Update Rules in Datalog Programs*

M. Halfeld Ferrari Alves[2,3], D. Laurent[1,2], N. Spyratos[2]

1. Université d'Orléans, LIFO, F-45067 Orléans Cedex 2, France

2. Université de Paris-Sud, LRI, U.R.A. 410 du CNRS, Bât. 490
F-91405 Orsay Cedex, France

3. Universidade Federal do Paraná, UFPR, Dep. Info., Curitiba, Brazil

Abstract

We consider Datalog^{neg} databases containing two kinds of rules: update rules of the form $L_0 \leftarrow L_1$, where L_0 and L_1 are literals, and query rules as in general logic programs. We regard update rules as constraints, *all* consequences of which must hold in the database until a new update. As a consequence, update-driven derivations act as exceptions to query-driven derivations. We introduce a semantics framework for database update and query answering, based on the well-founded semantics. In this framework, updating over intensional predicates is deterministic. A basic feature of our approach is that new knowledge inputs are always assimilated in such a way that the consistency of the database is preserved.

*Part of this work has appeared in the form of extended abstract in the Proceedings of the Third Logic Programming and Non-Monotonic Reasoning Conference, Lexington, KY, USA, June 1995.

Work partially supported by the French National Project GDR-PRC *BD3* and by the Brazilian Government (CNPq/UFPR).

1 Introduction

In this paper, we present formal semantics for Datalog databases containing two kinds of rules: *update rules* and *query rules.* Update rules are activated during an update request and query rules are activated during a query request. Thus the database comprises two inference mechanisms, one for updates and one for queries. We regard update rules as constraints *all* consequences of which hold in the database until a new update. In the remaining of this section, we illustrate by examples how these mechanisms work and how they interact with each other.

Update Rules

In our context, an update means the insertion or deletion of a *single* fact in the database, and an update rule describes a possible side effect of an update. Therefore, the update rules that we consider are rules of the form $L_0 \leftarrow L_1$ where L_0 and L_1 are literals. The literal L_1 in the body represents the fact being inserted or deleted while the literal L_0 in the head represents a side effect. We denote an update as $l \leftarrow$, where l is a ground literal, *i.e.,* we denote updates as update rules without body. Moreover, if $l = A$, where A is a ground atom, then we write $A \leftarrow$ and we read *insert* A, and if $l = \neg A$ then we write $\neg A \leftarrow$ and we read *delete* A. We illustrate the intuitive meaning of updates and update rules in the following example.

Running Example 1.1 Consider the predicate symbols cs, $math$ and sci (meaning computer scientist, mathematician and scientist, respectively) together with the predicate symbols $crazy$, $happy$ and sad (with their obvious meaning). We shall use the following updates and update rules as our running example.

Updates:

U_1 : $sci(Bob) \leftarrow$

U_2 : $math(Bob) \leftarrow$

Update rules:

$$\begin{array}{lrcl} UR_1: & \neg\, crazy(X) & \leftarrow & cs(X) \\ UR_2: & \neg\, cs(X) & \leftarrow & crazy(X) \\ UR_3: & sad(X) & \leftarrow & \neg\, crazy(X) \\ UR_4: & \neg\, happy(X) & \leftarrow & sad(X) \end{array}$$

Update U_1 means *insert* $sci(Bob)$ and update U_2 means *insert* $math(Bob)$. On the other hand, rule UR_1 means that if the ground literal $cs(a)$ becomes true then $crazy(a)$ must become false; UR_3 means that if $crazy(a)$ becomes false then $sad(a)$ must become true; and so on. We note that the body of two or more update rules can be the same literal. □

We note here an important difference between our approach and that of [MT95]. In [MT95], a literal can be either "true" or "false" while, in our approach, it can also be "unknown". Moreover, the interpretation of deletion is different in both approaches. Indeed, in our approach, "delete l" means that the negation of l is true, *i.e.,* deleting l corresponds to the insertion of its negation into the database. In the approach of [MT95], "$out(l)$" means that l is removed from the database and, in this context, as l is not in the database, it is considered to be false. We refer to [HLS95b] for a detailed comparison between our approach and that of [MT95].

An important aspect of update rules is that they act as constraints. For instance, rules UR_1 and UR_2 together express the constraint that nobody can be *computer scientist* and *crazy* at the same time.

Insertions and deletions are implemented by storing positive or negative literals in the database. But storing both positive and negative literals can generate inconsistencies. The system is going to perform actions in order to restore consistency, as shown by the following example.

Running Example 1.2 The updates U_1 and U_2 are implemented by storing the positive literals $sci(Bob)$ and $math(Bob)$ in the database. Denoting by $\mathcal{L}$ the set of stored literals, we have

$$\mathcal{L} = \{sci(Bob),\ math(Bob)\}.$$

As neither of the predicates *sci* or *math* appears in the body of an update rule, no update rule is activated. If we insert now $crazy(Bob)$ then rule UR_2 is activated and derives $\neg\ cs(Bob)$. Since the set $\mathcal{L}$ does not contain $cs(Bob)$, there is no inconsistency and the insertion is performed by simply storing $crazy(Bob)$ in the set $\mathcal{L}$. Note that we do *not* store $\neg\ cs(Bob)$ in $\mathcal{L}$ but we do take it into account in deciding consistency (as will be explained shortly). Thus, after the insertion of $crazy(Bob)$, we have:

$$\mathcal{L} = \{sci(Bob),\ math(Bob),\ crazy(Bob)\}.$$

Suppose next that we insert $cs(Bob)$. Then rule UR_1 is activated and derives $\neg\ crazy(Bob)$, thus creating an inconsistency with $\mathcal{L}$. In order to restore consistency we remove $crazy(Bob)$ from $\mathcal{L}$ and the updated database becomes:

$$\mathcal{L} = \{sci(Bob),\ math(Bob),\ cs(Bob)\}. \qquad \square$$

It is important to note that, our approach assimilates new knowledge inputs in such a way that the consistency of the database is preserved. However, in order to maintain consistency, it may be necessary to remove previously stored literals. Thus, for example, the insertion of $cs(Bob)$ above required the removal of $crazy(Bob)$. In doing so, our model privileges "new knowledge" over "old knowledge." In contrast, conventional database systems, such as relational systems, privilege old knowledge. This is manifest in the way insertions are done in relational systems: the insertion of a tuple is accepted only if the resulting database satisfies the constraints, otherwise the insertion is rejected.

In our model, database updating comprises two distinct levels, the *user level* and the *system level.* At user level, the user disposes of two operations, *insert* and *delete.* At system level, the system disposes again of two operations, *store* and *remove.* An insert or delete request at user level is translated by the system into a sequence of store/remove operations whose purpose is twofold: to store the positive or negative literal as requested by the user, and to maintain database consistency. Thus, for example, *insert* $cs(Bob)$ was translated by the system into the sequence: *remove* $crazy(Bob)$; *store* $cs(Bob)$. We note that in conventional database systems, such as relational systems, the two levels of database updating coincide, *i.e., insert f* is always translated as *store f* and *delete f* is always translated as *remove f.*

In the previous example, we saw that the update rules were activated during an insertion or deletion in order to compute the new set $\mathcal{L}$, based on the old and on the literal being inserted or deleted. Now, given that the literals of $\mathcal{L}$ hold in the database, the update rules imply that some other literals must also hold. Let us illustrate this point by an example.

Running Example 1.3 We recall that the set of stored literals after the insertion of $cs(Bob)$ is

$$\mathcal{L} = \{sci(Bob),\ math(Bob),\ cs(Bob)\}.$$

As these literals hold in the database, applying the update rules, we find that the literals $\neg crazy(Bob)$, $sad(Bob)$ and $\neg happy(Bob)$ must also hold. Let $\xi_{\mathcal{L}}$ denote the set of all ground literals that must hold in the database. The set $\xi_{\mathcal{L}}$ consists of all literals in $\mathcal{L}$ and their side effects, *i.e.,* all literals that can be obtained from $\mathcal{L}$ by repeated application of the update rules. It follows that, after the insertion of $cs(Bob)$, we have:

$$\xi_{\mathcal{L}} = \{sci(Bob),\ math(Bob),\ cs(Bob),\ \neg crazy(Bob),\ sad(Bob),\ \neg happy(Bob)\}. \qquad \square$$

So when an update request is addressed to the database, the update rules are activated to compute:

1. the new set $\mathcal{L}$, based on the old (*i.e.,* the one before the update) and the ground literal being inserted or deleted,

2. the new set $\xi_{\mathcal{L}}$ of all literals that must hold in the database until the next update.

We note again that the only literals stored are those of $\mathcal{L}$. The literals of $\xi_{\mathcal{L}}$ that are not in $\mathcal{L}$, although they are not stored, are necessary for the processing of queries (as we shall see shortly).

These computations reflect our philosophy of regarding update rules as constraints, *all* consequences of which must hold in the database. Now, the *only* event that can change the truth values of the literals in $\xi_{\mathcal{L}}$ is a new update. However, until that new update happens, the literals of $\xi_{\mathcal{L}}$ continue to hold in the database and form the basis for query answering.

Query Rules

The query rules that we consider in this paper are standard Datalog^{neg} rules. However, in order to perform query processing we must first settle two matters.

(1) Our update rules allow explicit derivation of negative information, so we can consider that the falsity of a literal corresponds to classical negation. On the other hand, to perform query processing, we use well-founded semantics [GRS91], with one important difference. Namely, in the usual computation of well-founded semantics, the initial interpretation is empty, whereas, in our model, the initial interpretation is $\xi_{\mathcal{L}}$. As a consequence, two kinds of negation are present: *classical negation*, denoted by $\neg$, and *negation of well-founded semantics*, denoted by *not*. Now, classical negation does not coincide with the negation of well-founded semantics because, in well-founded semantics, negation is not produced explicitly by rules. So, if we want to use negative literals of $\xi_{\mathcal{L}}$ with query rules, we must first say how classical negation is related to negation of well-founded semantics. In this respect we make the following assumption (in the spirit of [AP92, GL90a]):

- **Assumption:** for every atom A, $\neg A$ is transformed into *not* A.

(2) As the literals of $\xi_{\mathcal{L}}$ hold in the database (and must hold until a new update) the literals derived by query rules must not be in contradiction with those of $\xi_{\mathcal{L}}$. We cope with this problem by conditioning the derivation of literals through query rules as follows:

- if l is in $\xi_{\mathcal{L}}$ then l holds
  else if $\neg\ l$ is not in $\xi_{\mathcal{L}}$ then
  if there is a query rule $l \leftarrow L_1, \ldots, L_n$ and $L_1, \ldots, L_n$ hold then l holds.

In a sense, the literals of $\xi_{\mathcal{L}}$ act as exceptions to query rule derivations. We feel therefore justified in calling $\xi_{\mathcal{L}}$ the *exception set* of the database, and the literals of $\xi_{\mathcal{L}}$ the *exceptions*.

Running Example 1.4 Suppose that we have the following query rules (along with the update rules UR_1, UR_2, UR_3 and UR_4 seen earlier):

Query rules:

$$\begin{array}{lrcl} QR_1: & crazy(X) & \leftarrow & sci(X),\ not\ cs(X) \\ QR_2: & cs(X) & \leftarrow & math(X),\ not\ crazy(X) \\ QR_3: & happy(X) & \leftarrow & math(X),\ cs(X) \end{array}$$

As $math(Bob)$ and $cs(Bob)$ are in $\xi_{\mathcal{L}}$, QR_3 applies and derives $happy(Bob)$. However, as $\neg happy(Bob)$ is in $\xi_{\mathcal{L}}$ this derivation is not valid, *i.e.,* $\neg happy(Bob)$ remains true. $\Box$

As we assume the transformation $\neg A \rightarrow not\ A$, we feel justified in using, henceforth, *only* one negation symbol, namely *not*.

