J. of Cooperative Information Systems*, pp.369-385, vol.4, no.4, Dec., 1995.

[2] T. Kunishima and K. Yokota, "Workflow Management Based on Object-Base", *submitted for publication*, 1996.

[3] C. Takahashi and K. Yokota, "Constructing a Legal Database on $\mathcal{QUIXOTE}$", *Proc. ADC'95*, Adelaide,Australia, Jan. 30,31, 1995.

[4] S. Tojo, H. Tsuda, H. Yasukawa, K. Yokota, and Y. Morita, "$\mathcal{QUIXOTE}$ as a Tool for Natural Language Processing", *Proc. TAI'93*, Boston, USA, Nov. 8-11, 1993.

[5] K. Yokota, T. Nishioka, H. Tsuda, and S. Tojo, "Query Processing for Partial Information Databases in $\mathcal{QUIXOTE}$", Proc. TAI'94, New Orleans, Nov. 6-9, 1994.

Multi-sorted Preferences in the Issue of Belief Change

Anna Gomolińska

Warsaw University, Bialystok Division, Poland
anna.gom@math.uw.bialystok.pl

In the presented framework the issue of changing a belief state is addressed from the perspective of multi-sorted preferences upon given information. Most of the research in the area of modelling a belief change is influenced by the joint work of Alchourrón, Gärdenfors, and Makinson who developed the theory of belief revision which we will refer to as the AGM model of a belief change as well. Keeping with the usual convention, information and beliefs are represented by logical formulas. The key-concept of the AGM model is a belief set which is a set of formulas of a given language closed under logical consequence. For simplicity, let us consider belief change problems in the case of the classical propositional logic. Hence, a set of formulas K is a belief set provided that $K = Cn(K)$. Changing a belief state is understood in terms of certain transformations, namely, expansion, revision, and contraction of a given belief set when facing a new piece of information. Due to so-called Principle of Minimality, changes of a belief set should be as minimal as possible.

Expansion of a belief set K by a formula α takes place in case K is Cn-consistent with α (i.e., $Cn(K \cup \{\alpha\}) \neq FOR$ where FOR denotes the set of all formulas) and returns $Cn(K \cup \{\alpha\})$ as the new belief set. We deal with a revision process in case K is Cn-inconsistent with a new formula α which has to be incorporated into K as a new belief. Roughly speaking, *revision* of K by α consists in selecting a number of the most appropriate subsets of K being maximal Cn-consistent with α, taking their set-theoretical intersection, say K', and finally, accepting $Cn(K' \cup \{\alpha\})$ as the new belief set. Hansson proposed a certain modification of revision called *consolidation*. When revising K by α, α is given the highest priority with respect to other beliefs whereas no particular priority is ascribed to it in case of consolidation. As a consequence, maximal Cn-consistent subsets of $K \cup \{\alpha\}$ are considered. The last but not least is *contraction* performed on a belief set K when a formula α having just been in K has to be removed. Briefly speaking, one of the most appropriate subsets K' of K such that $\alpha \notin Cn(K')$ is selected and its Cn-closure serves as the new belief set. Contraction and revision are interdefinable transformations.

The transformations mentioned above can be expressed in terms of a certain pre-ordering relation (i.e., reflexive and transitive) on the set of formulas called *epistemic entrenchment* relation. Epistemic entrenchment, $\leq_e$ is a preference relation determining which of two given formulas should be kept if we had to remove one of them. The well-known rationality postulates for epistemic entrenchment state that for any formulas α, β, γ, and any belief set K we have that:

(EE1) if $\alpha \leq_e \beta$ and $\beta \leq_e \gamma$, then $\alpha \leq_e \gamma$;

(EE2) if $\beta \in Cn(\{\alpha\})$, then $\alpha \leq_e \beta$;

(EE3) $\alpha \leq_e \alpha \wedge \beta$ or $\beta \leq_e \alpha \wedge \beta$;

(EE4) if $K \neq FOR$, then $\alpha \notin K$ iff for every β, $\alpha \leq_e \beta$;

(EE5) if for every β, $\beta \leq_e \alpha$, then α is a tautology.

