Sustained models and sustained answers in first-order databases (short version)

Hendrik Decker* and Juan Carlos Casamayor†
*Siemens AG, ZFE ST SN 33, D-81370 München, hendrik@ztivax.zfe.siemens.de
†DSIC, Univ. Politéc. Ap'do 22 012, E-46071 Valencia, jcarlos@dsic.upv.es
†supported by CICYT grant TIC93-0475

Abstract

First-order databases may contain clauses with disjunctive heads and denials. We distinguish sustained models as alternatives of intended interpretations. Extending Reiter's CWA, we define the Sustained Worlds Assumption SWA and show that it satisfies desirable properties. A positive answer is sustained if it is true in some sustained model. Negative sustained answers are cautiously inferred to be false. We describe a procedure for computing sustained answers. It comprises consistency checking and is sound and complete for finitary queries.

1 Introduction

A first-order (f.o.) database (shortly, database) is a set of clauses, the head of which may contain disjunction or nothing at all. (For further notation and terminology, cf. [17] [18] [19]). Examples of disjunctive clauses (the upper three) and denials (two on the bottom), characterizing rules of marriage law, university matriculation and inheritance of blood groups, are

surname(x,y) ∨ surname(x,z) ← born(x,y) & married(x,w) & surname(w,z)

enrolled(x,maths) ∨ enrolled(x,engineering) ← student(x,computation)

blood-group(x,a) ∨ blood-group(x,b) ∨ blood-group(x,ab) ←
blood-group-pattern(father(x),aa) &
blood-group-pattern(mother(x),bb)

← blood-group(x,a) & blood-group(x,b)

← blood-group(x,0) &
blood-group-pattern(father(x),aa) &
blood-group-pattern(mother(x),bb).

We distinguish sustained models as alternatives of intended interpretations of f.o. databases. Sustained models are steady but not necessarily minimal. Theoremhood of non-negative clauses is equivalent to truth in all sustained models. If there are no sustained models, then the database is inconsistent.

Extending Reiter's CWA, we define the Sustained Worlds Assumption SWA. Theoremhood of a negative clause is inferred if that clause is true in all sustained models. SWA is cautious, cumulative, rational and modular. More such properties are studied in the long version of this paper.

Informally, a positive answer to a query is sustained if the universal closure of the body of the query, instantiated with the answer, is true in some sustained model. For queries without such an answer, we non-monotonically but cautiously infer negative sustained answers. To compute sustained answers, we use PL resolution, a procedure devised in [24]. Applied to databases with denials, PL may infer inconsistent results. Thus, we soundly extend PL by a second phase, for consistency checking of PL outcome. This extension is complete for finitary queries. (The notion of "finitary" is defined in section 6.2.)

Preliminary definitions

A *f.o. database* is a finite set of clauses of the form $H \leftarrow B$, where the *head* H is a disjunction $A_1 \vee \ldots \vee A_n$ $(n \geq 0)$ of atoms A_i $(1 \leq i \leq n)$, the *body* B is a conjunction $B_1 \,\&\, \ldots \,\&\, B_m$ $(m \geq 0)$ of atoms B_j $(1 \leq i \leq m)$ and $m+n > 0$. A clause with $n > 1$ is a *disjunctive clause*. A clause with $m = 0$ is a *positive clause* (also called *possible fact*, or simply *fact* if also $n = 1$), for which "←" is omitted. A clause with $n = 0$ is a *negative clause*, also called *denial*. Denials are used for expressing negative information and queries. A clause with $m + n = 1$ is a *unit clause*. The *empty clause* is denoted by [ ].

For a database P, the *Herbrand base* B_P is the set of ground atoms A such that there is an atom in some clause of P of which A is an instance. For a set S of clauses and a (possibly singleton) set of sentences S', $S \models S'$ means that each element of S' is true in each model of S. As usual, S is called *inconsistent* if $S \models$ false. A (*Herbrand*) *model* M of S is a subset of B_S such that $M \models S$.

2 Sustained models

Definition Let P be a database. A model M of P is *sustained* if there is a mapping |.| assigning to each atom in B_P a natural number such that, for each atom A in M, A occurs in the head of some ground instance $H \leftarrow B$ of some clause in P, $M \models B$ and, for each atom A' in B, $|A'| < |A|$.

For example, for $P = \{p(a) \leftarrow q(b)\}$, $\{p(a), q(b)\}$ is a model, but it is not sustained; also, none of the two models p(a) and q(a) is sustained. The only sustained model of this database is the empty set.

Intuitively, a model M is sustained if, for each atom A in M, A can be traced back to possible facts. Sustained models can be seen as alternatives of intended interpretations. For a definite database, each sustained model is supported [1], but not vice versa. For instance, r is a supported model of the database $\{r \leftarrow r\}$, but not a sustained one (and shouldn't be, since r can't be reduced to any possible fact). Rather, sustained models coincide with "well-supported" models [12], for definite databases. The definition of sustained models has been inspired by mappings (similar to |.|, used for other purposes) in [3]. In [25] [6], definitions of "possible models" and "possible worlds" are discussed. For f.o. databases, they turned out to coincide with sustained models.

Another example: {studies(Fred, maths) ∨ studies(Fred, computing)} admits the sustained model {studies(Fred, maths), studies(Fred, computing)}, which is not minimal. This is reasonable, since the exclusive interpretation of ∨ would exclude the possibility that both studies(Fred, maths) and studies(Fred, computing) hold (cf. [24] for a similar example). Exclusive disjunction can be expressed by additional denials. For example, if $p \vee q$ is supposed to mean "p *ex-or* q", then this is enforced by adding $\leftarrow p \,\&\, q$.

Clearly, sustain depends on the syntax. For example, the singleton database p and the database consisting of p and $p \vee q$ are logically equivalent, and $p \,\&\, q$ is a (non-minimal) model of both, but $p \,\&\, q$ is sustained only for the latter. This example also shows that sustained models are not necessarily minimal.

Sustained models have several desirable properties, as addressed in theorem 1. Steadyness (in 1*a*) is one of them. Steady models, defined below, are similar to stable models [15]; they coincide for definite databases. (A fixpoint characterization of sustained models is studied in [9,10].)

Definition For a database P and a Herbrand model M of P, let P* be obtained as follows. For each clause C in P, each ground instance $H \leftarrow B$ of C such that $M \models B$ and each atom A in H such that $M \models A$, $A \leftarrow B$ is a clause in P*. M is called *steady* if M is the least Herbrand model of P*.

Theorem 1 (Properties of sustained models) Let P be a database.

a) Each sustained model of P is steady, and each steady model of P is sustained.

b) Each minimal model of P is a sustained model of P.

c) For each atom A in B_P, $P \models A$ iff A is true in each minimal model of P.

d) For each atom A in B_P, $P \models A$ iff A is true in each sustained model of P.

e) For P definite, the only sustained model of P is its least Herbrand model.

f) P is inconsistent if and only if there is no sustained model of P.

g) If P is free of denials, then there is a unique *maximal* model Max of P such that Max is a sustained model of P, $M \subseteq Max$ for each sustained model M of P, and Max is the least fixpoint of [24].

Part *a* can be shown along the lines of a similar result in [12] (where "sustained" is replaced by "well-supported" and "steady" by "stable"); *b* - *g* are shown in [10]. The converse of 1*g* does not hold (e.g., the maximal model of $\{p \vee q, \leftarrow p\}$ is q). In general, there is no unique maximal model if there are denials; e.g., the only sustained models of $\{p \vee q, \leftarrow p \,\&\, q\}$ are p and q. Not each model which is a subset of Max is necessarily sustained. For instance, {p, r} is a model of $P = \{p \vee q, r \leftarrow q\}$ and a subset of the maximal model Max = {p, q, r} of P, but, unlike Max and the minimal models {p} and {q, r} of P, {p, r} is not sustained.