In the following Section 2 we recall briefly some facts concerning Datalog^{neg} databases under well-founded semantics. In Section 3, we introduce our basic definitions and notations concerning what we call *literal rules,* whose purpose is to model update rules. We also discuss literal rules consistency.

In Section 4, we present formal semantics for databases containing update and query rules and, in Section 5, we show how we can compute this semantics by usual Datalog. In Section 6 we present formal semantics for database updating. In Section 7, we describe briefly the principal ideas of [GL90a] and [KS90] in the context of extended logic programs. In Section 8, we show how our approach is related to extended logic programs, and in Section 9, we discuss alternatives offered by our approach. Finally, in the Section 10 we offer some concluding remarks.

2 Datalog^{neg} Databases

A Datalog^{neg} database is seen as a finite function-free set of facts and rules over an alphabet $\mathbf{A}$ consisting of constants, variables and predicates [CGT90, Ull89]. There is an infinite set CONST of constants usually denoted by lower case characters such as a, b, etc and an infinite set VAR of variables usually denoted by lower case characters such as x, y, etc. There is a *finite* set PRED of predicates usually denoted by lower case characters such as p, q, etc; each predicate is associated with a positive integer called its arity. We assume that the sets CONST, VAR and PRED are mutually disjoint. We note that, since we consider function-free languages, the only possible terms are constants or variables.

As a notational convenience, if p is an n-ary predicate, and if $t_1, t_2, \ldots, t_n$ are terms, then the formula $p(t_1, t_2, \ldots, t_n)$ is denoted by $p(\widetilde{t})$. A formula of the form $p(\widetilde{t})$ is referred to as *positive literal* and a formula of the form *not* $p(\widetilde{t})$ is referred to as *negative literal.* A literal is said to be *ground* if it contains no variable. A positive ground literal is also called a *fact.*

A *rule* is a formula of the form $p(\widetilde{t}) \leftarrow L_1, L_2, \ldots, L_k$ where, for every $i = 1, 2, \ldots, k$, L_i is a literal. The positive literal $p(\widetilde{t})$ is called the *head* of the rule and the set of literals $L_1, L_2, \ldots, L_k$ is called the *body* of the rule.

We now recall the definition of well-founded semantics of Datalog^{neg} [GRS91, Bid91]. Well-founded semantics are based on the notion of *partial* interpretation defined as follows. Let HB be the Herbrand base of a database alphabet $\mathbf{A}$, *i.e.,* HB is the set of all facts that can be obtained from $\mathbf{A}$. For every subset S of HB, let $not.S$ denote the set $\{not\ f \mid f \in S\}$. In particular, $not.HB$ denotes the set $\{not\ f \mid f \in HB\}$. A subset C of $HB \cup not.HB$ is called consistent if C does not contain a fact f and its negation *not* f. That is, C is *consistent* if there are disjoint subsets $pos(C)$ and $neg(C)$ of HB such that $C = pos(C) \cup not.neg(C)$. A *partial interpretation* of $\mathbf{A}$ is just a consistent subset I of $HB \cup not.HB$.

A Datalog^{neg} database is a set of facts and rules denoted as $DB = (F, R)$, where F stands for the set of facts and R for the set of rules. We denote by $inst\_DB$ the set of all facts in F together with all instantiations of the rules in R. Given a partial interpretation I and a Datalog^{neg} database DB, consider the following operators.

- Define the *immediate consequence* operator $T^{\in}$ by:
$$T^{\in}_{DB}(I) = \{head(r) \mid r \in inst\_DB \ \wedge \ \forall L \in body(r), L \in I\}.$$
- Define a set of facts U to be unfounded with respect to I, if for all f in U and for all instantiated rules r in $inst\_DB$ the following holds:
$$(head(r) = f) \ \Rightarrow \ \exists L \in body(r),\ [not\ L \in I \vee L \in U].$$
Define the *unfounded-set* operator U_{DB} by: $U_{DB}(I)$ is the greatest set of unfounded facts with respect to DB and I.

It can be shown [GRS91] that the operator $T^{\in}_{DB} \cup not.U_{DB}$ has a least fixed point which is a partial interpretation of $\mathbf{A}$. It is precisely this fixed point that is referred to as the *well-founded model* of DB.

3 Literal Rules

In this section, we introduce literal rules in order to model update rules. A *literal rule* is a rule whose head and body contain only one literal.

Definition 3.1 - Literal Rule. *A literal rule, or l-rule for short, is a rule of the form $L_1 \leftarrow L_2$, where L_1 and L_2 are literals.* □

It is important to note the differences between a literal rule and a usual Datalog^{neg} rule: the head of a literal rule *can be a negative* literal, while the head of a usual Datalog^{neg} rule is *always a positive* literal; on the other hand, the body of a literal rule contains only one literal, whereas the body of a usual Datalog^{neg} rule may contain more than one literal.

Given a set of l-rules LR, we denote by *inst*_LR the set of all instantiations of the rules in LR. We associate LR with an operator $^{\mathrm{LR}}\xi$, called *the exception operator*, defined as follows: for every set of ground literals I of the underlying alphabet $\mathbf{A}$,

$$^{\mathrm{LR}}\xi(I) = I \cup \{head(r) \mid r \in inst\_\mathrm{LR} \wedge body(r) \in I\}.$$

Clearly, $^{\mathrm{LR}}\xi$ is a monotonic operator over the power set of $HB \cup not.HB$. Now, as $^{\mathrm{LR}}\xi$ is monotonic, the sequence defined by:

$$^{\mathrm{LR}}\xi^0(I) = I \qquad \text{and} \qquad {}^{\mathrm{LR}}\xi^k(I) = \xi(\xi^{k-1}(I)), \;\; \text{for every integer } k > 0,$$

has a limit. This limit, referred to as the *least fixed point of* $^{\mathrm{LR}}\xi$ *with respect to* I, is denoted by *lfp*$(^{\mathrm{LR}}\xi, I)$ or by $^{\mathrm{LR}}\xi_I$.

In the remaining of this paper, we shall use the following simplified notations: when a specific set of literal rules is understood, we shall simply write ξ instead of $^{\mathrm{LR}}\xi$, and a singleton $\{L\}$ is simply denoted by its single element L. Thus, for example, we shall write ξ_L instead of $^{\mathrm{LR}}\xi_{\{L\}}$.

Due to their particular form, update rules enjoy some useful properties, as stated in the following lemma.

Lemma 3.1 *Let* LR *be a set of l-rules, and let* ξ *be the associated exception operator. For every partial interpretation* I*, we have:* (1) $I \subseteq \xi_I$*, and* (2) $\xi_I = \bigcup_{L \in I} \xi_L$. □

Proof: (1) It is clear from the definition of ξ that $I \subseteq \xi^k(I)$, for every $k \geq 0$. Thus, $I \subseteq \xi_I$.

(2) We first note that $\xi(I) = \bigcup_{L \in I} \xi(L)$. Indeed, by monotonicity of ξ, $\xi(L) \subseteq \xi(I)$, for every L in I. Thus, $\xi(I) \supseteq \bigcup_{L \in I} \xi(L)$. On the other hand, let L' be in $\xi(I)$. If $L' \in I$, then $L' \in \bigcup_{L \in I} \xi(L)$, because $L' \in \xi(L')$. Otherwise, if $L' \notin I$, then *inst*_LR contains a rule $L' \leftarrow L$ where L is a literal in I. Therefore, $L' \in \xi(L)$, which entails that $L' \in \bigcup_{L \in I} \xi(L)$.

It is easy to see by induction that, for every $k \geq 0$, the above reasoning holds. Thus, for every $k \geq 0$, $\xi^k(I) = \bigcup_{L \in I} \xi^k(L)$. This shows that $\xi_I = \bigcup_{L \in I} \xi_L$, which completes the proof of the lemma. □

We note that, since the head and the body of every l-rule contains a single literal, we can associate every set LR of literal rules with an "inverse" operator $^{\mathrm{LR}}\vartheta$. This operator is defined as follows: for every set I of ground literals, define

$$^{\mathrm{LR}}\vartheta(I) = I \cup \{body(r) \mid r \in inst\_\mathrm{LR} \wedge head(r) \in I\},$$

Using our previous convention about simplified notations, when no confusion is possible, $^{\mathrm{LR}}\vartheta$ will be denoted by ϑ. Then, it is easy to see that, for every set of literal rules LR and for all literals L and L', we have: $L \in \xi(L') \Leftrightarrow L' \in \vartheta(L)$.

Clearly, ϑ enjoys the same properties as ξ (see [HLS95a]). As a consequence, ϑ is monotonic, and we denote by ϑ_I the least fixed point of ϑ with respect to a set of ground literals I. Moreover, ϑ satisfies points (1) and (2) of Lemma 3.1, *i.e.* $I \subseteq \vartheta_I$ and $\vartheta_I = \bigcup_{L \in I} \vartheta_L$, for every partial interpretation I.

Since update rules may have positive or negative literals in their heads, the important problem of *consistency* raises here. Indeed, even if I is a partial interpretation, it may happen that ξ_I is *not* a partial interpretation. For example, if LR consists of the instantiated rules: $q \leftarrow p$ and $not\ q \leftarrow p$, then for $I = \{p\}$, we have $\xi_I = \{p, q, not\ q\}$, which is not a partial interpretation. Intuitively, the above two l-rules mean that when p is inserted then q must become both true *and* false, and this is an inconsistency. We call a set of l-rules *consistent* if it does not generate such inconsistencies.

Definition 3.2 - Consistent Set of L-Rules. *A set* LR *of l-rules is* consistent *if for every ground literal L the set ξ_L is a partial interpretation.* □

We now characterize consistent sets of update rules. To this end, we first define the notion of *rule path,* based on the concept of unification, as defined in [Llo87]. We would like to point out here that, as usual in logic programming, we assume that each time a l-rule r is used in a derivation, all variables occurring in r are new variables, never considered before.

Definition 3.3 - Rule Path. *Let* LR *be a set of l-rules and let $\pi = [r_1, r_2, \ldots, r_k]$ be a sequence of l-rules of* LR. *π is called a* rule path *if $k = 1$ or if, for every $i = 1, 2, \ldots, k-1$, $head(r_i)$ and $body(r_{i+1})$ are unifiable.* □

As a consequence of Definition 3.3, a single l-rule always constitutes a rule path. Now, based on the notion of *most general unifier,* or mgu for short ([Llo87]), we associate every rule path π of $G(\text{LR})$ with a unifier, denoted by $\theta(\pi)$, and defined as follows:

Definition 3.4 - Unifier of a Rule Path. *Let* LR *be a sequence of l-rules and let $\pi = [r_1, r_2, \ldots, r_k]$ be a rule path. The* unifier *of π, denoted by $\theta(\pi)$ is defined as follows:*

1. *If $k = 1$ then $\theta(\pi)$ is the identity substitution.*
2. *Otherwise, let θ_i be the most general unifier of $head(r_i)$ and $body(r_{i+1})$, for every $i = 1, 2, \ldots, k-1$. Then $\theta(\pi) = \theta_{k-1} \ldots \theta_2 \theta_1$.* □

Example 3.1 Let LR be the set containing the following three l-rules:

$$p(a, x) \leftarrow q(a, b) \qquad not\ p(a, b) \leftarrow q(a, y) \qquad q(a, z) \leftarrow not\ q(a, z).$$

The following sequence of update rules

$$\pi = [q(a, z) \leftarrow not\ q(a, z),\ p(a, x) \leftarrow q(a, b)]$$

is a rule path, whose unifier $\theta(\pi)$ is defined by the binding z/b. □

The following lemma shows that derivation through l-rules is closely related to rule paths.