1 Multi-sorted preference relations

Belief sets being closed under logical consequence suffer from the logical omniscience problem. In the processes of revision, consolidation, and contraction we need usually more than only one criterion, i.e., Cn-consistency, when choosing among the best alternatives. Although the epistemic entrenchment postulates seem to be rational (maybe exept for $(EE3)$), the preferences determined by them differ essentially from those based on probability of events and statistical evidence. These and similar observations gave rise to develop our approach. Indeed, taking into account additional available information about candidates for beliefs, for instance, credibility, utility, importance, relevance, etc., may be helpful in coping with logical omniscience and choosing the best alternative.

The concept of a sort of preference is taken for granted. In particular, such properties as being logically entailed (or derivable), credibility, utility, importance, and relevance of information are seen as sorts of preference worth considering. Formulas can be put into pre-orders called preference relations with respect to a given sort of preference in many different ways. For instance, let cn denote sort "being logically derivable". One of its preference relations, $\leq_{cn}$, can be determined by postulates $(EE1)$ and $(EE2)$ where subscript e is replaced by cn. Apparently, $\leq_e$ described by postulates $(EE1)$–$(EE5)$ may be seen as a preference relation of sort cn.

In case of the sort referred to as credibility, cr, preferences can be determined by comparing degrees of credibility of information or, in other words, formulas. By a degree of credibility of a formula α, $cr(\alpha)$, we mean here a quantitative value like a number, an interval or, generally, a set of numbers describing how credible a given piece of information represented by α is. Having obtained degrees of credibility we can be put them into a pre-order in many different ways. Next, a preference relation on formulas, $\leq_{cr}$, reflecting the pre-order on degrees of credibilities determined previously is defined as follows. Let $cr(\alpha)$, $cr(\beta)$ be degrees of credibility of α and β, respectively, and $\leq$ be a pre-ordering relation on these degrees.

$$\alpha \leq_{cr} \beta \quad iff \quad cr(\alpha) \leq cr(\beta).$$

Example. Consider a group of k experts (sources of information). Suppose that m experts state that α holds and n experts state that α does not hold. None of them can state both α holds and does not hold, i.e., $m + n \leq k$. The degree of credibility of α, $cr(\alpha)$, can be defined as $[\frac{m}{k}, 1 \Leftrightarrow \frac{n}{k}]$. Degrees of credibility obtained as above can be put, for instance, into the following pre-order. Let $cr(\alpha) = [a, b]$ and $cr(\beta) = [c, d]$.

$$cr(\alpha) \leq cr(\beta) \;\; iff \;\; (a = c \;\; and \;\; b = d) \;\; or \;\; b \leq c.$$

Suppose $1 < \ldots < m$ are sorts of preference ordered linearly. For each $i = 1, \ldots, k$, there is a preference relation, $\leq_i$, associated with sort i, and relations $\approx_i$, $<_i$ defined as usual. We can define a multi-sorted preference relation on formulas, $\preceq$, with respect to the given sorts as follows.

$$\begin{aligned} \alpha \preceq \beta \quad iff \quad & \alpha <_k \beta \vee (\alpha \approx_k \beta \wedge \alpha <_{k-1} \beta) \vee \ldots \\ \vee \quad & (\alpha \approx_k \beta \wedge \ldots \wedge \alpha \approx_2 \beta \wedge \alpha \leq_1 \beta). \end{aligned} \tag{0.1}$$

Relations $\approx$ and $\prec$ are defined customarily.

2 Belief sets

With each sort of preference one can associate an appropriate consequence operator in Tarski's sense. For instance, operator Cn may be associated with sort cn. With sorts like credibility we can associate operators defined as follows. Let i be a sort of preference, X be a set of formulas. Define

$$\nabla_i(X) = \{\alpha \mid (\exists\beta \in X)\beta \leq_i \alpha\}. \tag{0.2}$$

∇ is a consequence operator in Tarski's sense.

Now suppose that C_i is a consequence operator for sort i where $i = 1, \ldots, k$. Observe that $C_{1\ldots k} = \bigcap_{i=1}^{k} C_i$ is a consequence operator as well.

The notion of Cn-consistency of sets of formulas may be in a natural way adapted to the case of any consequence operator. Let C be any consequence operator, X, Y be sets of formulas, and α, β be formulas. α and β are called opponents to each other in case $\alpha \leftrightarrow \neg\beta$ is a classical tautology. Set X is called C-consistent (resp., absolutely C-consistent) in case $C(X)$ contains no opponents (resp., $C(X) \neq FOR$). X is C-consistent with Y iff $X \cup Y$ is C-consistent.