3 Sustained answers

For a database P and a query $\leftarrow B$, a substitution θ is a (*positive*) *answer* if $P \models \forall(B\theta)$. θ is a *positive hypothetical answer* if there is a set of sentences (*hypotheses*) Δ such that $P \cup \Delta$ is consistent (if P is) and $P \cup \Delta \models \forall(B\theta)$. Δ is called an *explanation* of θ. If there is no substitution θ such that $P \models \forall(B\theta)$, then the question is if there is a "default explanation" Δ such that $P \cup \Delta \models \neg\exists B$. We then also speak of a *negative hypothetical answer*. A special kind of hypothetical answers is the class of sustained answers:

Definition Let P be a database, $\leftarrow B$ a query and Δ a set of unit clauses.

a) (θ, Δ) is a *positive sustained answer of* $\leftarrow B$ *in* P if θ is a substitution such that $P \cup \Delta \models \forall(B\theta)$ and there is a sustained model M of P such that $M \models \Delta$.

b) We speak of a *negative sustained answer of* $\leftarrow B$ *in* P if $P \cup \Delta \models \neg \exists B$ and, for each sustained model M of P, $M \models \Delta$. Then, Δ is called a *default explanation* of the answer.

c) A *sustained answer of* $\leftarrow B$ *in* P is either a positive or a negative sustained answer of $\leftarrow B$ in P. The Δ associated to a sustained answer in parts *a* and *b* of this definition is also called a *sustained explanation* of the answer.

Clearly, for a negative sustained answer of a query $\leftarrow B$ in a database P with explanation Δ, $P \not\models \forall(B\theta)$ holds for any substitution θ. But the converse does not hold. For instance, in $P = \{p \vee q\}$, $P \not\models p$ holds, but there is no negative sustained answer of $\leftarrow p$ since there is a sustained model of P in which $\neg p$ is not true. The same holds if P is replaced by $P' = \{p \vee q, q\}$. However, there are positive sustained answers of $\leftarrow p$, with explanations $\Delta_1 = p$ (which is "trivial" [7]) and $\Delta_2 = \neg q$ in P and again Δ_1 in P', while Δ_2 is not an explanation for p in P' since $P' \cup \Delta_2$ is inconsistent. In this paper, we are not interested in non-sustained hypothetical answers (such as, e.g., the identity substitution *yes* for query $\leftarrow p$ in $\{p \leftarrow q\}$, with non-sustained explanation q). Moreover, we are not interested in disjunctive answers. (For a database P and a query $\leftarrow B$, a set of ground substitutions $\theta_1, \ldots, \theta_n$ is a *disjunctive answer* to $\leftarrow B$ in P if $P \models \forall(B\theta_1 \vee \ldots \vee B\theta_n)$; for more details, see [19] [5].)

Besides sustained explanations of positive answers, we are particularly interested in default explanations of negative answers that are maximal: A default explanation Δ of some negative sustained answer in some theory P is *maximal* if Δ consists precisely of all ground negative unit clauses that are true in each sustained model of P. The default explanations that we are going to deal with below are always maximal.

Consistency of explanations of sustained answers is ensured by the following result. The proof is easy and therefore left away.

Theorem 2 (Consistency of sustained explanations)
Let P be a consistent database and C a query. If (θ, Δ) is a sustained answer of C in P, then $P \cup \Delta$ is consistent.

4 The Sustained Worlds Assumption

If P is definite, the usual default explanation Δ for inferring negative answers is the well-known *closed world assumption* CWA(P), i.e. all negative ground literals $\neg A$ such that A is not a logical consequence of P [23]. For P non-definite, it is well-known that $P \cup$ CWA(P) may be inconsistent.

For a database P, there are essentially two different generalizations of CWA in the literature, viz. the *generalized closed world assumption* GCWA(P) and the *weak generalized closed world assumption* WGCWA(P) [19]. The latter is also known as DDR [24]. Both satisfy several dersirable properties, such as consistency with P, upward compatibility in the sense that both coincide with CWA(P) for P definite, etc. For function- and denial-free P, GCWA(P) contains the negation of each ground fact which is not true in any minimal model of P. Admitting functions but not denials in P, it is easy to show that WGCWA(P) consists precisely of the negation of each ground fact which is not true in the maximal sustained model of P.

GCWA generalizes straightforwardly to databases with functions and denials. However, GCWA does not comply with the requirement that disjunction in the head of clauses should preserve its original meaning in first-order logic. For example, for $P = \{p \vee q,\ q\}$, GCWA(P) infers that $\neg p$ holds, although that is not a cogent consequence of P, from a logical point of view. In many examples, GCWA(P) effectively amounts to interpreting disjunction as exclusive "or", which may not always be the intended meaning. On the other hand, if disjunction is always interpreted as exclusive, then, e.g., the database $P = \{p \vee q,\ q \vee r,\ p \vee r\}$ is inconsistent (since there is no model of P that interprets each of the three disjunctions as exclusive), contrary to GCWA(P), since $P \cup$ GCWA(P) is (rightfully) consistent. Other problems with GCWA are identified in [11].

As opposed to the possibly befuddling semantics of GCWA, WGCWA does comply with a reading of disjunction that is not necessarily exclusive [24]. However, generalizing WGCWA to arbitrary f.o. databases is less obvious, since there may not be a unique maximal sustained model in the presence of denials. Thus, we are going to propose a generalization of CWA, essentially by replacing "minimal" in the definition of GWCA by "sustained".

That generalization, called SWA, intuitively expresses the assumption that the possible worlds intended by P are precisely the models of P that are sustained, and that all non-sustained sentences are deemed false.

Definition For a database P, let $\underline{\text{SWA}}(P)$ be the set of negated atoms $\neg A$ such that $A \in B_P$ and A is not true in any sustained model of P. Then, the *sustained worlds assumption* SWA(P) is defined as the union of P and $\underline{\text{SWA}}(P)$: $\text{SWA}(P) = P \cup \underline{\text{SWA}}(P)$.

5 Properties of SWA

Next, we show that SWA(P) satisfies several properties which have been identified as desirable for non-monotonic inference in [13] [24] [21] [14] [19] [11]. For saving space, we do not give proofs, which can be obtained along the lines of proofs of similar results in [24] [25] [6] [11].

Consistency, as expressed below, is certainly a least requirement, in the sense that any reasonable inference rule should yield consistent conclusions at any rate.

Theorem 3 (Consistency) For a consistent database P, SWA(P) is consistent.

Informally speaking, the next result says that SWA is an upward-compatible generalization of CWA and WGCWA.

Theorem 4 (Compatibility)

a) For a definite database P, $\underline{\text{SWA}}(P) = \text{CWA}(P)$.

b) For a denial-free database P, $\underline{\text{SWA}}(P) = \text{WGCWA}(P)$.

Intuitively, theorem 5, below, says that $\text{SWA}(P) = P \cup \underline{\text{SWA}}(P)$ is balanced, in the sense that P alone suffices for inferring all inferrable positive information and $\underline{\text{SWA}}(P)$ alone caters for all sustained negative information. 5*a* corresponds to the "stability" of WGCWA(P) in [19].

Theorem 5 (Balance)

Let P be a database. Then, the following holds.

a) For a positive clause C, $P \models C$ if and only if $\text{SWA}(P) \models C$.

b) For a negative clause C, $\underline{\text{SWA}}(P) \models C$ if and only if $\text{SWA}(P) \models C$.

Theorem 6 (Downward Monotony)

For a database P, let P^+ denote the set of all clauses with non-empty head in P, and P^- be the set of denials in P. Further, let P_1, P_2 be two consistent databases such that $P_1^+ \subseteq P_2^+$ and $P_2^- \subseteq P_1^-$. Then, $\underline{\text{SWA}}(P_2) \subseteq \underline{\text{SWA}}(P_1)$.

Intuitively, downward monotony of SWA says that the amount of implicit negative information inferred from a database decreases as its explicit positive knowledge increases. Actually, that corresponds to the "decreasing" property of WGCWA in [24] (called "monotonicity" in [19]), as expressed in 6*b*. As shown in [24], GCWA is not downward monotone, as opposed to SWA.