Lemma 3.2 *Let* LR *be a set of l-rules and let L and L' be two ground literals. L' belongs to ξ_L iff either $L = L'$ or there exists a rule path $\pi = [r_1, r_2, \ldots, r_k]$ and a substitution σ such that $L = body(r_1)\theta(\pi)\sigma$ and $L' = head(r_k)\theta(\pi)\sigma$.* □

Proof: The case $L = L'$ being trivial, we assume that $L \neq L'$ and that $L' \in \xi_L$. It follows that there exists $k > 0$ such that $L' \in \xi^k(L)$. Therefore, $inst\_\text{LR}$ contains k instantiated l-rules $\widetilde{r_1}, \widetilde{r_2}, \ldots, \widetilde{r_k}$ such that $body(\widetilde{r_1}) = L$, $head(\widetilde{r_k}) = L'$ and $head(\widetilde{r_i}) = body(\widetilde{r_{i+1}})$, for $i = 1, 2, \ldots, k-1$. As a consequence, $\pi = [r_1, r_2, \ldots, r_k]$ is a rule path. If $k = 1$, then $L = body(\tilde{r}_1)$ and $L' = head(\tilde{r}_1)$. As, in this case, $\theta(\pi)$ is the identity, σ is the substitution such that $L = body(\tilde{r}_1)$. If $k \neq 1$, then, by definition of the mgu, for every $i = 1, 2, \ldots, k-1$, $head(\tilde{r}_i)$ is an instance of $head(r_i)\theta_i$ and $body(\widetilde{r_{i+1}})$ is an instance of $body(r_{i+1})\theta_i$. Therefore, L is an instance of $body(r_1)\theta(\pi)$. Let σ_1 be the substitution such that $L = body(r_1)\theta(\pi)\sigma_1$, and let σ_2 be the substitution such that $L' = head(r_k)\theta(\pi)\sigma_2$. Since variables occurring in $body(r_1)\theta(\pi)$ and in $head(r_k)\theta(\pi)$ are not the same, $\sigma = \sigma_1\sigma_2$ is a substitution such that $L = body(r_1)\theta(\pi)\sigma$ and $L' = head(r_k)\theta(\pi)\sigma$.

Conversely, let us assume that $L \neq L'$ and that there exists a rule path $\pi = [r_1, r_2, \ldots, r_k]$ and a substitution σ such that $L = body(r_1)\theta(\pi)\sigma$ and $L' = head(r_k)\theta(\pi)\sigma$. This implies that L' belongs to $\xi^k(L)$, which completes the proof of the lemma. $\square$

The pair $\langle \pi, \sigma \rangle$ where π is a rule path and σ is a substitution as in Lemma 3.2 is referred to as a *derivation path.* Based on the notion of derivation path, we now define what we call an *inconsistent derivation path.*

Definition 3.5 - Inconsistent Derivation Path. *Let* LR *be a set of l-rules, then:*

- *An* inconsistent linear derivation path *is a derivation path $\langle \pi, \sigma \rangle$ such that, if $\pi = [r_1, r_2, \ldots, r_k]$, then: $body(r_1)\theta(\pi)\sigma = not\ head(r_k)\theta(\pi)\sigma$.*
- *An* inconsistent fork derivation path *is a pair of derivation paths $(\langle \pi^1, \sigma^1 \rangle, \langle \pi^2, \sigma^2 \rangle)$ such that, if $\pi^1 = [r^1_1, r^1_2, \ldots, r^1_{k_1}]$ and $\pi^2 = [r^2_2, r^2_2, \ldots, r^2_{k_2}]$, then the rules in r^1_i $(1 \leq i \leq k_1)$ are distinct from the rules r^2_j $(1 \leq j \leq k_2)$ and:*

$$body(r^1_1)\theta(\pi^1)\sigma^1 = body(r^2_1)\theta(\pi^2)\sigma^2 \quad and \quad head(r^1_{k_1})\theta(\pi^1)\sigma^1 = not\ head(r^2_{k_2})\theta(\pi^2)\sigma^2.$$

An inconsistent derivation path *is either an inconsistent linear derivation path or an inconsistent fork derivation path.* $\square$

Example 3.2 As is Example 3.1, let LR be the set containing the following three l-rules:

$$p(a,x) \leftarrow q(a,b) \qquad not\ p(a,b) \leftarrow q(a,y) \qquad q(a,z) \leftarrow not\ q(a,z).$$

Thus, denoting by ϵ the identity substitution, $\pi_1 = [q(a,z) \leftarrow not\ q(a,z)]$ is a rule path whose unifier is ϵ. Moreover, $\langle \pi_1, \epsilon \rangle$ is an inconsistent linear derivation path.
Now, $\pi_2 = [p(a,x) \leftarrow q(a,b)]$ and $\pi_3 = [not\ p(a,b) \leftarrow q(a,y)]$ are also two rule paths whose unifier is ϵ. Moreover, let θ^1 be the substitution that changes x to b and let θ^2 be the substitution that changes y to b. Then, $q(a,b)\theta^1 = q(a,y)\theta^2 = q(a,b)$, and $p(a,x)\theta^1 = p(a,b)\theta^2 = p(a,b)$. As π_2 and π_3 do not contain the same rule, $(\langle \pi_2, \theta^1 \rangle, \langle \pi_3, \theta^2 \rangle)$ is an inconsistent fork derivation path.
Note that, $\pi'_2 = [q(a,z) \leftarrow not\ q(a,z),\ p(a,x) \leftarrow q(a,b)]$ and $\pi'_3 = [q(a,z) \leftarrow not\ q(a,z),\ not\ p(a,b) \leftarrow q(a,y)]$ do *not* define an inconsistent fork derivation path, since π'_2 and π'_3 both contain the third l-rule of LR (see Definition 3.5). This illustrates that derivation paths in inconsistent fork derivation paths are of minimal length. $\square$

The following theorem shows that inconsistent derivation paths characterize exactly the consistency of a given set of l-rules.

Theorem 3.1 *Let* LR *be a set of l-rules. Then,* LR *is* not *consistent iff there exists an inconsistent derivation path in* LR. $\square$

Proof: Assume first that LR is not consistent. Then, by Definition 3.2, there exists a ground literal L and a fact f such that f and *not* f both belong to ξ_L. Three cases occur: (i) $L = f$, (ii) $L = not\ f$, and (iii) $L \neq f$ and $L \neq not\ f$.

(i) $L = f$: By Lemma 3.2, there exists a rule path $\pi = [r_1, r_2, \ldots, r_k]$ and a substitution σ such that $f = body(r_1)\theta(\pi)\sigma$ and $not\ f = head(r_k)\theta(\pi)\sigma$. Therefore, according to Definition 3.5, $\langle \pi, \sigma \rangle$ is an inconsistent linear derivation path.
(ii) $L = not\ f$: Similar to (i) above by exchanging the roles of f and $not\ f$.
(iii) $L \neq f$ and $L \neq not\ f$: In this case, by Lemma 3.2, we have two derivation paths $\langle \pi^1, \sigma^1 \rangle$ and $\langle \pi^2, \sigma^2 \rangle$ such that, if $\pi^1 = [r_1^1, r_2^1, \ldots, r_{k_1}^1]$ and $\pi^2 = [r_2^2, r_2^2, \ldots, r_{k_2}^2]$ then $L = body(r_1^1)\theta(\pi^1)\sigma^1 = body(r_1^2)\theta(\pi^2)\sigma^2$, $f = head(r_{k_1}^1)\theta(\pi^1)\sigma^1$ and $not\ f = head(r_{k_2}^2)\theta(\pi^2)\sigma^2$ (or vice versa). Moreover, the rules $r_{k_1}^1$ and $r_{k_2}^2$ are not the same, since the head of one of these rules is a positive literal and the head of the other is a negative literal. Moreover, as $body(r_1^1)$ and $body(r_1^2)$ are unifiable literals, there exist i_1 $(1 \leq i_1 \leq k_1)$ and i_2 $(1 \leq i_2 \leq k_2)$ such that $\pi_0^1 = [r_{i_1}^1, \ldots, r_{k_1}^1]$ and $\pi_0^2 = [r_{i_2}^2, \ldots, r_{k_2}^2]$ have no common rule and such that $body(r_{i_1}^1)\sigma^1 = body(r_{i_2}^2)\sigma^2$. If σ_0^1 (respectively σ_0^2) denotes the restriction of σ^1 (respectively of σ^2) to the variables occurring the rules of π_0^1 (respectively of π_0^2), then $(\langle \pi_0^1, \sigma_0^1 \rangle, \langle \pi_0^2, \sigma_0^2 \rangle)$ is an inconsistent fork derivation path.

Thus, according to Definition 3.5, $(\langle \pi^1, \sigma^1 \rangle, \langle \pi^2, \sigma^2 \rangle)$ is an inconsistent fork derivation path.

Conversely, assume that $\langle \pi, \sigma \rangle$ with $\pi = [r_1, r_2, \ldots, r_k]$ is an inconsistent linear derivation path. According to Definition 3.5, $body(r_1)\theta(\pi)\sigma = not\ head(r_k)\theta(\pi)\sigma$. Let $L = body(r_1)\theta(\pi)\sigma$, by Lemma 3.2, this implies that ξ_L contains L and $not\ L$, and so, that LR is inconsistent. Assume now that $(\langle \pi^1, \sigma^1 \rangle, \langle \pi^2, \sigma^2 \rangle)$ is an inconsistent fork derivation path with $\pi^1 = [r_1^1, r_2^1, \ldots, r_{k_1}^1]$ and $\pi^2 = [r_2^2, r_2^2, \ldots, r_{k_2}^2]$. Then, according to Definition 3.5:

- $body(r_1^1)\theta(\pi^1)\sigma^1 = body(r_1^2)\theta(\pi^2)\sigma^2$ and $head(r_{k_1}^1)\theta(\pi^1)\sigma^1 = not\ head(r_{k_2}^2)\theta(\pi^2)\sigma^2$.

If $L = body(r_1^1)\theta(\pi^1)\sigma^1$ and $L' = head(r_{k_1}^1)\theta(\pi^1)\sigma^1$, by Lemma 3.2, this implies that L' and $not\ L'$ both belong to ξ_L. So, LR is not consistent, which completes the proof of the theorem. □

According to Theorem 3.1, testing the consistency of a set of l-rules reduces to testing whether there exists an inconsistent derivation path in LR. We refer to Appendix B for the actual computation of inconsistent derivation paths.

We end this section by noting that a *consistent* set of l-rules may still generate inconsistencies from a given partial interpretation. For example, if LR consists of the instantiated rules $q \leftarrow p$ and $not\ q \leftarrow not\ r$ then LR is consistent. However, for $I = \{p, not\ r\}$, we have $\xi_I = \{p, q, not\ q, not\ r\}$, which is not a partial interpretation. This point will be discussed shortly when dealing with database consistency.

4 Database Semantics

In this section, we present formal semantics for databases and, in the following section, we show how this semantics can be computed by usual Datalog programs.