Let I be a set of initial beliefs referred to as a set of premises as well. By a belief set given I we mean $C_{1\ldots k}(I)$. Belief sets defined in this way may or may not suffer from logical omniscience because consequence operators other than Cn may produce finite sets of sentences. Let us note that if I is C_i-consistent (resp., absolutely C_i-consistent) then it is $C_{1\ldots k}$-consistent (absolutely $C_{1\ldots k}$-consistent) as well.

3 Restoring of consistency

The problem of restoring consistency of a set is of the highest priority among the problems addressing changing a belief state. We copy with it each time when revisions, consolidations, and contractions are performed on sets of beliefs. Without going into details, restoring of an intended form of consistency of a set, say X, (for instance, C-consistency, absolute C-consistency, or C-consistency with another set) consists in selecting one of consistent subsets of X. Usually, maximal elements among all the consistent subsets are considered as appropriate alternatives due to Principle of Minimality. Then we need a selection strategy, as effective as possible, to obtain a unique set. (This set may be none of the alternatives but it may be obtained from all or some of them by set-theoretical operations). The well-known strategies for restoring Cn-consistency when performing contractions on belief sets in AGM approach are: Maxi-choice, Full-meet Contraction, and Partial-meet Contraction.

Suppose we are given a method, S_i, of selecting maximal C_i-consistent alternatives (the best available to us so far) for each $i = 1, \ldots, k$. What we need is a strategy of selection with respect to the multi-sorted case. The starting point is ordering of available methods. For instance, we can put the same order as in case of sorts of preference or we can order the methods respectively to other criteria, for instance, effectiveness. Next, we decrease the number of alternatives as much as possible using given methods subsequently with respect to the order defined previously.

4 Belief change

In our approach the notions of expansion, revision, and consolidation of a belief set are generalised to the case of a multi-sorted consequence operator and preference relations on formulas. The process of belief change takes the following form. Let I be a given set of premises and α be a new formula integrated with the actual belief set $C_{1\ldots k}(I)$. Suppose that formulas logically equivalent are preferred equally. Suppose that I and $\{\alpha\}$ are $C_{1\ldots k}$-consistent. Let BS denote the new belief set.

CASE 1. (EXPANSION) I is $C_{1\ldots k}$-consistent with $\{\alpha\}$. $BS = C_{1\ldots k}(I \cup \{\alpha\})$.

CASE2. I is not $C_{1\ldots k}$-consistent with $\{\alpha\}$, i.e., there is an opponent, β, to α in $C_{1\ldots k}(I \cup \{\alpha\})$.

(2A) $\alpha \prec \beta$. $BS = C_{1\ldots k}(I)$.

(2B) (BELIEF REVISION) $\beta \prec \alpha$. Then restore $C_{1\ldots k}$-consistency of I with $\{\alpha\}$ first. Suppose I' is the selected subset of I. $BS = C_{1\ldots k}(I' \cup \{\alpha\})$.

(2C) (CONSOLIDATION) $\alpha \approx \beta$ or they are incomparable. Then restore $C_{1\ldots k}$-consistency of $I \cup \{\alpha\}$ first. Suppose I' is the selected set. $BS = C_{1\ldots k}(I')$.

5 Summary

In our approach it is argued that considering multi-sorted preferences rather than epistemic entrenchment relations only may be helpful in overcoming the problem of logical omniscience and in obtaining a more decisive method of choosing a candidate for the new belief set.

Structuring and Change in SLDNF-Proofs — Principles and Applications –Abstract–

Stefan Decker

Institut AIFB
Universität Karlsruhe (TH)
`decker@aifb.uni-karlsruhe.de`

1 Introduction

Changes in deductive databases usually mean updates of these databases: new facts and rules are added, others are deleted. We aim at understanding the effects of an update by a logic programming approach: by structuring and analysing a SLDNF-proof the exact consequences of an update relative to a proof can be determined. The approach is interesting for the field of integrity checking in deductive databases: better conditions for an update violating the proof of an integrity constraint can be defined. In recent approaches to integrity checking the existence but not the structure of a proof has been exploited.

Analysing a SLDNF-proof is a difficult task: a SLDNF-proof can be defined as a set of SLDNF-refutations and finitely failed SLDNF-trees (for an exact definition see below). Similar subproofs are very distributed in these proofs: usually in different SLDNF-trees resp. refutations. For these reasons we are first structuring the SLDNF-proof by using an alternative data structure.