Since CWA is not monotone, it follows from theorem 4*a* that SWA is not monotone either. Likened to the scheme in theorem 7*c* below, monotony would have that, if, for clauses C and C', $\text{SWA}(P) \models \neg C$ and $\text{SWA}(P) \models C'$, then $\text{SWA}(P \cup \{C\}) \models C'$, but that does not hold (e.g., for $P = \{p \leftarrow q\}$, $C = q$ and $C' = \leftarrow p$, adding C to P decreases the set of logical consequences of $\underline{\text{SWA}}(P)$ and hence of SWA(P)). However, several restricted forms of monotonicity, as discussed in the literature, hold for SWA. One of them is the downward monotony of SWA, as expressed in theorem 6 above. Other forms are exhibited in theorems 7 and 8, below.

Theorem 7

Let P be a database and C, C' two clauses. Then, the following holds.

a) If $\text{SWA}(P) \models C$ and $\text{SWA}(P) \models C'$, then $\text{SWA}(P \cup \{C\}) \models C'$. (Cautious monotony)

b) If $\text{SWA}(P) \models C$, then $\text{SWA}(P) \models C'$ iff $\text{SWA}(P \cup \{C\}) \models C'$. (Cumulativity)

c) If $\text{SWA}(P) \not\models \neg C$ and $\text{SWA}(P) \models C'$, then $\text{SWA}(P \cup \{C\}) \models C'$. (Rationality)

The last property that we are going to deal with is *modularity*. It enables a modular computation of SWA(P), by deriving consequences of given predicate definitions separately, focusing on modules consisting of the clauses on which those definitions depend. (Dependency is defined as the transitive closure of direct dependency: A clause C *depends directly* on a clause C' if there is an atom in the body of C that unifies with an atom in the head of C'.)

Theorem 8 exhibits modularity in two forms. 8 *a* caters for unrelated modules. 8 *b* is interesting, e.g., for the incremental computation of SWA(P) in a database P that is layered into modules ("strata"), such that dependencies between modules are not circular [26].

Theorem 8 (Modularity) Let P be a database. Then, *a* and *b*, below, hold.

a) Let $P_1 \cup P_2$ be a partition of P such that no clause in P_1 depends on any clause in P_2 and vice-versa. Then, $SWA(P) = SWA(P_1) \cup SWA(P_2)$.

b) Let P' be a set of clauses in P such that no clause in P' depends on any clause in $P - P'$. Then, $SWA(P') \subseteq SWA(P)$.

6 Computing sustained answers

For a database P, procedures that compute logical consequences of WGCWA(P) are described in [19] [25]. Since only each negative, but not necessarily each positive sustained answer is a logical consequence of SWA(P), the purpose of this section is different from that of the procedures just mentioned.

In [24], a procedure called PL (Positive Linear) resolution is introduced. PL is sound for computing answers sustained by WGCWA(P), i.e. SWA(P) for P denial-free. Below, we first briefly describe PL (or, rather, our version of PL, which originally has been defined for inferring negative answers only), and then specify a sound two-folded extension of PL, called PLY, which is complete for significant classes of queries and databases.

6.1 PL resolution

We assume familiarity with SLD resolution [16] (cf., e.g., [18]).

Definition Let P be a database.

a) Let $P_{max} = \{A \leftarrow B : H \leftarrow B \in P, A \text{ is an atom in } H\}$.

b) For an SLD refutation *d* with computed answer θ and input clauses from P_{max}, the *(d-)computed explanation of* θ is the set of unit clauses of form Aθ such that there is an input clause $A \leftarrow B$ in *d*, there is a variant $H \leftarrow B$ of a disjunctive clause in P, and A occurs in H.

PL hypothesizes atoms in the head of disjunctive input clauses and ignores the alternative disjuncts. Intuitively, PL-computed substitutions are explained by the hypothesized atoms. Similar to [24], we define:

Definition Let P be a database, C a query and θ a substitution.

a) A *PL refutation of* C *in* P *with outcome* (θ, Δ) is an SLD refutation of $P_{max} \cup \{C\}$ with computed answer θ, the computed explanation of which is Δ.

b) A *PL tree of* C *in* P *of finite depth* is an SLD tree of $P_{max} \cup \{C\}$, each branch of which is of finite length (i.e., either successful or failed). A PL tree of finite depth is *finitely failed* if each of its branches is failed.

For P denial-free, the least Herbrand model of P_{max} clearly is the maximal model of P. Hence, for a query C, the outcome of each PL refutation of C in P is a sustained answer of C in P. Moreover, the substitution of each sustained answer of C in P is subsumed by the outcome of some PL refutation of C in P. However, if P contains denials, then P_{max} may become inconsistent with P. For example, running PL on query $\leftarrow p$ in $P = \{p \leftarrow q, q \vee r, \leftarrow q\}$ succeeds with computed explanation q, although $SWA(P) \not\models p$. Generally, if the computed explanation Δ of some PL outcome (θ, Δ) is not a sustained explanation of θ, then it is inconsistent, as follows from part *a* of the next result, which can be shown by induction.

Theorem 9 (Soundness and completeness of PL)
Let P be a consistent range-restricted database and C a query.

a) If (θ, Δ) is the outcome of a PL refutation of C in P and $P \cup \Delta$ is consistent, then (θ, Δ) is a positive sustained answer of C in P.

b) If there is a PL tree of finite depth of C in P and for each refutation *d* in the tree (if any) with computed explanation Δ_d, $P \cup \Delta_d$ is inconsistent, then (*no*, SWA(P)) is a negative sustained answer of C in P.

c) If θ is the substitution of a positive sustained answer of C in P, then there is a PL refutation of C in P with outcome (θ', Δ) and a substitution θ" such that $\theta = \theta'\theta''$ and $P \cup \Delta$ is consistent.

d) If (*no*, SWA(P)) is a negative sustained answer of C in P and Δ is the computed explanation of any PL refutation of C in P, then $P \cup \Delta$ is inconsistent.

According to 9 *a*, positive PL outcome is sound only if the computed explanation is consistent. According to 9 *b*, inferring a negative answer from PL outcome is sound only if each computed explanation (if any) is inconsistent. Thus, a procedure that checks PL-computed outcome for (in)consistency is needed.

6.2 (In)consistency checking with SLC resolution

The definition of SLC resolution below is a logic-programming-style formalization of the positive refinement [22] of model elimination [20]. For brevity, we skip the definition of finitely failed SLC trees (i.e., roughly, each attempt of finding an SLC refutation terminates with failure). As shown in theorem 10, SLC is a theorem prover which can be used for (in-) consistency checking of f.o. databases. More on SLC is available in [5].

Definition Let P be a database and C a clause. An *SLC refutation of* C *in* P, and its *rank* k ($k \geq 0$), consists of a sequence $C_0 = C, \ldots, C_n = [\,]$ $(n \geq 0)$ of clauses such that, for each $i < n$, an atom A_i is selected in C_i, and the two points below hold. We call A_i an *ancestor* if A_i is selected in the head of C_i and A_i is ground and $A_i \notin P \cup \{C\}$. If, for each $i < n$, A_i is not an ancestor, then the rank of the refutation is 0. Otherwise, it is k, for some sufficiently large $k > 0$. (A more elaborate definition would induct on k).

① If A_i is not an ancestor, then C_{i+1} is a resolvent of C_i and a fresh variant C′ of some clause in $P \cup \{C\}$ on A_i and an atom A′ in C′, using an mgu of A_i and A′.

② If A_i is an ancestor, then there is an SLC refutation of A_i in $P \cup \{C\}$ of rank less than k, and C_{i+1} is the resolvent of C_i and $\leftarrow A_i$ on A_i.