Definition 4.1 - Database. *A* database *is a triple* $\Delta = (\mathcal{L}, UR, QR)$ *where:*

- *$\mathcal{L}$ is a set of ground literals (meant to be the set of literals stored in the database).*
- *UR is a consistent set of l-rules (meant to be the set of update rules).*
- *QR is a set of usual Datalog^{neg} rules (meant to be the set of query rules).* □

Given a database Δ, let ξ be the exception operator associated with the set UR. The least fixed point of ξ with respect to $\mathcal{L}$, denoted $\xi_{\mathcal{L}}$, will be referred to as the *exception set* of Δ, and the literals of $\xi_{\mathcal{L}}$ as the *exceptions* of Δ. It follows from Lemma 3.1 that:

$$(1)\ \ \mathcal{L} \subseteq \xi_{\mathcal{L}} \qquad \text{and} \qquad (2)\ \ \xi_{\mathcal{L}} = \left(\bigcup_{l \in \mathcal{L}} \xi_l\right).$$

The following definition says that a database Δ is consistent if the literals of $\mathcal{L}$ do not lead to inconsistent exceptions.

Definition 4.2 - Consistent Database. *A database Δ is said to be consistent if $\xi_{\mathcal{L}}$ is a partial interpretation.* $\Box$

As $\mathcal{L} \subseteq \xi_{\mathcal{L}}$, the above definition implies that: if Δ is a consistent database then $\mathcal{L}$ is a partial interpretation. This simply means that, in a consistent database, $\mathcal{L}$ cannot contain a fact f and its negation.

Running Example 4.1 Let us call $\Delta = (\mathcal{L}, UR, QR)$, the database of our running example after the insertion of $math(Bob)$ and $cs(Bob)$ (with "$\neg$" replaced by "*not*"). That is, we have:

- $\mathcal{L} = \{math(Bob), cs(Bob)\}$,
- UR :

$$\begin{array}{lrcl} UR_1: & not\ crazy(X) & \leftarrow & cs(X) \\ UR_2: & not\ cs(X) & \leftarrow & crazy(X) \\ UR_3: & sad(X) & \leftarrow & not\ crazy(X) \\ UR_4: & not\ happy(X) & \leftarrow & sad(X) \end{array}$$

- QR :

$$\begin{array}{lrcl} QR_1: & crazy(X) & \leftarrow & sci(X),\ not\ cs(X) \\ QR_2: & cs(X) & \leftarrow & math(X),\ not\ crazy(X) \\ QR_3: & happy(X) & \leftarrow & math(X),\ cs(X) \end{array}$$

The operator ξ works over the update rules, beginning with the partial interpretation $\mathcal{L}$:

$$\begin{array}{ll} \xi^0(\mathcal{L}) = & \mathcal{L} \\ \xi^1(\mathcal{L}) = & \{math(Bob), cs(Bob), not\ crazy(Bob)\} \\ \xi^2(\mathcal{L}) = & \{math(Bob), sad(Bob), cs(Bob), not\ crazy(Bob)\} \\ \xi^3(\mathcal{L}) = & \{math(Bob), sad(Bob), cs(Bob), not\ crazy(Bob), not\ happy(Bob)\} \end{array}$$

It is easy to see that $\xi^3(\mathcal{L})$ is the least fixed point of ξ. The set of positive literals of $\xi_{\mathcal{L}}$ is:

$pos(\xi_{\mathcal{L}}) = \{math(Bob), sad(Bob), cs(Bob)\}$

and the set of negative literals of $\xi_{\mathcal{L}}$ is:

$neg(\xi_{\mathcal{L}}) = \{crazy(Bob), happy(Bob)\}$.

Roughly speaking, as we know that Bob is mathematician and computer scientist (because this information is stored in the database), we can be sure that he is sad, he is not crazy and he is not happy (because these are consequences of the update rules). $\Box$

To explain how the semantics of our database is computed, first we consider an algorithm that illustrates this computation. Next we formalize the idea presented in the algorithm by defining some operators based on the well-founded semantics operators. We note that the exception set $\xi_{\mathcal{L}}$ is the starting point for the computation of database semantics with respect to the given query rules.

Algorithm 4.1 - Database Semantics.

INPUT: The exception set $\xi_{\mathcal{L}}$, computed during the most recent update
OUTPUT: A set of ground literals S, representing the semantics of the database
METHOD:
//* *new*, *old* and *inc* are sets of ground literals
begin
  $new := \xi_{\mathcal{L}}$;
  repeat
    $old := new$;

```
            new := apply_the_query_rules_to(old);
            inc := ∅;
            for each literal l ∈ new do
                    if not l ∈ ξ_L then add l to inc;
            new := new − inc;
      until new = old;
      S := new;
end.
```

Now, along the lines of [HLS94a], we define two operators, T^* and U^*, as follows:

$$T^*(I) = (T^{\in}(I) \setminus neg(\xi_{\mathcal{L}})) \cup pos(\xi_{\mathcal{L}}) \quad \text{and} \quad U^*(I) = (U(I) \setminus pos(\xi_{\mathcal{L}})) \cup neg(\xi_{\mathcal{L}}).$$

Here $T^{\in}$ and U are the well-founded operators associated with the positive literals in $\xi_{\mathcal{L}}$ and with the query rules of QR. It can be shown (see [HLS94b]) that T^* and U^* have the following property: the operator $T^* \cup not.U^*$ is monotonic, *i.e.* for all partial interpretations I and I' : *if* $I \subseteq I'$ *then* $(T^*(I) \cup not.U^*(I)) \subseteq (T^*(I') \cup not.U^*(I'))$.

It follows that the sequence defined by:

$$SEM^0 = \xi_{\mathcal{L}} \qquad \text{and} \qquad SEM^k = T^*(SEM^{k-1}) \cup not.U^*(SEM^{k-1}) \ \text{ for every integer } k > 0,$$

has a limit. This limit, will be referred to as the *semantics* of Δ, and will be denoted by $SEM(\Delta)$. We have the following theorem.

Theorem 4.1 *For every consistent database* $\Delta = (\mathcal{L}, UR, QR)$, $SEM(\Delta)$ *is a partial interpretation such that* $\mathcal{L} \subseteq \xi_{\mathcal{L}} \subseteq SEM(\Delta)$. $\square$

Proof: As in [GRS91], the proof is done by induction. Thus, first we prove that SEM^0 is a partial interpretation. In fact, $SEM^0 = \xi_{\mathcal{L}}$. As we consider Δ to be a consistent database and the update rules to be a consistent set of l-rules, we have that $\xi_{\mathcal{L}}$ is a partial interpretation. Now, for every $k > 0$ we assume that SEM^{k-1} is a partial interpretation.

We prove that SEM^k is also a partial interpretation by contradiction. That is, we suppose that SEM^k is not consistent. Thus, we assume that there is a literal l such that $l \in SEM^k$ and $not\ l \in SEM^k$. Therefore, we can say that $l \in T^*(SEM^{k-1})$ and $l \in U^*(SEM^{k-1})$. This implies that $l \in (T^{\in}(SEM^{k-1}) \setminus neg(\xi_{\mathcal{L}})) \cup pos(\xi_{\mathcal{L}})$ and $l \in (U(SEM^{k-1}) \setminus pos(\xi_{\mathcal{L}})) \cup neg(\xi_{\mathcal{L}})$. If we consider $l \in pos(\xi_{\mathcal{L}})$ then $l \notin U^*(SEM^{k-1})$, a contradiction. If we consider $l \in neg(\xi_{\mathcal{L}})$ then $l \notin T^*(SEM^{k-1})$, again a contradiction. Therefore, we consider the case where $l \in T^{\in}(SEM^{k-1})$ and $l \in U(SEM^{k-1})$.

By the definition of $T^{\in}(SEM^{k-1})$, there exists a rule $l \leftarrow L_1, \ldots, L_n$ in the instantiation of QR such that, for all $1 \leq i \leq n$, we have $L_i \in SEM^{k-1}$. On the other hand, by definition of $U(SEM^{k-1})$, one of the literals in the body of the rule, say L_{i_0}, is in one of the following situations:

1. $not\ L_{i_0} \in SEM^{k-1}$. As SEM^{k-1} is a partial interpretation, this is a contradiction because we cannot have L_{i_0} in both $pos(SEM^{k-1})$ and $neg(SEM^{k-1})$.
2. $L_{i_0} \in U(SEM^{k-1})$. Thus, L_{i_0} is a positive literal that belongs to $SEM^{k-1} \cap U(SEM^{k-1})$. In other words, $L_{i_0} \in pos(SEM^{k-1}) \cap neg(SEM^k)$. As $pos(SEM^{k-1}) \subseteq pos(SEM^k)$, then $L_{i_0} \in pos(SEM^k)$. We can apply to L_{i_0} the same reasoning we have used for l.

   That is, $L_{i_0} \in T^{\in}(SEM^{k-1})$, $L_{i_0} \in U(SEM^{k-1})$. $L_{i_0} \notin pos(\xi_{\mathcal{L}})$ and $L_{i_0} \notin neg(\xi_{\mathcal{L}})$. Following the previous ideas, there exists a literal L_{i_1} that belongs to $pos(SEM^{k-1}) \cap neg(SEM^k)$. But we also have that $L_{i_1} \in pos(SEM^{k-2})$ (because $L_{i_0} \in pos(SEM^{k-1})$. As a consequence, $pos(SEM^{k-2}) \cap neg(SEM^k) \neq \emptyset$. Similarly, for every j, $0 \leq j \leq k$, $pos(SEM^{k-j}) \cap neg(SEM^k) \neq \emptyset$. In particular, $pos(SEM^0) \cap neg(SEM^k) \neq \emptyset$. This is a contradiction because $pos(SEM^0) = pos(\xi_{\mathcal{L}})$ and because, following our reasoning, we are always considering a literal that is not in $pos(\xi_{\mathcal{L}})$.

Therefore, if $l \in T^{\in}(SEM^{k-1})$ then $l \notin U(SEM^{k-1})$ and vice-versa. In other words, if $l \in T^*(SEM^{k-1})$ then $l \notin U^*(SEM^{k-1})$ and vice-versa. This is a contradiction to the assumption that $l \in SEM^k$ and *not* $l \in SEM^k$. Thus, we proved that SEM^k is a partial interpretation.

From Lemma 3.1, we know that $\mathcal{L} \subseteq \xi_{\mathcal{L}}$. In [HLS94b] it is shown that $T^* \cup not.U^*$ is monotonic. Therefore, by the definition of SEM, we know that $\xi_{\mathcal{L}} \subseteq SEM^k$ for any $k > 0$ and thus $\xi_{\mathcal{L}} \subseteq SEM(\Delta)$. □

The above theorem shows that a consistent database Δ always has consistent semantics. Moreover, since $SEM(\Delta)$ can be written as $pos(SEM(\Delta)) \cup not.neg(SEM(\Delta))$, the underlying Herbrand base HB_{Δ} is partitioned into the following three sets: the set $pos(SEM(\Delta))$, the set $neg(SEM(\Delta))$ and the set $HB_{\Delta} \setminus (pos(SEM(\Delta)) \cup neg(SEM(\Delta)))$. The ground literals of these sets are called respectively *true, false* and *unknown* (with respect to Δ). It is important to note that the presence of unknown facts shows that our approach is *not* based on the so-called *Closed World Assumption* of [Rei78].