1.1 Notations and Definitions

Most notions are taken from [4]. Because there is no standard definition of a SLDNF-proof, a definition is given:

Definition 1 (SLDNF-Proof)
Let P be a logic program, $\leftarrow G$ be a goal clause. A SLDNF-proof is the least set $\mathbf{S}$ with the following properties:

- a SLDNF-refutation for $P \cup \{\leftarrow G\}$ is element of S.
- for each selected negative literal $\neg A$ in a SLDNF-refutation in $\mathbf{S}$ exists exactly one finitely failed SLDNF-tree for $P \cup \{\leftarrow A\}$ in $\mathbf{S}$.
- for each selected negative literal $\neg A$ in a leaf node of a finitely failed SLDNF-tree in $\mathbf{S}$ exists exactly one SLDNF-refutation for $P \cup \{\leftarrow A\}$ in $\mathbf{S}$.

A SLDNF-proof is called a SLDNF-proof via a computation rule R iff all SLDNF-trees and refutations in $\mathbf{S}$ are computed via R.

2 Structuring SLDNF-Proofs

2.1 Prooftrees

From a logic program and a query an and-or-tree can be built up in the usual way: starting from the goal clause (an and-node) the successors of an and-node (the or-nodes) are the atoms, that are used in the literals of the clause. If a literal is negative the edge from the clause to the atom is marked with a negation sign. The following example shows the construction.

Example 1

Rules	**Facts**
$ic \leftarrow \neg p$	$employee(r) \leftarrow$
$p \leftarrow employee(E) \wedge has\_computer(E) \wedge access(E, menu)$	$employee(p) \leftarrow$
$access(E, F) \leftarrow owner(E, F)$	$has\_computer(r) \leftarrow$
$access(M, F) \leftarrow manager(M, E) \wedge owner(E, F)$	$has\_computer(p) \leftarrow$
$access(E, F) \leftarrow unrestricted(F)$	$owner(r, menu) \leftarrow$
$manager(M, E) \leftarrow reports(E, M)$	$reports(r, p) \leftarrow$
$manager(M, E) \leftarrow reports(E, I) \wedge manager(M, I)$	

The program clauses can be ordered in the following way:

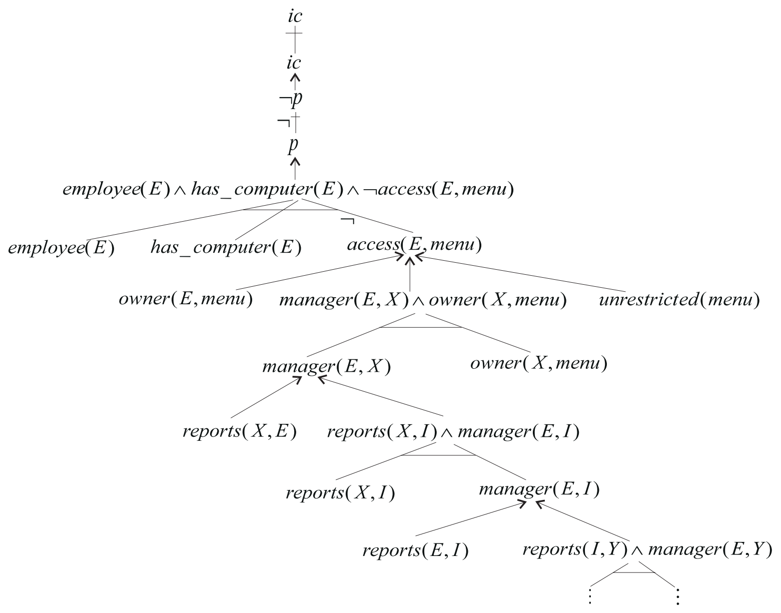

Figure 1: And-Or-Tree

In general due an and-or tree can be of infinite size (due to recursive clauses). However, a proof system (like PROLOG) with certain computation rules moves only in this tree. If a proof is found this proof has a finite size. To exploit the structure of an and-or-tree the following questions have to be answered:

1. What is missing in an and-or-tree to make it equivalent to a SLDNF-proof?

2. How can unrelevant parts of the prooftree be identified?

One thing obviously missing are the relevant substitutions of the variables in the nodes. To answer the other question the following observation is useful: Let $W \equiv \leftarrow L_1 \wedge \ldots \wedge L_n$ be a normal goal clause, for which we want to obtain a finitely failed SLDNF-tree via the standard computation rule (e.g. the literals are selected from left to right). The SLDNF-resolution tries to find a finitely failed SLDNF-tree for $\leftarrow L_1$. Let Φ_1 be the set of all answer substitutions for L_1. Each $\sigma \in \Phi_1$ can be viewed as an *output substitution* for L_1 and *input substitution* for L_2 (L_2 is the next selected literal). The sequence of substitution sets can be formally defined as follows:

1. $\Phi_0 = \{\epsilon\}$
2. $\Phi_i = \{\sigma'\sigma \mid \sigma'$ is the computed answer substitution for $\leftarrow L_i\sigma$, $\sigma \in \Phi_{i-1}$ $\}$ with $1 \leq i \leq n$.

If there is no computed answer substitution for $L_i\sigma$, $\sigma \in \Phi_{i-1}$ we say L_i has filtered out the substitution σ. If $\Phi_n = \emptyset$ (all substitutions have been filtered out by a literal), then there exists a finitely failed SLDNF-tree for $\leftarrow W$. A literal L_i can thus be identified as unimportant if $\Phi_{i-1} \subseteq \Phi_i$, because then L_i has filtered out no substitution and is not relevant for building up a finitely failed SLDNF-tree. With this observation we can define the notion of a *prooftree*. An example of a prooftree is given in figure 2.

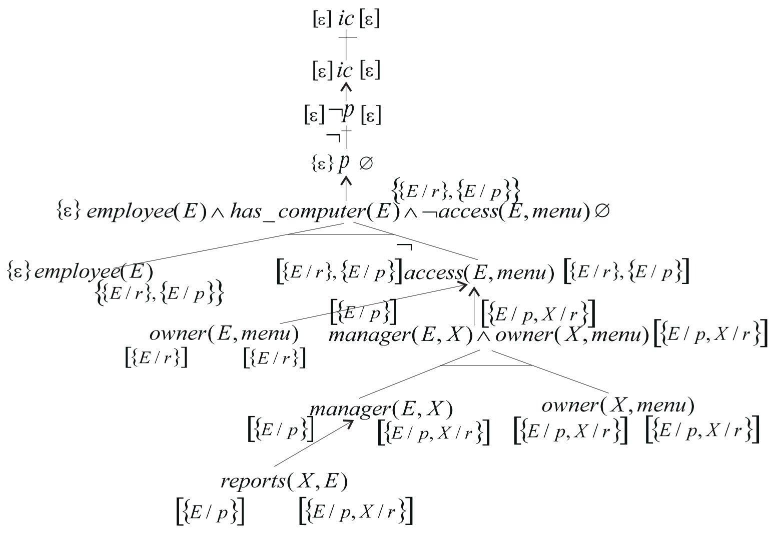
[ε] ic [ε]
[ε] ic [ε]
[ε] ¬p [ε]
{ε} p ∅
{{E/r},{E/p}}
{ε} employee(E) ∧ has_computer(E) ∧ ¬access(E, menu) ∅
{ε} employee(E)
{{E/r},{E/p}}
[{E/r},{E/p}] access(E, menu) [{E/r},{E/p}]
[{E/p}]
[{E/p, X/r}]
owner(E, menu)
manager(E, X) ∧ owner(X, menu) [{E/p, X/r}]
[{E/r}] [{E/r}]
manager(E, X)
owner(X, menu)
[{E/p}] [{E/p, X/r}] [{E/p, X/r}] [{E/p, X/r}]
reports(X, E)
[{E/p}] [{E/p, X/r}]

Figure 2: Prooftree for the Example Database

Analogously to logic programs the notion of a prooftree can be refined. A prooftree can be:

- save (for every negative literal L_i and every substitution $\sigma \in \Phi_{i-1}$ $L_i\sigma$ is a ground Literal).
- allowed (for every literal L_i and every substitution $\sigma \in \Phi_i$ $L_i\sigma$ is a ground literal).
- complete (every predicate symbol of a leaf node is element of the extensional database).

2.2 Soundness

With a formal definition of a prooftree we can proof the following result:

Theorem 1 Let P be a normal programm, $\leftarrow W$ a normal goal clause, T a save and allowed prooftree for $P \cup \{\leftarrow W\}$ and θ the last substitution in the root node of T. Then there exists a SLDNF-refutation of $P \cup \{\leftarrow W\}$ with computed answer substitution θ.