For P definite, SLC clearly coincides with SLD. In general, the root of an SLC refutation may be any clause. Condition $A_i \notin P \cup \{C\}$, for ancestors A_i, avoids circularity through point ② of the definition. Soundness and completeness of SLC in theorem 10 *a*-*d*, below, follow from results proved in [2]. For ensuring the termination of SLC, we impose the restriction of "finitary" (cf. the similar concept of "acyclicity" and "boundedness", e.g. in [3]), which relies on the following notions:

Definition Let C_1, C_2 be clauses and P a (possibly infinite) set of clauses.

a) C_1 is *directly connected to* C_2 *via* atom A_1 in the body (or head) of C_1 *and* atom A_2 in the head (or, resp., body) of C_2 if A_1 unifies with A_2.

b) C_1 is *connected to* C_2 *in* P *via* atom A_1 in C_1 *and* atom A_2 in C_2 if either C_1 is directly connected to C_2 via A_1 and A_2, or there is a clause C in P with distinct occurrences of atoms A, A′ such that C_1 is directly connected to C via A_1 and A, and C is connected to C_2 in P via A′ and A_2.

c) A *connection path in* P is a sequence of connected clauses in P such that the following holds: For each triple C_1, C_2, C_3 of consecutive clauses in the sequence, there are distinct occurrences A_2, A'_2 of atoms in C_2 such that C_1 is directly connected to C_2 via some atom A_1 in C_1 and A_2, and C_2 is directly connected to C_3 via A'_2 and some atom A_3 in C_3. The number of clauses in a connection path is its *length*.

Definition Let P be a database and $\underline{P}$ the set of ground instances of clauses in P. We say that P is *finitary* if the length of each connection path in $\underline{P}$ is finite. A query C is *finitary in* P if, for the set P_C of clauses in P to which C is connected, $P_C \cup \{C\}$ is finitary.

It can be shown that, if query C is finitary in database P, then there is no infinite SLC derivation of C in P. For example, ← related(child(eve), y) is finitary in {related(child(x), y) ← related(x, parent(y))}. In the long version of the paper, the condition of "finitary" is further relaxed.

Imposing "range-restricted" in 10*c*, below, and additionally "finitary" in 10*d*, is sufficient for obtaining completeness of SLC. For 10*b*, *c*, the following definition is convenient.

Definition For a database P, let $\tilde{P}$ be obtained by replacing each denial of form ← B in P by inconsistent ← B, where inconsistent is a predicate that is not used elsewhere.

Theorem 10 (Soundness and completeness of in/consistency checking with SLC) For a database P and a clause C, SLC resolution is sound, as expressed in *a* and *b*, and complete, as expressed in *c* and *d*, below.

a) If there is an SLC refutation of C in P, then $P \cup \{C\}$ is inconsistent.

b) If there is a finitely failed SLC tree of ← inconsistent in $\tilde{P}$, then P is consistent.

c) If P is inconsistent and range-restricted, then there is an SLC refutation of ← inconsistent in $\tilde{P}$.

d) If $P \cup \{C\}$ is consistent and range-restricted and C is finitary in P, then there is a finitely failed SLC tree of C in P, and each SLC tree of C in P is finitely failed.

6.3 PLY resolution

Definition Let P be a range-restricted database, C a query and $\tilde{P}$ as in 6.2.

a) A *PLY refutation of* C *in* P with *computed answer* (θ, Δ) consists of two phases. The first is a PL refutation of C in P with outcome (θ, Δ) and the second a finitely failed SLC tree of $\leftarrow$ inconsistent in $\tilde{P} \cup \Delta$.

b) A *failed PLY run of* C *in* P consists of two phases. The first is a PL tree T of C in P of finite depth. The second is a set of SLC refutations of $\leftarrow$ inconsistent in $\tilde{P} \cup \Delta_d$, one for each PL refutation d in T, where Δ_d is the computed explanation of d.

Frequently, the second phase of PLY is not necessary, e.g. if P is denial-free. From theorems 9 and 10, the results below follow immediately.

Theorem 11 (Soundness and completeness of PLY)
For a consistent range-restricted database P and a query C, PLY resolution is sound, as expressed in *a* and *b*, below. If, additionally, P is finitary, then PLY is also complete, as expressed in *c* and *d*.

a) If there is a PLY-computed answer (θ, Δ) of C in P, then (θ, Δ) is a positive sustained answer of C in P.

b) If there is a failed PLY run of C in P, then (*no*, SWA(P)) is a negative sustained answer of C in P.

c) If θ is the substitution of a positive sustained answer of C in P, then there is a PLY refutation of C in P with computed answer (θ', Δ) and a substitution θ'' such that $\theta = \theta'\theta''$.

d) If there is a negative sustained answer of C in P, then there is a failed PLY run of C in P, and each PLY run of C in P is failed.

Conclusion

For first-order databases, we have developed a default semantics, SWA, that satisfies several desirable properties. Based on SWA, we identified various kinds of answers and developed a sound and complete proof procedure, PLY, for computing sustained answers, which form an interesting class of hypothetical answers. While PLY-computed explanations of negative answers always hold by default, computed explanations of positive answers, though sustained, typically do not hold by default, and are sometimes redundant. On the other hand, we have developed, in [4], Yet another selection-driven linear resolution procedure, called SLY, for computing several kinds of hypothetical answers, among them all sustained ones, and also all that are true under the weak default of GCWA. Current work is concerned with tayloring PLY and SLY such that only explanations which serve some useful applications are computed.

Acknowledgement A preliminary version of this paper appeared in the proceedings of the 4th International Workshop on the Deductive Approach to Information Systems and Databases, Lloret de Mar, Catalunya (Spain), 20 – 22 September 1993.

References

[1] K.R. Apt. H.A. Blair and A. Walker, Towards a theory of declarative knowledge, in J. Minker (ed), *Foundations of Deductive Databases and Logic Programming*, 89-148, Morgan Kaufmann, 1988.

[2] J.C. Casamayor, New paradigms of resolution-based reasoning in computational theories, *draft*, DSIC, Univ. Politéc. Valencia, 1994. To be submitted as PhD thesis.

[3] L. Cavedon, Acyclic logic programs and the completeness of SLDNF-resolution, *Theoretical Computer Science* 86, 81-92, 1991.

[4] J.C. Casamayor and H. Decker, Hypothetical query answering in first-order databases, DSIC, Univ. Politéc. Valencia, 1994; preliminary version in *Proc. Dutch/German Workshop on Nonmonotonic Inference Techniques and their Applications*, RWTH Aachen, 1993.

[5] J.C. Casamayor, H. Decker and F. Marqués, A logic-programming-style inference technique for disjunctice databases, DSIC, Univ. Politéc. Valencia, 1994.

[6] E. Chan, A possible world semantics for disjunctive databases, *IEEE Transactions on Knowledge and Data Engineering* 5, 282-292, 1993.

[7] P.T. Cox and T. Pietrzykowski, Causes for events: Their computation and applications, in J. Siekmann (ed), *Proc. 8th CADE*, 608-621, Springer LNCS, 1986.

[9] H. Decker, Alternative models and fixpoints for first-order databases, Siemens, 1991, revised 1994.

[10] H. Decker, Foundations of first-order databases, Siemens, 1992.

[11] J. Dix, Classifying semantics of disjunctive logic programs, in K. Apt (ed), *Proc. Joint Int'l Conf. and Symp. Logic Programming*, 798-812, MIT Press, 1992.

[12] F. Fages, A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics, *New Generation Computing* 9, 425-443, 1991.

[13] D.M. Gabbay, Theoretical foundations for non-monotonic reasoning in expert systems, in K.R. Apt (ed), *Proc. NATO Advanced Study Institute on Logics and Models of Concurrent Systems*, 439-457, Springer, 1985.

[14] P. Gärdenfors, *Knowledge in Flux*, MIT Press, 1988.

[15] M. Gelfond and V. Lifschitz, The stable model semantics for logic programs, in R.A. Kowalski and K.A. Bowen (eds), *Proc. 5th Int'l Conf. and Symp. Logic Programming*, Vol. 2, 1070-1080, MIT Press, 1988.

[16] R.A. Kowalski, Predicate logic as a programming language, *Proc. IFIP '74*, 569-574. North-Holland, 1974.

[17] R.A. Kowalski, *Logic for Problem Solving*, North Holland, 1979.

[18] J.W. Lloyd, *Foundations of Logic Programming*, Springer, 1987.

[19] J. Lobo, J. Minker and A. Rajasekar, *Foundations of Disjunctive Logic Programming*, MIT Press, 1992.

[20] D.W. Loveland, A simplified format for the model elimination procedure, *J. ACM* 16, 349-363, 1969.

[21] D. Makinson, General theory of cumulative inference, in M. Reinfrank, J. de Kleer and M.L. Ginsberg (eds), *Proc. 2nd Int'l Workshop on Non-monotonic Reasoning*, Springer LNAI 346, 1989.