Moreover, if $\Delta = (\mathcal{L}, UR, QR)$ is a consistent database such that $\xi_{\mathcal{L}} = \mathcal{L}$, and if $\mathcal{L}$ contains no false fact, then, for every partial interpretation I, we have $T^*(I) = T^{\in}(I)$ and $U^*(I) = U(I)$. So, in this case, $SEM(\Delta)$ is obtained through least fixed point computations as in traditional approaches to Datalog^{neg} databases under well-founded semantics. This means that our approach extends traditional approaches, in which negative literals are not stored in the database and where update rules are not present. Moreover, our approach also extends that of [LPS93], where negative literals are stored in the database but where update rules are not present (in other words, our formalism reduces to that of [LPS93] if $\xi_{\mathcal{L}} = \mathcal{L}$).

Running Example 4.2 Let us compute the semantics $SEM(\Delta)$ of the database Δ of our running example. We first recall that

$$pos(\xi_{\mathcal{L}}) = \{math(Bob), sad(Bob), cs(Bob)\} \quad \text{and} \quad neg(\xi_{\mathcal{L}}) = \{crazy(Bob),\ happy(Bob)\}.$$

In *step* 0 we have:

$$SEM^0 = \quad \xi_{\mathcal{L}} = \quad \{math(Bob), sad(Bob), cs(Bob), not\ crazy(Bob), not\ happy(Bob)\}.$$

In *step* 1 we have:

$T^{\in}(SEM^0) = \{math(Bob), sad(Bob), cs(Bob), happy(Bob)\}$ and
$T^*(SEM^0) = (T^{\in}(SEM^0) \setminus neg(\xi_{\mathcal{L}})) \cup pos(\xi_{\mathcal{L}}) = \{math(Bob), sad(Bob), cs(Bob)\}$.

We compute $U(SEM^0)$ following the method of [BF91]. That is, we define $SPF(SEM^0)$ to be the limit of the sequence $[SPF_i]_{i \geq 0}$ defined by:

$$\begin{aligned} SPF_0 = &\quad \{head(r) \mid r \in inst\_QR \wedge pos(body(r)) = \emptyset \wedge \forall L \in body(r) : \neg L \notin SEM^0\} \\ SPF_i = &\quad \{head(r) \mid r \in inst\_QR \wedge pos(body(r)) \subseteq SPF_{i-1} \wedge \forall L \in body(r) : \neg L \notin SEM^0\} \end{aligned}$$

where $inst\_QR$ corresponds to the instantiation of query rules together with the positive literals of $\xi_{\mathcal{L}}$. The limit of this sequence corresponds, in fact, to the smallest set of potentially founded facts. SPF_0 is equal to the positive literals of $\xi_{\mathcal{L}}$. Actually:

$$\begin{aligned} SPF_0 = &\quad \{math(Bob), sad(Bob), cs(Bob)\} \\ SPF_1 = &\quad \{math(Bob), sad(Bob), cs(Bob), happy(Bob)\} \end{aligned}$$

Since $SPF_2 = SPF_1$, we have computed $SPF(SEM^0)$. As shown in [BF91], $U(SEM^0)$ is the complement of $SPF(SEM^0)$ with respect to the underlying Herbrand base. Therefore, considering that *Bob* is the only constant in our database, we have:

$$\begin{aligned} U(SEM^0) = &\quad \{sci(Bob), crazy(Bob)\}, \\ U^*(SEM^0) = &\quad (U(SEM^0) \setminus pos(\xi_{\mathcal{L}})) \cup neg(\xi_{\mathcal{L}}) \\ = &\quad U(SEM^0) \cup \{happy(Bob)\}. \end{aligned}$$

Thus:

$$SEM^1 = \{math(Bob), sad(Bob), cs(Bob), not\ sci(Bob), not\ crazy(Bob), not\ happy(Bob)\}.$$

As $SEM^2 = SEM^1$, the semantics of Δ is the partial interpretation SEM^1. We note that, in the above computation, *happy*(*Bob*) was derived as true by the query rule QR_3 (because $happy(Bob) \in T^{\in}(SEM^0)$), but also as a false exception (because $happy(Bob) \in neg(\xi_{\mathcal{L}})$). As we explained earlier, priority is given to derivations driven by update rules, thus *not happy*(*Bob*) is regarded as an exception to QR_3. □

5 Computing the Semantics Using Datalog

In this section we show that a consistent database Δ, over an alphabet $\mathbf{A}$, can be associated with a Datalog database Δ^+, such that $SEM(\Delta)$ is contained in the well-founded model of Δ^+. We recall from Section 2, that $\mathbf{A} = \text{CONST} \cup \text{VAR} \cup \text{PRED}$. Let us extend $\mathbf{A}$ to an alphabet $\mathbf{A}'$ as follows:

1. For every n-ary predicate p in PRED, define a *new* n-ary predicate *nonp* not in PRED.
2. Define $\text{PRED}' = \text{PRED} \cup \{nonp \mid p \text{ is a predicate of PRED}\}$.
3. Define $\mathbf{A}'$ to be the alphabet $\text{CONST} \cup \text{VAR} \cup \text{PRED}'$.

To every literal L, we associate a literal L^+ defined as follows:

$$L^+ = \begin{cases} p(\tilde{t}) & \text{if } L = p(\tilde{t}) \\ nonp(\tilde{t}) & \text{if } L = not\ p(\tilde{t}) \end{cases}$$

Now, given a consistent database $\Delta = (\mathcal{L}, UR, QR)$ we define $\mathcal{L}^+$, UR^+ and QR^+ as follows:

1. $\mathcal{L}^+ = \{L^+ \mid L \in \mathcal{L}\}$.
2. $UR^+ = \{L_0^+ \leftarrow L_1^+ \mid L_0 \leftarrow L_1 \in UR\}$.
3. QR^+ is the set of all rules $p(\tilde{t}) \leftarrow L_1, L_2, \ldots, L_k, not\ nonp(\tilde{t})$ where $p(\tilde{t}) \leftarrow L_1, L_2, \ldots, L_k$ is in QR.

Running Example 5.1 For the database defined in our running example, we have:

$$\begin{array}{lrcl}
UR_1^+: & noncrazy(X) & \leftarrow & cs(X) \\
UR_2^+: & noncs(X) & \leftarrow & crazy(X) \\
UR_3^+: & sad(X) & \leftarrow & noncrazy(X) \\
UR_4^+: & nonhappy(X) & \leftarrow & sad(X) \\
 & & & \\
QR_1^+: & crazy(X) & \leftarrow & sci(X),\ not\ cs(X),\ not\ noncrazy(X) \\
QR_2^+: & cs(X) & \leftarrow & math(X),\ not\ crazy(X),\ not\ noncs(X) \\
QR_3^+: & happy(X) & \leftarrow & math(X),\ cs(X),\ not\ nonhappy(X)
\end{array}$$

and $\mathcal{L}^+ = \{cs(Bob), math(Bob)\}$ □

We can remark that, if we consider $\mathcal{L}^+$ and UR^+ together, then we have a *positive* logic program in which we can apply the standard immediate consequence operator T [Bid91]. Based on the fact that applying ξ to $\mathcal{L}$ and UR corresponds to applying T to $\mathcal{L}^+$ and UR^+, we have the following proposition.

Proposition 5.1 *Let $\Delta = (\mathcal{L}, UR, QR)$ be a database. For every literal L, we have: $L \in \xi_{\mathcal{L}}$ if and only if $L^+ \in T_{\mathcal{L}^+}$, where $T_{\mathcal{L}^+}$ is the least fixed point of T with respect to $\mathcal{L}^+$ and UR^+.* □

Now, given a consistent database $\Delta = (\mathcal{L}, UR, QR)$ over the alphabet $\mathbf{A}$, we associate Δ with a Datalog database $\Delta^+ = (\mathcal{L}^+, UR^+, QR^+)$. In fact, Δ^+ corresponds to a logic program containing the rules of QR^+ and the facts of $T_{\mathcal{L}^+}$. We denote by $SEM^{wf}(\Delta^+)$ the well-founded model of Δ^+. Let us illustrate the relationship between Δ and Δ^+ using an example.

Running Example 5.2 The database Δ^+ of Running Example 5.1 corresponds to the following Datalog program:

$$
\begin{array}{rrcl}
1) & crazy(X) & \leftarrow & sci(X),\ not\ cs(X),\ not\ noncrazy(X) \\
2) & cs(X) & \leftarrow & math(X),\ not\ crazy(X),\ not\ noncs(X) \\
3) & happy(X) & \leftarrow & math(X),\ cs(X),\ not\ nonhappy(X) \\
4) & cs(Bob) & & \\
5) & math(Bob) & & \\
6) & sad(Bob) & & \\
7) & noncrazy(Bob) & & \\
8) & nonhappy(Bob) & &
\end{array}
$$

If we compute the well-founded semantics of this program, we find the following:

$$
\begin{aligned}
SEM^{wf}(\Delta^+) = \{\ & math(Bob), sad(Bob), cs(Bob), noncrazy(Bob), nonhappy(Bob), not\ sci(Bob), \\
& not\ crazy(Bob), not\ happy(Bob), not\ nonsci(Bob), not\ nonmath(Bob), \\
& not\ nonsad(Bob), not\ noncs(Bob)\}.
\end{aligned}
$$

That is,

$$
\begin{aligned}
SEM^{wf}(\Delta^+) = {} & SEM(\Delta)\ \cup\ \{noncrazy(Bob), nonhappy(Bob)\}\ \cup \\
& \{not\ nonsci(Bob), not\ nonmath(Bob), not\ nonsad(Bob), not\ noncs(Bob)\}
\end{aligned}
$$

where $SEM(\Delta)$ is the model of Δ calculated in Running Example 4.2. In other words, the ground literals in $SEM^{wf}(\Delta^+)$ over the predicates *sci*, *math*, *cs*, *crazy*, *sad* and *happy* constitute exactly the semantic $SEM(\Delta)$ of Δ. Obviously, $SEM^{wf}(\Delta^+)$ contains also literals that correspond to the new predicates which are in PRED$'$ but not in PRED. □

Theorem 5.1 *Let Δ be a consistent database and let Δ^+ be the associated Datalog database. Then, for every predicate p in* PRED*:*

- $p(\tilde{a}) \in SEM(\Delta)$ *iff* $p(\tilde{a}) \in SEM^{wf}(\Delta^+)$ *and*
- *not* $p(\tilde{a}) \in SEM(\Delta)$ *iff not* $p(\tilde{a}) \in SEM^{wf}(\Delta^+)$. □

Proof : See Appendix A. □

6 Update Semantics

In this section, we show that our update processing transforms, in a *deterministic* way, a consistent database into a new database which is again consistent, and in which the requested update has been effectively performed.

In order to define the way in which the set $\mathcal{L}$ is modified in our approach, we extend the notation *not.I* to partial interpretations. Given a partial interpretation $I = pos(I) \cup not.neg(I)$, we define *not.I* to be the partial interpretation $neg(I) \cup not.pos(I)$.

Updating a database means inserting or deleting a fact. In both cases, the update requires to store a ground literal l in $\mathcal{L}$, while leaving the database consistent. Recall that l is a positive literal in the case of insertion and l is a negative literal in the case of deletion.

As storing l in $\mathcal{L}$ may create inconsistencies, we eliminate them by removing from $\mathcal{L}$ all literals μ which generate exceptions contradicting the exceptions generated by l, *i.e.*, we remove all literals μ such that $(not.\xi_\mu) \cap \xi_l \neq \emptyset$. Hence the following definition, where $upd(l, \Delta)$ denotes the database Δ after the storage of a (positive or negative) literal l.