The proof requires an extension of the ususal notion of an SLDNF-tree and refutation: in the extended version leaf nodes with no selected literal in a goal clause are allowed. The key idea is then to build up a finitely failed SLDNF-tree resp. refutation out of partial ones.

2.3 Completeness

The completeness of a SLDNF-tree concerns the following question: which SLDNF-proof can be structured as a prooftree? It is known, that not all SLDNF-proofs can be structured, but we don't have an exact characterization of the structurable SLDNF-proofs.

Fortunatley, it is provable that SLDNF-proofs via the standard computation rule (as used in PROLOG) are structurable. So we obtain the following result:

Theorem 2 Let P be an allowed logic program, $\leftarrow G$ an allowed goal clause and $\mathbf{S}$ a SLDNF-proof for $P \cup \{\leftarrow G\}$ via the standard computation rule with computed answer θ. Then there exists an allowed, save and complete prooftree for $P \cup \{\leftarrow G\}$ with last substitution θ.

The prooftree is effective constructable out of the SLDNF-proof. The key idea is to decompose the SLDNF-trees and refutations into smaller pieces.

3 Incremental Integrity Checking

The general method of integrity checking in deductive databases is to translate the constraint (given as a closed first order formula) in a normal logic program (see [6]) and to evaluate it. The translation of the sorted formula

$$\forall E : employee\ has\_computer(E) \rightarrow access(E, menu)$$

is given in example 1.

To reduce the overhead several incremental checking methods have been developed (see [5, 1, 3, 7]). However, they mostly exploit the *existence* of a proof, not its structure.

We have two different possibilities to exploit a prooftree in this setting:

1. Often the storing of all substitutions might be too costly (although there is not more memory needed then for an optimized bottom-up evaluation of the query). In this case it is possible to store only the relevant rules of the deductive database without further considering the substitutions. Because we know all relevant rules and literals of a proof we can easily determine, when an update influences the validity of the proof.

2. If it is possible to store the substitutions algorithms exists, that realize an effcient JTMS (see [2]) for normal logic programs.

Example 2 The deletion and addition of facts with the predicate symbol '*unrestricted*' does not influence the validity of the prooftree in figure 2. Exact conditions are defined, that specify, when a update is relevant for a proof.

If we delete the fact '$reports(r, p)$' the proof is influenced (the conclusion '$access(p, menu$' can not be derived any more). But if we also delete the fact '$employee(r)$' the proof can be automatically repaired.

4 Conclusion

We have developed a structuring method for SLDNF-proofs. This structuring method can be used as a framework for integrity checking in deductive databases. Most known approaches in integrity checking can be related to this general method. However, a number of questions are unanswered. We list a few:

- How can the computation rules for which a construction of a prooftree is possible be characterized?
- How is a prooftree related to optimized evaluation strategies for deductive databases (magic sets etc.)? This question is important, because query evaluation in deductive databases is usually done via a bottom-up approach. An integration seems possible, because the substitutions in a prooftree and the materialized relations in a optimized evaluation method are very similar. This would give another relation of bottom-up and top-down evaluation strategies.

Bibliography

[1] Hendrik Decker. Integrity Enforcement on Deductive Databases. In Larry Kerschberg, editor, *Proceedings from the 1st International Conference on Expert Database Systems*, pages 381–395, Charleston, South Carolina, April 1986. The Benjamin/Cummings Publishing Company, Inc.

[2] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[3] Ulrike Griefahn and Stefan Lüttringhaus. Top-down integrity constraint checking for deductive databases. In David H.D. Warren and Peter Szeredi, editors, *Proceedings of the 7th International Conference on Logic Programming, Jerusalem*, pages 130–143, Jerusalem, 1990. The MIT Press.

[4] John W. Lloyd. *Foundations of Logic Programming (2nd Edition)*. Springer-Verlag, 1987.

[5] John W. Lloyd, E.A. Sonenberg, and Rodney W. Topor. Integrity constraint checking in stratified databases. *Journal of Logic Programming*, 4:331–343, 1987.

[6] John W. Lloyd and Rodney W. Topor. Making prolog more expressive. *Journal of Logic Programming*, 3(1):225–240, 1984.

[7] Fariba Sadri and Robert A. Kowalski. A theorem-proving approach to database integrity. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 9, pages 313–362. Morgan Kaufmann Publishers, Inc., Los Altos, California, 1988.