[22] D.A. Plaisted, A sequent-style model elimination strategy and a positive refinement, *J. Automated Reasoning* 6, 389-402, 1990.

[23] R. Reiter, On closed world databases, in H. Gallaire and J. Minker (eds), *Logic and Data Bases*, 55-76,Plenum Press, 1978.

[24] K.A. Ross and R.W. Topor, Inferring negative information from disjunctive databases, *J. Automated Reasoning* 4, 397-424, 1988.

[25] C. Sakama, Possible model semantics for disjunctive databases, *Proc. 1st Int'l Conf. Deductive and Object-Oriented Databases*, 337-351, 1989.

[26] J. Sebelik and P. Stepanek, Horn clause programs for recursive functions, in K.J. Clark and S.Å. Tarnlund (eds), *Logic Programming*, 325-340, Academic Press, 1982.

Abductive Update of Deductive Databases

G. Plagenza
Dipartimento di Informatica
Università di Pisa
Corso Italia, 40
I-56125 Pisa, ITALY
Email: paolo@di.unipi.it

Abstract

In this paper the problem of view updates in deductive databases is studied. Our aim is to show how this problem can be tackled with an abductive–based approach. We give a procedure that "translates" the update request to a deductive database into a new modification request regarding only the extensional part of the database.

This approach appears as a simple yet powerful method, permitting both to handle the difficulties connected to the presence of negation in deductive databases and to define a uniform common update procedure for insert and delete requests. This procedure is formally defined and its correctness wrt a simple extension of the Dung's preferential semantics for standard abduction is investigated. The abductive approach to the update problem permits also many natural extensions of the base version of this procedure. One of them permits to deal with an integrity theory associated with the database. We will show how it is possible in a really natural way to adapt the procedure to handle this new type of constraints. The extendend version of the procedure is presented, a simple declarative semantics obtained by extending the previous one is proposed and the correctness of this modified version of the procedure wrt this semantics is proved.

1 Introduction and Motivation

The problem we want to tackle can be formulated in this way. Given a deductive database and an update request *insert*(ϕ) (resp. *delete*(ϕ)), where ϕ is a closed first order formula, we want to change the database so that in the new state the update request is satisfied.

This problem is a generalization of the view update problem for relational databases (see e.g. [1,2,3,7,8,9]). But we can also consider it as a particular case of a more general problem, namely that of Knowledge Assimilation (see e.g. [12]), where the

information we want to assume is the truth (resp. falsity) of ϕ. It has been argued that it is not necessary to insert into the database exactly the information that is the subject of the update request, but only a plausible explanation of it; in other words, we insert into the deductive database the cause and not the effect. There are several reasons that justify this approach. First, because of the particular structure of the rules and the facts in the database, if we just insert explicitly the observation and not a plausible cause, some implicit consequences of this observation would be lost.

Example 1

$grandfather(X,Y) \leftarrow father(X,Z), father(Z,Y)$
$sibling(X,Y) \leftarrow father(Z,X), father(Z,Y)$

Suppose that we acquire the new knowledge that Tom and Bob are siblings. If we add explicitly this observation to the database, we cannot deduce the implicit consideration that Tom and Bob have the same father and the same grandfather. In the same way, by asserting that Tom and Bob have the same grandfather, we cannot deduce that they are sons of the same person. So, only by inserting a plausible hypothesis for the fact, in this case the knowledge $father(x, Tom) \wedge father(x, Bob)$, we can deduce all the informations that are implicitly connected to the initial observation.

Another related reason to assimilate information implicitly, is that relevant new information may not always be acquired in terms directly related to the way the database itself is organized. We must "translate" some parts of this information into a form that reflects the organization of the information within the database. In a deductive database DB[1] we can distinguish two parts:

(a) the *extensional* part (EDB), that is a set of ground instances of *base* predicates which describe the current state of the world modelled by the database;

(b) the *intensional* part (IDB), that is a set of rules defining the *view* predicates of the deductive database, which basically represents the real inferential engine of the database.

The database can be shared among more different users, and so it is natural to think of another partition, as in the next figure. In this partition each user has got its own IDB and shares with the others only the extensional part of the database. Whenever a new piece of information on a view relation arrives at a particular user, this is assimilated by appropriately changing the EDB rather than adding it explicitly into the view. So, the modification can be immediately shared among all the users and does not remain confined in the particular IDB of the user who acquired the new information.

There is another important reason that justifies this approach. Generally, the rules of IDB in a deductive database are regarded as a static and non-modifiable part of the database, whereas EDB represents the knowledge that we have about the

[1] In this paper we will regard a deductive database as a logic program.

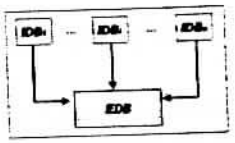

Figure 1: Data partition in shared deductive databases.

world modelled in the deductive database, that can change dinamically. In fact, new knowledge about base relations can dinamically be acquired and retracted. Then, it is natural that every update request affects only EDB, by considering the base predicates as *abducibles* (that is predicates whose instances can be assumed when necessary) and the view update problem as an abductive problem, as suggested in [11]. The basic idea is to associate to the particular IDB of a deductive database an appropriate abductive framework, so that a generic update request $insert(\phi)$ (resp. $delete(\phi)$) is regarded as a new observation to be explained, that is $explain(\phi)$ ($explain(\neg\phi)$), in terms of hypotheses on base predicates, treated as abducibles. Our procedure lets us "translate" the update request to a deductive database in a new update request, affecting only the extensional part of the database. This aim can be represented in this way:

Figure 2: Aim of the abductive update procedure.

Given a generic update request, the procedure computes a plausible explanation for the fact we want to insert into the database, and the real insertion will involve only this explanation (the realization of the procedure that really executes the update lies outside the goal of this paper). To further clarify our aim, consider the next example.

Example 2

IDB : $p(x) \leftarrow B(x)$
EDB : $B(c)$
$B(d)$
Update: $insert(p(a))$

The first step of our procedure will have to compute a plausible explanation for $p(a)$. During this phase the structure of the IDB is investigated and, at the end, the result $\Delta = \{B(a)\}$ is given. So, the update request can be translated into a modification affecting only the EDB, that is, in this particular case, into the insertion of $B(a)$ into the extensional part of the database.

2 Specification of the Base Update Procedure

Let us first briefly summarize the concept of *abduction*. By such a term we mean a form of nonmonotonic reasoning that has recently received much attention in Artificial Intelligence. Let us briefly review the basic notion of abduction. In its simplest form, abduction allows to infer β as a plausible explanation for α, given the knowledge $\alpha \leftarrow \beta$.

Definition 1 By an *abductive framework* we mean a triple $\langle KB, IC, HY \rangle$ where:

(a) KB is a first order theory,

(b) HY is a set of formulas which represents all the admissible hypotheses,

(c) IC is a set of integrity constraints.

The abducible predicates have no definition in KB but are assumables, and given an observation Q the aim of abduction is to *explain* Q by finding a set Δ of hypotheses in HY which satisfies IC and which, together with KB, entails Q (see [10]). In [6], abduction has been used to simulate and extend Negation as Failure in Logic Programming. The basic idea is to view negative literals of the form "*not* $p(t)$" as positive atoms of the form $p^*(t)$, where p^* is a new predicate symbol (see Definition 2). The abducibles are exactly the ground instances (over the Herbrand Universe) of these newly introduced predicate symbols. Due to lack of space, we omit here the formal definition of this transformation (the reader can refer to [5,6,11]). A similar transformation will be given in the sequel (definition 2).

In [5], Dung has proposed an interesting declarative semantics for negation as failure through abduction. Our aim is to show how it is possible to extend, in a natural way, this semantics to handle also base abducibles. The result of this extension will be the semantic counterpart of the procedure for the view update problem in its simplest version (originally proposed in [11]). To describe the method that the procedure uses to treat base abducibles, let us consider the next example.