Definition 6.1 - Update of Δ. *The update of a database Δ by a literal l is a database $upd(l, \Delta) = (\mathcal{L}', UR, QR)$ defined by:*

$$\mathcal{L}' = (\mathcal{L} \setminus \mathit{lfp}(\vartheta, not.\xi_l)) \cup \{l\}. \qquad \square$$

Before we show that the above definition modifies the set $\mathcal{L}$ so that (1) the update is actually performed and (2) database consistency is preserved, let us illustrate updating using our running example.

Running Example 6.1 Consider the database Δ of Running Example 4.1, where $\mathcal{L} = \{math(Bob), cs(Bob)\}$. First we suppose the deletion of $sad(Bob)$ from Δ. From $l = not\ sad(Bob)$, we obtain $\xi_l = \{not\ sad(Bob)\})$. Clearly, $not.\xi_l = \{sad(Bob)\}$ and we find that $\mathit{lfp}(\vartheta, not.\xi_l) = \{not\ sad(Bob), cs(Bob), not\ crazy(Bob)\}$.
Thus the database $\Delta' = upd(not\ sad(Bob), \Delta)$ is defined by:
$\mathcal{L}' = \mathcal{L} \setminus \{sad(Bob), cs(Bob), not\ crazy(Bob)\} \cup \{not\ sad(Bob)\} = \{math(Bob), not\ sad(Bob)\}$.
Now, consider the insertion of $crazy(Bob)$ in Δ'. We have $l = crazy(Bob)$, $\xi_l = (\{crazy(Bob), not\ cs(Bob)\})$ and $\mathit{lfp}(\vartheta, not.\xi_l) = (\{crazy(Bob), cs(Bob)\})$. Therefore, nothing has to be removed from the database and the database $\Delta'' = upd(crazy(Bob), \Delta')$ is defined by:

$$\mathcal{L}'' = \{math(Bob),\ crazy(Bob), not\ sad(Bob)\}. \qquad \square$$

Proposition 6.1 *Given a consistent database Δ and a literal l, let $\Delta' = upd(l, \Delta)$. Then Δ' is consistent and $l \in SEM(\Delta')$.* $\square$

Proof: The fact that l is in $SEM(\Delta')$ follows from the fact that l is in $\mathcal{L}'$. The consistency of Δ' is shown by contradiction as follows: Assume that $\xi_{\mathcal{L}'}$ is inconsistent and let f be a fact such that f and $not\ f$ are in $\xi_{\mathcal{L}'}$. This implies that there are two literals l_1 and l_2 in $\mathcal{L}'$ such that $f \in \xi_{l_1}$ and $not\ f \in \xi_{l_2}$. We consider separately four cases as follows:

1. *If l_1 and l_2 are both positive,* then let $l_1 = f_1$ and $l_2 = f_2$, where f_1 and f_2 are two facts of $\mathcal{L}'$. Since UR is a consistent set of l-rules, ξ_l is consistent. Thus, we cannot have $l = f_1 = f_2$. Moreover, since Δ is assumed to be consistent, and since $\xi_{f_1} \cup \xi_{f_2}$ is not consistent, f_1 and f_2 cannot both belong to $\mathcal{L}$.

   Therefore, by Definition 6.1, f_1 and f_2 are distinct facts, one of them belongs to $\mathcal{L}$ while the other is being inserted. Thus, we assume that $l_2 \neq l_1 = f_1$, $f_1 \in \mathcal{L}$ and that $l = f_2$. In this case, $f \in \xi_{f_1}$ and $not\ f \in \xi_l$. Thus, $f \in \xi_{f_1}$ and $f \in not.\xi_l$. Therefore, $f_1 \in \vartheta_f$ and, as $f \in not.\xi_l$, f_1 belongs to $\mathit{lfp}(\vartheta, not.\xi_l)$. By Definition 6.1, this implies that $f_1 \notin \mathcal{L}'$, which is a contradiction to our previous hypothesis that $l_1 = f_1$ belongs to $\mathcal{L}'$. This shows that l_1 and l_2 cannot both be positive literals.

2. *If l_1 is positive and l_2 is negative,* then let $l_1 = f_1$ and $l_2 = not\ f_2$, where f_1 and $not\ f_2$ belongs to $\mathcal{L}'$. Since Δ is assumed to be consistent, and since $\xi_{l_1} \cup \xi_{l_2}$ is not consistent, f_1 and $not\ f_2$ cannot both belong to $\mathcal{L}$. Therefore, by Definition 6.1, we have two possibilities:

   (a) $l = l_1 = f_1$ and $not\ f_2 \in \mathcal{L}$.
   In this case, $f \in \xi_l$ and $not\ f \in \mathit{lfp}(\xi, not\ f_2)$. Thus, $not\ f \in not.\xi_l$ and $not\ f \in \mathit{lfp}(\xi, not\ f_2)$. Therefore, $not\ f_2 \in \mathit{lfp}(\vartheta, not\ f)$ and, as $not\ f \in not.\xi_l$, $not\ f_2$ belongs to $\mathit{lfp}(\vartheta, not.\xi_l)$. By Definition 6.1, this implies that $not\ f_2 \notin \mathcal{L}'$, which is a contradiction to our previous hypothesis that $l_2 = not\ f_2$ belongs to $\mathcal{L}'$.

(b) $l = l_2 = not\ f_2$ and $f_1 \in \mathcal{L}$.
In this case, $not\ f \in \xi_l$ and $f \in \xi_{f_1}$. Thus, $f \in not.\xi_l$ and $f_1 \in \vartheta_f$. Therefore, $f_1 \in lfp(\vartheta, not.\xi_l)$. By Definition 6.1, this implies that $f_1 \notin \mathcal{L}'$, which is a contradiction to our previous hypothesis that $l_1 = f_1$ belongs to $\mathcal{L}'$.

This shows that l_1 cannot be a positive literal while l_2 is a negative literal.

3. *If l_1 is negative and l_2 is positive,* then the proof is similar to 2 above, and we omit it here.

4. *If l_1 and l_2 are both negative,* then let $l_1 = not\ f_1$ and $l_2 = not\ f_2$, where $not\ f_1$ and $not\ f_2$ belongs to $\mathcal{L}'$. Since UR is a consistent set of l-rules, ξ_l is consistent. Thus, we cannot have $l = not\ f_1 = not\ f_2$. Moreover, since Δ is assumed to be consistent, and since $lfp(\xi, not\ f_1) \cup lfp(\xi, not\ f_2)$ is not consistent, $not\ f_1$ and $not\ f_2$ cannot both belong to $\mathcal{L}$.

   Therefore, by Definition 6.1, l_1 and l_2 are distinct literals, one of them belongs to $\mathcal{L}$ while the other is being deleted. Thus, we assume that $l_2 \neq l_1 = not\ f_1$, $not\ f_1 \in \mathcal{L}$ and that $l = not\ f_2$. In this case, $f \in lfp(\xi, not\ f_1)$ and $not\ f \in \xi_l$. Thus, $f \in lfp(\xi(not\ f_1))$ and $f \in not.\xi_l$. Therefore, $not\ f_1 \in \vartheta_f$ and, as $f \in not.\xi_l$, $not\ f_1$ belongs to $lfp(\vartheta, not.\xi_l)$. By Definition 6.1, this implies that $not\ f_1 \notin \mathcal{L}'$, which is a contradiction to our previous hypothesis that $l_1 = not\ f_1$ belongs to $\mathcal{L}'$. This is a contradiction which shows that l_1 and l_2 cannot both be negative literals. □

The above proposition implies that, starting with the empty database (which is consistent), our method of updating maintains database consistency during database operations.

To end this section, it is interesting to note that, according to Theorem 3.1, two policies can be followed in the presence of an inconsistent set of update rules:

1. The first policy is to reject the set of update rules. This is the solution adopted in this paper, since we assume the set of update rules to be consistent. According to this policy, it follows from Proposition 6.1 that *every* update is accepted and performed, while maintaining the database consistent.

2. The second policy, is to accept any set of update rule and test, at every update request, whether the update may cause inconsistencies. According to this policy, updating a database Δ by a literal l would be processed as follows:

   (a) Compute ξ_l.
   (b) **If** ξ_l is a partial interpretation **then** compute $upd(l, \Delta)$
   else reject the update.

   We note that, in order to improve efficiency of this test, we propose in Appendix B two algorithms which allow for some *a priori* computation concerning all updates to be potentially rejected. Moreover, we note that, according to this policy, and contrary to the previous one, *not all* updates are accepted and performed.

7 Related Works

In all semantics of logic programming that applies the Closed World Assumption [Rei78], an answer to a ground query is either *yes* or *no*. Therefore, in the Closed World Assumption (two-valued logic), if a ground literal cannot be derived as true, then it is assumed false. Based on this assumption, a *general logic program* [Llo87] is defined to be a set of rules of the form: $A_0 \leftarrow A_1, \ldots, A_m,\ not\ A_{m+1}, \ldots,\ not\ A_n$, where $n \geq m \geq 0$ and A_i is an atom.

Example 7.1 Consider the general logic program:

(1) p
(2) $q \leftarrow p$
(3) $t \leftarrow not\ s$

Below there are the answers that an implementation of this language will give to the following ground queries:

$$p:\ yes,\quad q:\ yes,\quad s:\ no,\quad t: yes \qquad \Box$$

However, if we consider the execution of a logic program, when a ground query is given to it, three things can happen: the answer is *yes,* the answer is *no* or *no answer* is given (corresponding to an infinite loop). In fact, the set of ground sentences is partitioned into three parts, namely the set of *true* sentences, the set of *false* sentences and the set of *unknown* sentences, that are neither true nor false. Due to this fact, some authors [Fit85] proposed three-valued logic semantics to represent the meaning of a logic program.

As a consequence, in the context of three-valued logic, the notion of *extended logic program* [GL90a, GL92, KS90] was introduced. In an extended logic program, the answer to a ground query can be: *yes* (ground literal derived as true), *no* (ground literal derived as false) and *unknown* (ground literal derived neither as true nor as false). To represent these three values, two types of negation are used: *not*, as negation by failure,[1] meaning that an information is not available (*i.e.,* unknown); and $\neg$, as classical negation, meaning that a negative information can be inferred from the rules (*i.e.,* explicitly). An extended logic program can be defined as a set of rules of the form $L_0 \leftarrow L_1, \ldots, L_m,\ not\ L_{m+1}, \ldots,\ not\ L_n$, where $n \geq m \geq 0$ and where L_i is a literal (a literal is a formula of the form A or $\neg A$ where A is an atom).

Example 7.2 Consider the extended logic program:

(1) p
(2) $q \leftarrow p$
(3) $t \leftarrow not\ s$
(4) $\neg r \leftarrow p$
(5) $u \leftarrow not\ \neg u$

In fact, rule number (3), for example, means that if there is no evidence of s then t is true. In the same way, rule (4) says that if p is true, then r is false and rule (5) says that if there is no evidence that $\neg u$ holds then u is true. Below there are the answers that an implementation of this language will give to the following ground queries:

$$p:\ yes,\quad q:\ yes,\quad s:\ unknown,\quad t:\ yes,\quad r:\ no,\quad u:\ yes \qquad \Box$$

In the context of extended logic programs, a relationship between our work and the approaches proposed in [GL90a] and [KS90] can be established. Before doing that, let us describe briefly the principal ideas of [GL90a] and [KS90].