Example 3 Let $\langle KB, IC, HY \rangle$ be the following abductive program:

$$KB : p(x) \leftarrow q^*(x)$$
$$q(x) \leftarrow B(x)$$

where B is a base predicate, and suppose that the query is $\leftarrow p(a)$. The integrity constraints, as in [6], are denials of the form $\leftarrow p(x), p^*(x)$, for each predicate occurring in EDB and IDB. The computation proceeds as in the next figure, giving the result $\Delta = \{q^*(a), B^*(a)\}$.

Recall that, due to the transformation of Eshghi and Kowalski, $q^*(a)$ (resp. $B^*(a)$) stands for "*not* $q(a)$" (resp."*not* $B(a)$"). Moreover, the procedure uses only the rules in

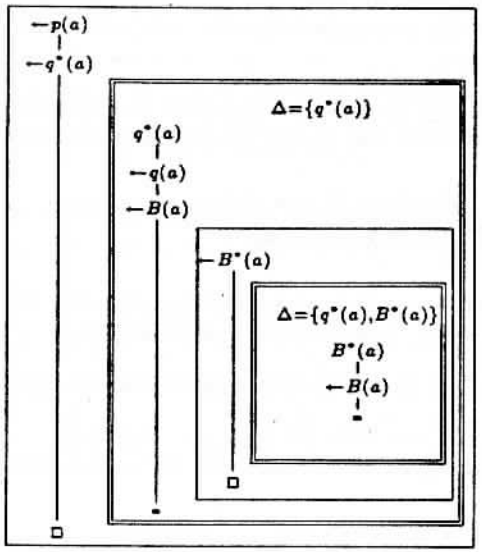

Figure 3: The base update procedure.

IDB and considers the base predicates as abducibles. The procedure can be described as follows. The computation proceeds as in standard SLD–resolution, using the *IDB* rules, until an abducible atom is selected. Since abducibles have no definitions in the *IDB* part, the only way to proceed is then to assume them (as for $q^*(a)$ in the example). But this can be done only if consistency is ensured (in this case if $q(a)$ can be shown not to be provable). This is the aim of the inner double–lined box, where the current set of hypotheses is $\Delta = \{q^*(a)\}$. The consistency of $q^*(a)$ is ensured by assuming that $B(a)$ is not provable, i.e. by abducing $B^*(a)$. Notice that the consistency of $B^*(a)$ is simply ensured by adding $B^*(a)$ itself into the Δ, provided $B(a)$ has not been already added to the Δ.

As shown by the example, the basic procedure is an interleaving of two phases:

(a) the *abductive phase* corresponds to the single–lined boxes, where SLD–resolution is exploited until abducibles are selected;

(b) the *consistency phase* corresponds to the double–lined boxes, where the converse of an abductive hypothesis is shown not to be provable by failing all its possible derivations.

Notice that consistency phases can require nested abductive phases and vice versa.

It is worth remembering that the technique used for the consistency checking has been obtained by adapting and extending the method of Sadri and Kowalski (see e.g.

[14]). Note that, according to such method, we can adfirm the consistency between the currently assumed hypothesis and the database constraints only if all the branches in the search space of the corresponding consistency derivation end with the black box.

Formally, we first have to extend the notion of abductive program treated by Dung, since we want to treat also base predicates as abducibles.

Definition 2 Given a database P, the *abductive program* associated to it is an abductive framework $\langle P^*, IC, Ab\rangle$ such that:

- P^* is the Horn theory obtained from P by replacing all the occurrences of a negative literal $\neg q(t_1,\ldots,t_n)$ with a new atom $q^*(t_1,\ldots,t_n)$;
- $IC = \{\leftarrow q(x), q^*(x) \mid q$ is a predicate symbol occurring in $P\}$;
- $Ab = \{p(t_1,\ldots,t_n) \mid p$ is base $\} \cup \{p^*(t_1,\ldots,t_n) \mid p(t_1,\ldots,t_n) \in HB\}$ is the set of all the abducible predicates, where HB is the Herbrand Base of the language of P.

Notice that this corresponds to the abductive framework described in [5,6], apart from the fact that base predicates are considered as abducibles (see also [11]).

The presence of base abducibles introduces a substantial optimization in the Dung's procedure. In fact, to abduce a base abducible we do not need to open a consistency derivation because the complement of a base abducible is abducible itself, and so it is not provable unless it is explicitly abduced. In other words, to assume a base abducible B we need only to check if B^* is not already assumed. Then, let us look at the abductive procedure to update a deductive database in its simplest version ([11]). Note that in the rest of this paper we will use the notation according to which, if A is the atom $p(t_1,\ldots,t_n)$ (resp. $p^*(t_1,\ldots,t_n)$) then A^* denotes the corresponding atom $p^*(t_1,\ldots,t_n)$ (resp. $p(t_1,\ldots,t_n)$). Let $\langle KB, IC, Ab\rangle$ be an abductive program and R a safe selection rule, that is a rule which selects abducibles only if they are ground.

Definition 3 An *abductive derivation* from (G_1, H_1) to (G_n, H_n) is a sequence $(G_1, H_1), (G_2. H_2), \ldots, (G_n, H_n)$ such that, for each i, with $0 < i \leq n$, G_i has the form $\leftarrow l, l'$ where, without loss of generality, R selects l and l' is a (possibly empty) conjunction of atoms. Moreover, H_i is a set of hypotheses and G_{i+1}, H_{i+1} are computed according to these rules:

BA1) if l is not abducible then $G_{i+1} = C$ and $H_{i+1} = H_i$, where C is the resolvent of some clause in KB with G_i on the selected literal l;

BA2) if l is abducible and $l \in H_i$ then $G_{i+1} = \leftarrow l'$ and $H_{i+1} = H_i$;

BA3) if l is a base abducible, $l \notin H_i$ and $l^* \notin H_i$, then $G_{i+1} = \leftarrow l'$ and $H_{i+1} = H_i \cup \{l\}$;

BA4) if l is a non–base abducible, $l \notin H_i$ and there exists a consistency derivation from $(\{\leftarrow l^*\}, H_i \cup \{l\})$ to $(\{\}, H')$ then $G_{i+1} = \leftarrow l'$ e $H_{i+1} = H'$.

Step [BA3] describes exactly what we have previously mentioned. By selecting a base abducible that has not been already abduced, we can abduce it unless its complement has already been assumed. All the other steps are really analogous to the corresponding ones in the Dung's abductive derivation.

Definition 4 Let ϕ be a conjunction of atoms. Then, an *abductive refutation* for ϕ is an abductive derivation from $(\leftarrow \phi, \{\})$ to $(\Box, H)$, for some H.

Definition 5 A *consistency derivation* from (F_1, H_1) to (F_n, H_n) is a sequence $(F_1, H_1), (F_2, H_2), \ldots, (F_n, H_n)$ such that, for each i, with $0 < i \leq n$, F_i is a set of goals of the form $\{\leftarrow l_1,\ldots,l_s\} \cup F_i'$ where, without loss of generality, $\leftarrow l_1,\ldots,l_s$ has been selected. Moreover, H_i is a set of hypotheses and, for some $l = l_{j(j=1,\ldots,s)}$, F_{i+1} and H_{i+1} are computed according to these rules:

BC1) if l is not abducible then $F_{i+1} = C' \cup F_i'$ and $H_{i+1} = H_i$, where C' is the set of all the resolvents between clauses of KB and the selected goal on the literal l, such that $\Box \notin C'$;

BC2) if l is an abducible and $l \in H_i$ then $F_{i+1} = \{\leftarrow l'\} \cup F_i'$ and $H_{i+1} = H_i$;

BC3) if l is a base abducible and $l^* \in H_i$ then $F_{i+1} = F_i'$ and $H_{i+1} = H_i$;

BC4) if l is a base abducible, $l \notin H_i$ and $l^* \notin H_i$, then $F_{i+1} = F_i'$ and $H_{i+1} = H_i \cup \{l^*\}$;

BC5) if l is a non–base abducible and $l \notin H_i$ then:
if there exists an abductive derivation from $(\leftarrow l^*, H_i)$ to $(\Box, H')$ then $F_{i+1} = F_i'$ and $H_{i+1} = H'$, else $F_{i+1} = \{\leftarrow l'\} \cup F_i'$ and $H_{i+1} = H_i$.

Cases [BC1], [BC2] and [BC5] are analogous to the corresponding cases of the Dung's procedure. Step [BC3] treats the case that the contrary of a certain base abducible B has been already abduced. In this case the current goal fails because the selected atom (B) must be considered false. Case [BC4] is applicable when there is no assumption on a certain base abducible B and we want to abduce the contrary of B (B^*) so that the current goal fails.

After extending the notion of abductive program and the declarative specification of the procedure with respect to the Dung's definition, we need also to extend the concepts regarding the preferential semantics ([5]).

Definition 6 As *scenario* of an abductive program $\langle KB, IC, Ab\rangle$ we mean an abductive framework $\langle KB \cup \Delta, IC, Ab\rangle$ where $\Delta \subseteq Ab$ and $KB \cup \Delta \cup IC$ is consistent.

Notice that, since $KB \cup \Delta$ is always consistent, inconsistency can arise only when both A and A^* are derivable for some atom A, i.e. when an integrity constraint is violated.

Definition 7 An hypothesis A is *acceptable* with respect to a scenario $\langle KB \cup \Delta, IC, Ab\rangle$ if, for each set E of hypotheses such that $KB \cup E \models A^*$, $E \cup Der(KB, \Delta) \cup \Delta \cup IC$ is inconsistent, where $Der(KB, \Delta) = \{X \in HB \mid KB \cup \Delta \models X\}$.

Definition 8 A *preferred extension* of an abductive program $\langle KB, IC, Ab\rangle$ is a scenario $S = \langle KB \cup \Delta, IC, Ab\rangle$, where Δ is a maximal set with respect to the property that each hypothesis $A \in \Delta$ is acceptable wrt S.

The above definitions are a natural extension of the corresponding definitions in [5]. One can easily prove that the abductive procedure is correct with respect to the preferential semantics.

Theorem 1 (Soundness) *Let $(\leftarrow A, \{\}), \ldots, (\Box, H)$ be an abductive refutation wrt an abductive program $\langle KB, IC, Ab\rangle$. Then there exists a preferred extension $Æ = \langle KB \cup \Delta, IC, Ab\rangle$ such that $H \subseteq \Delta$ and $KB \cup H \models A$.*[2]

3 Handling Further Integrity Constraints.

The procedure presented in the previous section can be extended in various directions. One of them permits to handle a more complex set of integrity costraints. In fact, we can imagine that there is a real integrity theory associated with the deductive database. These constraints, involving abducible predicates, permit to reject and accept certain plausible sets of hypotheses. Despite this, such an extension has a very small impact on the abductive procedure. When a new hypothesis is abduced, the procedure resolves it with a single forward step against all the integrity costraints that "contains" it. Then, it tries to reason backward from any resolvent obtained in this way. The next example will show what problems and difficulties can occur in this context.

Example 4 Let us consider the following framework:

$$KB: \begin{array}{l} s \leftarrow a \\ p \leftarrow q^* \\ q \leftarrow b \end{array} \qquad I: \begin{array}{l} \leftarrow a, p \\ \leftarrow a, q \end{array}$$

In this extended abductive framework I is the set of integrity costraints associated with the database, and a,b are both base abducibles. Moreover, suppose that the top query is $\leftarrow s$.

[2]Due to lack of space the proof is omitted but it can be obtained from the author.

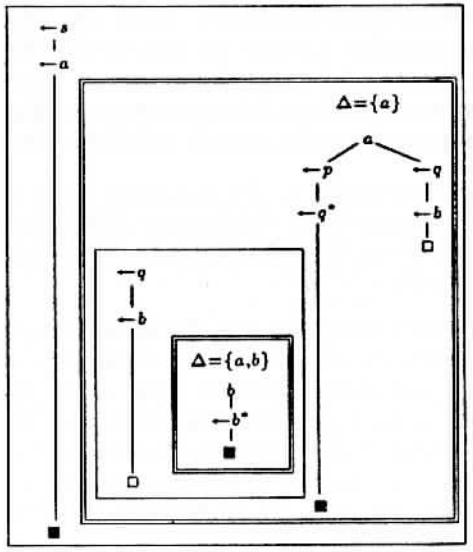

Figure 4: The extended update procedure.

When the goal $\leftarrow a$ is selected within the main derivation, since neither a nor a^* have been already abduced, we can assume the atom a. But this is not sufficient anymore, because it is also necessary that all the resolvents between the abduced atom and the integrity constraints of the database fail. So two consistency derivations on the goals $\leftarrow p$ and $\leftarrow q$ are opened. Now, the computation proceeds in the usual way, because the only atom that will be abduced in the sequel is b and it admits no resolvent against the constraints of I. The whole computation fails. because the abduction of b, that is necessary to guarantee that the first constraint is satisfied, causes the violation of the latter. Note that the start query would be satisfied without these new constraints.

To give the formal specification of the new problem we want to treat, we need to modify the overall context. In fact, we must further extend the notion of abductive program to cover a bigger class than the one considered in the previous section.

Definition 9 Given a logic program P, an *abductive program* associated to it is an abductive framework $\langle P^*, IC, Ab\rangle$ such that P^* and Ab are defined exactly as in definition 2, whereas the set IC of integrity constraints is now given by $IC = IC^* \cup IC^{DB}$, with $IC^* = \{\leftarrow q(x), q^*(x) \mid q$ is a predicate symbol occurring in $P\}$ and IC^{DB} is the integrity theory associated with the database, that is a set of denials of the form $\leftarrow A_1, \ldots, A_n$ where at least an A_i is base abducible.

The integrity constraints occurring in this new definition of abductive program belong to quite a restricted class. However we think this type of constraints is not too restrictive in this particular context. In fact, in the field of view updates for deductive databases it is reasonable to be able to express view predicates in terms of base predicates. Moreover, if this dependence is not expressed in a direct way, we can try to make clear such dependences by initially unfolding non-base atoms.

The impact on the base version of the procedure is not substantial. The only modification regards the cases when a certain atom A is abduced. Now we also have to fail all the resolvents between A and the integrity constraints of the set IC^{DB}. Let us look briefly at this aspect.

Without loss of generality we can suppose that in the current state of the database all the integrity constraints are satisfied. These constraints involve abducibles, i.e. only the abduction of a new fact can violate them. Then, at every abduction the procedure will have to open a consistency derivation on the resolvents between all the constraints and the new hypothesis, so that these restrictions carry on holding. Only if this derivation succeeds, that is we can guarantee that the whole integrity theory is satisfied also after the last abduction, we can continue the computation. We are now ready to formalize the previous ideas. Let $\langle KB, IC, Ab\rangle$ be an abductive program as in the previous definition and let R be a safe selection rule.

Definition 10 An *abductive derivation* from (G_1, H_1) to (G_n, H_n) is a sequence $(G_1, H_1), (G_2, H_2), \ldots, (G_n, H_n)$ such that, for each i, with $0 < i \leq n$, G_i has the form $\leftarrow l, l'$ where, without loss of generality, R selects l and l' is a (possibly empty) conjunction of atoms. Moreover, H_i is a set of hypotheses and G_{i+1}, H_{i+1} are computed according to these rules:

EA1) if l is not abducible then $G_{i+1} = C$ and $H_{i+1} = H_i$, where C is the resolvent of some clause in KB with G_i on the selected literal l;

EA2) if l is abducible and $l \in H_i$ then $G_{i+1} = \leftarrow l'$ and $H_{i+1} = H_i$;

EA3) if l is a base abducible, $l \notin H_i$, $l^* \notin H_i$ and there exists a consistency derivation from $(Rs, H_i \cup \{l\})$ to $(\{\}, H')$, where Rs is the set of all the resolvents between l and the integrity constraints of the deductive database (IC^{DB}), then $G_{i+1} = \leftarrow l'$ and $H_{i+1} = H'$;

EA4) if l is a non-base abducible, $l \notin H_i$ and there exists a consistency derivation from $(\{\leftarrow l^*\} \cup Rs, H_i \cup \{l\})$ to $(\{\}, H')$, where Rs is the set of all the resolvents between l and the integrity constraints of the deductive database (IC^{DB}), then $G_{i+1} = \leftarrow l'$ and $H_{i+1} = H'$.