Extended Logic Programs

The *semantics of extended logic programs* [GL90a], is based on the method of stable models [GL90b], and consists in defining when a set S of ground literals is an *answer set* of a program. Since a rule with variables is considered as a shorthand for a set of ground instances, it is sufficient to define answer sets for extended programs without variables. The definition is given in two steps: The first step considers programs without negation by failure and the second step considers the general

[1] Negation by failure rule [Llo87]: rule used to infer negative information. In fact, this rule is less powerful than the Closed World Assumption but, in practice, implementing anything beyond negation by failure is difficult.

case. In fact, these two steps constitute a procedure to verify if an answer set proposed by a rational agent is valid.

- STEP 1: Suppose that Π is an extended program without variables and without *not*, and that Lit is the set of ground literals in the language of Π. The *answer set* of Π, denoted by $\alpha(\Pi)$, is the smallest subset S of Lit such that:

  1. for any rule $L_0 \leftarrow L_1 \ldots L_m$ in Π, if $L_1 \ldots L_m \in S$ then $L_0 \in S$ and
  2. if S contains a pair of complementary literals, then $S = Lit$ (and Π is contradictory).

- STEP 2: Suppose that Π is an extended program without variables and Lit is the set of ground literals in the language of Π. For any set $S \subset Lit$, let Π^S be the extended program obtained from Π by deleting:

  1. each rule that has a formula *not* L in its body with $L \in S$ (if $L \in S$ and there is a rule with *not* L in its body, then this rule is not going to be applied - it can be deleted) and
  2. all formulas of the form *not* L in the body of the remaining rules (after (1) the remaining rules with *not* L in their bodies are those for which $L \notin S$, therefore *not* L is true (trivial) and can be eliminated from the body of the rules).

Clearly, Π^S does not contain *not*, so its answer set can be defined using STEP 1. If $\alpha(\Pi^S)$, the answer set of Π^S (found by STEP 1), is equal to S then S is the answer set of Π. Therefore, the answer sets of Π are characterized by the equation $S = \alpha(\Pi^S)$.

Example 7.3 Consider the following extended logic program Π:

$$
\begin{array}{lrcl}
(1) & crazy(X) & \leftarrow & sci(X),\ not\ \neg crazy(X) \\
(2) & cs(X) & \leftarrow & math(X),\ not\ \neg cs(X) \\
(3) & \neg crazy(X) & \leftarrow & cs(X) \\
(4) & \neg cs(X) & \leftarrow & crazy(X) \\
(5) & sci(Bob) & & \\
(6) & math(Bob) & &
\end{array}
$$

In this program, rule (1) is interpreted as follows: if $sci(X)$ is true and there is no evidence of $\neg crazy(X)$, then we can derive $crazy(X)$. The remaining rules are interpreted similarly.
The following two sets are answer sets for this program:

- $S_1 = \{sci(Bob), math(Bob), crazy(Bob), \neg cs(Bob)\}$.
  Following STEP 2, rule (2) is deleted and *not* $\neg crazy(X)$ is removed from rule (1).
- $S_2 = \{sci(Bob), math(Bob), \neg crazy(Bob), cs(Bob)\}$.
  Following STEP 2, rule (1) is deleted and *not* $\neg cs(X)$ is removed from rule (2). □

In [GL90a] the authors also show how an extended logic program can be transformed into a general logic program by renaming predicates (see also [She88]). This is done by (a) transforming every literal of the form $L = \neg p(\tilde{t})$ into the literal $L^+ = nonp(\tilde{t})$, and (b) transforming the extended logic program Π into the general logic program Π^+, obtained from Π by replacing every rule of the form: $L_0 \leftarrow L_1, \ldots, L_m,\ not\ L_{m+1}, \ldots,\ not\ L_n$ by the rule $L_0^+ \leftarrow L_1^+, \ldots, L_m^+,\ not\ L_{m+1}^+, \ldots,\ not\ L_n^+$. It has been shown [GL90a] that a consistent set S is an answer set of Π if and only if S^+ is an answer set of Π^+.

It has also been shown [GL90a] that the language of extended logic programs can be embedded into default logic and that the answer set semantics can be reformulated as a reduction of extended logic programs to other non-monotonic formalisms (such as autoepistemic logic, or introspective circumscription [GPP89, Kon88, Prz88]).

Exception Rules

A modification of the answer set semantics defined in [GL90a] is proposed in [KS90]. Two types of rules are used: *general rules* and *exception rules.* Both rules follow the syntax proposed by extended logic programs.

Example 7.4 Consider the following program Π:

GENERAL RULES

(1)	$crazy(X)$	$\leftarrow$	$sci(X)$
(2)	$cs(X)$	$\leftarrow$	$math(X)$
(3)	$sci(Bob)$		
(4)	$math(Bob)$		

EXCEPTION RULES

(1)	$\neg crazy(X)$	$\leftarrow$	$cs(X)$
(2)	$\neg cs(X)$	$\leftarrow$	$crazy(X)$ $\Box$

The difference between these two types of rules is that general rules have always positive heads while exception rules have always negative heads. Moreover, exception rules have higher priority than general rules. That is, if there is a contradiction between derivations by the two different groups of rules, then the derivation by exception rules overrides the one by general rules. In other words, negative information is preferred over positive information.

The modification with respect to [GL90a] is done by adding a third provision in STEP 2, thus redefining Π^S. The goal of this new provision is to delete any rule in Π having conclusion L with $\neg L \in S$. Therefore STEP 2 for [KS90] is:

- NEW STEP 2: For any set $S \subset Lit$, let ${}^S\Pi$ be the extended logic program obtained from Π by deleting:
  1. each rule that has a formula *not* L in its body with $L \in S$,
  2. all formulas of the form *not* L in the body of the remaining rules, with $L \notin S$ and
  3. every rule having a positive conclusion L, with $\neg L \in S$.

As before, we can use STEP 1 to find the answer set of ${}^S\Pi$. Therefore, for any variable-free program Π, S is defined as an *e-answer set* if and only if $S = \alpha({}^S\Pi)$.

Example 7.5 Program Π of Example 7.4 has two e-answer sets:

- $S_1 = \{sci(Bob), math(Bob), crazy(Bob), \neg cs(Bob)\}$.
  Following NEW STEP 2, general rule (2) is deleted.
- $S_2 = \{sci(Bob), math(Bob), \neg crazy(Bob), cs(Bob)\}$.
  Followig NEW STEP 2, general rule (1) is deleted. $\Box$

We note that, following [GL90a], an answer set may contain contradictions whereas, following [KS90], an e-answer set never contains contradictions (due to the priorities stated between exceptions and general rules). Moreover, a non-contradictory answer set of an extended logic program is also an e-answer set.

We also note that in [KS90], a transformation to eliminate exceptions is proposed. The initial program with two types of rules is transformed into an extended logic program using negation by failure as shown below.

Example 7.6 Program Π of Example 7.4 can be tranformed into the following program:

$$\begin{array}{lrcl}
(1) & crazy(X) & \leftarrow & sci(X),\ not\ \neg crazy(X) \\
(2) & cs(X) & \leftarrow & math(X),\ not\ \neg cs(X) \\
(3) & sci(Bob) & & \\
(4) & math(Bob) & & \\
(5) & \neg crazy(X) & \leftarrow & cs(X) \\
(6) & \neg cs(X) & \leftarrow & crazy(X) \quad \Box
\end{array}$$

Clearly, the program Π (with two types of rules) represents a more natural way of understanding default reasoning. The transformed program, however, is easier to implement: standard logic programming methods with negation by failure, such as SLDNF [Llo87], can be used. As a consequence of this transformation, the approach of [KS90] can be related to non-monotonic formalisms, as shown in [GL90a].

8 Comparison with Our Approach

The approach of [KS90], where two types of rules are used, is closely related to our approach as we also use two types of rules: query rules and update rules. However, the objectives are different and also the interaction between the two types of rules is different in the two approaches. The objective of [GL90a, KS90] is to study essentially queries, whereas our objetive is to study database queries *and* updates.

Regarding the interaction between the two types of rules there is an important difference, namely, the derivations by the two types of rules are interleaved in [KS90] whereas they are completely separated in our approach. More precisely, in [KS90] the intuition is that after deriving a positive literal using a general rule, one has to see if an exception rule can be applied, contradicting the result of the derivation (or that of a future derivation). For instance, in Example 7.4, $sci(Bob)$ is used in the rule $crazy(Bob) \leftarrow sci(Bob)$ to derive $crazy(Bob)$. Afterwards, an interaction with exception rules takes place and $crazy(Bob)$ is going to be used by the rule $\neg cs(Bob) \leftarrow crazy(Bob)$ meaning that, if Bob is crazy he cannot be computer scientist. In other words, the facts derived by general rules are used as "input" to derivations by exception rules.

In our approach, update rule derivations are completely independent from query rule derivations. The intuition is not the same as in [KS90]. For instance, if $crazy(Bob) \leftarrow sci(Bob)$ is a query rule, its consequence ($crazy(Bob)$) is not going to be used by an update rule to generate an exception. Update rules are activated only over the literals stored in the database (*i.e.,* the literals of $\mathcal{L}$) to compute $\xi_{\mathcal{L}}$. The literals of $\xi_{\mathcal{L}}$ are used as *input* to the derivations by query rules, and derivation by a query rule is accepted only if it is not in contradiction with a literal in $\xi_{\mathcal{L}}$. However, $\xi_{\mathcal{L}}$ does not change during query-driven derivations: $\xi_{\mathcal{L}}$ is recalculated *only* when an update is performed.

Example 8.1 Consider the database $\Delta = (\mathcal{L}, UR, QR)$:

UPDATE RULES

$$\begin{array}{lrcl}
(1) & \neg crazy(X) & \leftarrow & cs(X) \\
(2) & \neg cs(X) & \leftarrow & crazy(X)
\end{array}$$

QUERY RULES

$$\begin{array}{lrcl}
(1) & crazy(X) & \leftarrow & sci(X), not\ cs(X) \\
(2) & cs(X) & \leftarrow & math(X), not\ crazy(X)
\end{array}$$

and $\mathcal{L} = \{sci(Bob), math(Bob)\}$

Following our approach, the semantics of the database is given by $SEM(\Delta) = \{sci(Bob), math(Bob)\}$ which is the intersection of the two e-answer sets of Example 7.5.

Consider now the insertion of $crazy(Bob)$ in the programs of examples 7.3 and 7.4. If $crazy(Bob)$ is added to the general rules of Example 7.4, there will be no change in the semantics of that program. That is, both e-answer sets $S_1 = \{sci(Bob), math(Bob), crazy(Bob), \neg cs(Bob)\}$ and $S_2 = \{sci(Bob), math(Bob), \neg crazy(Bob), cs(Bob)\}$ will still be correct. However, if $crazy(Bob)$ is added in the extended logic program of Example 7.3, only the answer set $S1$ will be found.
In our approach, the goal is to privilege the literals (positive or negative) inserted explicitly by the users. Therefore, $sci(Bob)$, $math(Bob)$ and $crazy(Bob)$ will be true and $cs(Bob)$ will be false because: (i) $sci(Bob)$, $math(Bob)$ and $crazy(Bob)$ are inserted literals, and (ii) $crazy(Bob)$ and $cs(Bob)$ cannot be true at the same time. □

It is important to note that, in a certain sense, our approach is an extension of the one proposed in [KS90], since exceptions in our approach can have true or false heads. Therefore, in our approach, there is no priority between a negative and a positive fact. Instead, there is priority between derivations: update rule derivations have priority over query rule derivations. Another important difference is that, whereas in [KS90] a fact can activate just one exception rule, in our approach, we perform fixed point computation in order to find all facts derived by update rules. Thus, a fact in the database can activate an update rule to derive a new fact which, in turn, can activate another rule and so on, until reaching the least fixed point. This is in keeping with our philosophy of considering update rules as constraints, *all* consequences of which must hold in the database until a new update.