Case [EA3] is applied if the currently selected atom is a base abducible on which we have not made any assumption yet. Unlike the base version, we cannot directly add it to Δ, but it is necessary to open a consistency derivation on the set Rs of all the resolvents between the current selection and the database's constraints, to

guarantee that they carry on holding also after the new abduction. Analogously we can understand the definition of case [EA4]. The base version instead tried only to fail every proof for the complement of the abduced atom.

Definition 11 Let ϕ be a conjunction of atoms. Then, an *abductive refutation* for ϕ is an abductive derivation from $(\leftarrow \phi, \{\})$ to $(\square, H)$, for some H.

Definition 12 A *consistency derivation* from (F_1, H_1) to (F_n, H_n) is a sequence $(F_1, H_1), (F_2, H_2), \ldots, (F_n, H_n)$ such that, for each i, with $0 < i \leq n$, F_i is a set of goals of the form $\{\leftarrow l_1, \ldots, l_s\} \cup F_i'$ where, without loss of generality, $\leftarrow l_1, \ldots, l_s$ has been selected. Moreover, H_i is a set of hypotheses and, for some $l = l_{j(j=1,\ldots,s)}$, F_{i+1} and H_{i+1} are computed according to these rules:

EC1) if l is not abducible then $F_{i+1} = C' \cup F_i'$ and $H_{i+1} = H_i$, where C' is the set of all the resolvents between clauses of KB and the selected goal on the literal l, such that $\square \notin C'$;

EC2) if l is an abducible and $l \in H_i$ then $F_{i+1} = \{\leftarrow l'\} \cup F_i'$ and $H_{i+1} = H_i$;

EC3) if l is a base abducible and $l^* \in H_i$ then $F_{i+1} = F_i'$ and $H_{i+1} = H_i$;

EC4) if l is a base abducible, $l \notin H_i$ and $l^* \notin H_i$, then $F_{i+1} = Rs \cup F_i'$ and $H_{i+1} = H_i \cup \{l^*\}$, where Rs is the set of all the resolvents between l^* and the integrity constraints of the database (IC^{DB});

EC5) if l is a non-base abducible and $l \notin H_i$ then:
if there exists an abductive derivation from $(\leftarrow l^*, H_i)$ to $(\square, H')$ then $F_{i+1} = F_i'$ and $H_{i+1} = H'$, else $F_{i+1} = \{\leftarrow l'\} \cup F_i'$ and $H_{i+1} = H_i$.

For case [EC4] the same considerations made with respect to step [EA3] apply. We want again to assume an atom, namely the complement of the current selection, and we must fail on all the resolvents between this atom and all the elements of IC^{DB}. Obviously, it is not necessary to open a new consistency derivation, because we are already within a consistency derivation. So, we need only to add these resolvents to the set of the remaining goals we want to fail. The definition of step [EC5] is analogous to the one of the corresponding step in the base version, because we have to assume no fact.

Before presenting the soundness result of this extended procedure we have to define a declarative semantics for this extended notion of abduction. The correctness of the procedure will be proved with respect to a small extension of the semantics presented in the previous section. Note that the definitions given in the previous section are particular cases of the ones that we are giving in this part of the paper.

Definition 13 Let $\langle KB, IC, Ab\rangle$ be an abductive program associated to a certain logic program as in definition 9. A *scenario* is an abductive framework $\langle KB \cup \Delta, IC, Ab\rangle$ where $\Delta \subseteq Ab$ and $KB \cup \Delta \cup IC$ is consistent.

Note that IC also contains the constraints of the integrity theory associated with the database.

Definition 14 An hypothesis A is *acceptable* with respect to a scenario $\langle KB \cup \Delta, IC, Ab\rangle$ if, for each $\leftarrow \phi \in IC$ that admits a resolvent $\leftarrow A_1, \ldots, A_n$ against A, and for each set E of hypotheses such that $KB \cup E \models A_1 \wedge \ldots \wedge A_n$, $E \cup Der(KB, \Delta) \cup \Delta \cup IC$ is inconsistent, where $Der(KB, \Delta) = \{X \in HB \mid KB \cup \Delta \models X\}$.

Definition 15 A *preferred extension* of an abductive program $\langle KB, IC, Ab\rangle$ is a scenario $S = \langle KB \cup \Delta, IC, Ab\rangle$, where Δ is a maximal set with respect to the property that each hypothesis $A \in \Delta$ is acceptable wrt S.

Again, we can state a soundness result. In fact, the extended version of the procedure is sound with respect to the new declarative semantics.

Theorem 2 (Soundness) *Let* $(\leftarrow A, \{\}), \ldots, (\square, H)$ *be an abductive refutation wrt an abductive program* $\langle KB, IC, Ab\rangle$. *Then there exists a preferred extension* $\mathcal{E} = \langle KB \cup \Delta, IC, Ab\rangle$ *such that* $H \subseteq \Delta$ *and* $KB \cup H \models A$.[3]

4 Conclusions and Further Works

In this paper we have shown that the abductive view of the update problem in deductive databases originally presented in [11] can be given a declarative semantics obtained by extending in a natural way Dung's preferential semantics for normal logic programming. Furthermore, we have extended the proof procedure of [11] to handle also an interesting class of update problems, where an integrity theory is associated with the database. This extension can also be given a declarative semantics by a natural extension of the semantics for the base procedure. In [13] we have further extended this framework in order to handle also non-ground hypotheses by using a technique based on skolemization of non-ground abducibles (similar to the technique used by [4] in the so-called SLDNFA-resolution for abductive logic programs). The interested reader can refer to [13] for further details.

Acknowledgements

We thank Paolo Mancarella for its precious collaboration and many helpful suggestions. Work partially supported by the EEC activity KIT011-LPKRR.

[3] Due to lack of space the proof is omitted but it can be obtained from the author.

References

[1] Banchilon, F., Spyratos, N., Update semantics of relational views, ACM TODS, Vol. 6, No 4, 1981

[2] Cosmadakis, C.C., Papadimitriu, C.H., Updates of Relational Views, JACM 31 (4) pp. 742–760, 1984

[3] Dayal, U., Bernstein, P.A., On the correct translation of Update Operations on Relational Views, ACM TODS, Vol. 8, No 3, 1982

[4] Denecker, M., De Schreye, D., SLDNFA: an abductive procedure for normal abductive programs. *Proc. of the Joint International Conference and Symposium on Logic Programming*, pp. 686–700, 1992

[5] Dung, P. M., Negations as hypotheses: an abductive foundation for logic programming. *Proc. 8th International Conference on Logic Programming*, Paris (1991), pp. 3–17

[6] Eshghi, K., Kowalski, R.A., Abduction Compared with Negation by Failure, *Proc. 6th ICLP 89*, MIT Press, 1989

[7] Fagin, R., Ullman, J., Vardi, M., On the Semantics of Updates in Databases, Proc. 2nd ACM PODS, 1983

[8] Furtado, A.L., Casanova, M.A., Updating Relational Views, in W. Kim et al (eds.), Query Processing in Database Systems. Springer Verlag, 1985

[9] Gottlob, G., Paolini, P., Zicari, R., Properties and Update Semantics of Consistent Views, ACM TODS, Vol. 13, No 4, 1988

[10] Kakas, A.C., Kowalski, R.A., Toni, F., Abductive Logic Programming. *Journal of Logic and Computation*, 2, pp. 719–770, 1992

[11] Kakas, A.C., Mancarella, P., Database updates through abduction. *Proc. of the 16th International Conference on Very Large Data Bases*, Brisbane, August 13–16, 1990

[12] Kowalski, R.A., Logic for Problem Solving, Elsevier North–Holland, Amsterdam, 1979

[13] Plagenza, G., "Aggiornamento abduttivo di basi di dati deduttive.", Tesi di laurea, Dipartimento di Informatica, Università di Pisa, 1993 (in Italian)

[14] Sadri, F., Kowalski, R.A., A Theorem Proving Approach to Database Integrity, in J. Minker (ed.) *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufman, 1988