On the other hand, update rules in our approach are special rules having exactly one literal in the body and one in the head (we have called these rules literal rules). Therefore, they are less general than those proposed in [KS90]. The reason why we consider only literal rules is because we are interested only in modeling the elementary update operations (the insertion or deletion of a *single* fact) and their possible side effects. If one is interested only in the transitions from one database state to another, then literal rules seem to be quite adequate. However, if one is interested in stating "constraints" that any given database state must satisfy then, rules with more that one literal in the body are required. In the presence of such rules, updating becomes nondeterministic. Indeed, suppose that $q \leftarrow p, t$ is a constraint and that the database contains p and t. If we want to delete q then there are three possibilities: (i) delete p, (ii) delete t or (iii) delete p and t. Our objective in this paper is not how to choose one of these possibilities but, assuming that a choice has been made, how to perform the required update. However, a possible solution to this situation would be to create a hierarchy of exceptions. In this case, over the update rules there would exist new rules with higher priority, *i.e.,* capable of deriving exceptions to the exceptions, and so on.

As in [GL90a, KS90], we regard rules as *inference rules, i.e.,* we do not use classical implication. Therefore, in our semantics, instantiated rules such as $\neg sad(Bob) \leftarrow happy(Bob)$ and $\neg happy(Bob) \leftarrow sad(Bob)$ have different meanings.

9 Alternative Approaches

We end this paper with a few remarks on the alternatives offered by our approach. First, let us recall that our approach relies on two basic concepts:

1. Update rules of the form $L_0 \leftarrow L_1$, where L_0 and L_1 are literals.
2. Query rules as in general logic programs.

In order to implement these concepts we had to make certain choices with respect to the following questions:

1. Which kind of semantics to use for update rules.

2. Which kind of semantics to use for query rules.
3. Which kind of negation to use if the semantics for query rules is different than that for update rules.

The choices made in this paper were to use (in the framework of a three-valued logic):

1. Fixpoint semantics with classical negation for update rules.
2. Well-founded semantics for query rules.
3. Transform classical negation into negation of well-founded semantics (in the spirit of [AP92, GL90a]).

However, other choices are possible. For example, another choice would be:

1. To use query rules not as in general logic programs but as in extended logic programs. In this case, rule QR_1 of our running example would have to be rewritten as:
$$crazy(X) \leftarrow\ sci(X),\ \mathit{not}\ cs(X),\ \mathit{not}\ \neg crazy(X),$$
and interpreted as follows:

   - if someone is a scientist and there is no evidence that he is a computer scientist and there is no evidence that he is not crazy
     then he is crazy.

   The intended semantics of a query rule is that a fact can be derived only if there is no exception to this fact.

2. Instead of using well-founded semantics and transforming one type of negation into another, to use the method proposed in [GL90a]. In this case, given a consistent database $\Delta = (\mathcal{L}, UR, QR)$, we would associate Δ with an extended logic program $\Pi(\Delta)$ as follows:

   - $\xi_{\mathcal{L}}$ is first computed (with classical negation) and its literals are put in $\Pi(\Delta)$.
   - If $p(\widetilde{t}) \longleftarrow L_1, L_2, \ldots, L_k$ is in QR then the rule $p(\widetilde{t}) \longleftarrow L_1, L_2, \ldots, L_k, \mathit{not}\ \neg p(\widetilde{t})$ is in $\Pi(\Delta)$.

   NOTE: $\xi_{\mathcal{L}}$ is the minimal set of literals that are required to hold in the database (see Proposition 5.1). This is the reason why they should be considered in $\Pi(\Delta)$.

Running Example 9.1 Transforming the database Δ of Running Example 4.1, we have the following extended logic program $\Pi(\Delta)$.

1)	$crazy(X)$	$\leftarrow$	$sci(X),\ \mathit{not}\ cs(X),\ \mathit{not}\ \neg crazy(X)$
2)	$cs(X)$	$\leftarrow$	$math(X),\ \mathit{not}\ crazy(X),\ \mathit{not}\ \neg cs(X)$
3)	$happy(X)$	$\leftarrow$	$math(X),\ cs(X),\ \mathit{not}\ \neg happy(X)$
4)	$cs(Bob)$		
5)	$math(Bob)$		
6)	$sad(Bob)$		
7)	$\neg crazy(Bob)$		
8)	$\neg happy(Bob)$		

To find the semantics of $\Pi(\Delta)$ we can use the method proposed in [GL90a]. We find the following answer set:
$$S = \{math(Bob), sad(Bob), cs(Bob), \neg crazy(Bob), \neg happy(Bob)\}. \qquad \square$$

Since it is possible to translate Δ into $\Pi(\Delta)$, the results found in [GL90a] apply here, *i.e.,* our database model can be defined in terms of non-monotonic formalisms. Indeed, it is easy to see that query rules work as default rules with respect to update rules.

Moreover, applying the transformation used in [GL90a, She88] that renames classically negated predicates, QR_1 would be rewritten as

$$crazy(X) \leftarrow\ sci(X),\ not\ cs(X),\ not\ noncrazy(X).$$

Recall that, in Section 5, we obtained the same result when transforming QR into QR^+.

Indeed, using this transformation, we can obtain $\Pi^+(\Delta)$ from $\Pi(\Delta)$. It is clear that $\Pi^+(\Delta)$ is identical to the logic program associated to Δ^+ in Running Example 5.2.

Running Example 9.2 Consider the general logic program $\Pi^+(\Delta)$ obtained from $\Pi(\Delta)$ in our previous example (Running Example 9.1). Its answer set is:

$$S = \{math(Bob), sad(Bob), cs(Bob), noncrazy(Bob), nonhappy(Bob)\}.$$

REMARK: It is worth noting that the answer set of a general logic program $\Pi^+(\Delta)$ corresponds to its stable model [GL90a]. Stable models refer to two-valued logic. When speaking of total (or two-valued logic) interpretations it is more usual to represent a model only with positive literals (indeed, S contains only positive literals). Negative literals are the missing ones. However, to maintain consistency with the rest of the paper (where a three-valued logic is used), we will explicitly represent negative literals. Therefore, S will be rewritten as:

$$\begin{aligned} S = \{\ & math(Bob), sad(Bob), cs(Bob), noncrazy(Bob), nonhappy(Bob), not\ sci(Bob), \\ & not\ crazy(Bob), not\ happy(Bob), not\ nonsci(Bob), not\ nonmath(Bob), \\ & not\ nonsad(Bob), not\ noncs(Bob)\}. \qquad \square \end{aligned}$$

Now, let $\Delta = (\mathcal{L}, QR, UR)$ be a consistent database over an alphabet $\mathbf{A}$. Let $\Pi(\Delta)$ be the corresponding extended logic program and $\Pi^+(\Delta)$ be the general logic program obtained from $\Pi(\Delta)$.

(A) Considering the results in [GL90a], we know that $S(\Pi(\Delta))$ is a consistent answer set of the extended logic program $\Pi(\Delta)$ if and only if $S(\Pi^+(\Delta))$ is an answer set of the general logic program $\Pi^+(\Delta)$. Therefore, we write:

$$S(\Pi(\Delta)) \Longleftrightarrow S(\Pi^+(\Delta)).$$

This expression means that for every predicate p in PRED we have:

- $p(\tilde{a}) \in S(\Pi(\Delta))$ iff $p(\tilde{a}) \in S(\Pi^+(\Delta))$.
- $\neg p(\tilde{a}) \in S(\Pi(\Delta))$ iff $nonp(\tilde{a}) \in S(\Pi^+(\Delta))$.

Running Example 9.3 In the Running Example 9.1 we found that the answer set of program $\Pi(\Delta)$ was :

$$S(\Pi(\Delta)) = \{math(Bob), sad(Bob), cs(Bob), \neg crazy(Bob), \neg happy(Bob)\}.$$

This answer set corresponds to the answer set of $\Pi^+(\Delta)$ in the Running Example 9.2:

$$S(\Pi^+(\Delta)) = \{math(Bob), sad(Bob), cs(Bob), noncrazy(Bob), nonhappy(Bob)\}. \qquad \square$$

(B) Now let $SEM(\Delta)$ be the semantics of Δ in our approach and let $SEM^{wf}(\Pi^+(\Delta))$ (identical to $SEM^{wf}(\Delta^+)$) be the well-founded model of $\Pi^+(\Delta)$. By Theorem 5.1, we can write:

$$SEM(\Delta) \Longleftrightarrow SEM^{wf}(\Pi^+(\Delta)).$$

This expression means that, for every predicate p in PRED, we have:

- $p(\tilde{a}) \in SEM(\Delta)$ iff $p(\tilde{a}) \in SEM^{wf}(\Pi^{+}(\Delta))$ and
- *not* $p(\tilde{a}) \in SEM(\Delta)$ iff *not* $p(\tilde{a}) \in SEM^{wf}(\Pi^{+}(\Delta))$.

Indeed, in Section 5, Running Example 5.2 illustrates this situation.

(C) Now, considering the results presented in [GRS91] we know that:

- If $\Pi^{+}(\Delta)$ has a well-founded total model, then that model is its unique stable model.
- The well-founded partial model of a general logic program $\Pi^{+}(\Delta)$ is a subset of every stable model of $\Pi^{+}(\Delta)$.

It is worth noting that $S(\Pi^{+}(\Delta))$ is a stable model of $\Pi^{+}(\Delta)$. Moreover, as we explained in the REMARK of Running Example 9.2, stable models refer to two-valued logic and thus they are usually represented as a set containing only positive literals. Negative literals are the missing ones. However, in this paper, where we deal with three-valued models, we decide to "complete" stable models by representing explicitly negative literals.

- Denote by $S_c(\Pi^{+}(\Delta))$ the completed stable model of $\Pi^{+}(\Delta)$.

Thus we write

$$SEM^{wf}(\Pi^{+}(\Delta)) \subseteq S_c(\Pi^{+}(\Delta)).$$

Moreover, if $\mathcal{S}_c(\Pi^{+}(\Delta))$ is the set of all completed stable models of $\Pi^{+}(\Delta)$ then we can write:

$$SEM^{wf}(\Pi^{+}(\Delta)) \subseteq \bigcap(\mathcal{S}_c(\Pi^{+}(\Delta))).$$

Running Example 9.4 In the Running Example 5.2, the well-founded semantics of $\Pi^{+}(\Delta)$ (Datalog program that corresponds to Δ^{+}) is the total model.

$$\begin{aligned} SEM^{wf} = \{\ & math(Bob), sad(Bob), cs(Bob), noncrazy(Bob), nonhappy(Bob), not\ sci(Bob),\\ & not\ crazy(Bob), not\ happy(Bob), not\ nonsci(Bob), not\ nonmath(Bob),\\ & not\ nonsad(Bob), not\ noncs(Bob)\}. \end{aligned}$$

In Running Example 9.2, we have the same result for the completed stable model of $\Pi^{+}(\Delta)$. □

All the above expressions can be shown in the following diagram:

$$\begin{array}{ccc} SEM(\Delta) & & S(\Pi(\Delta)) \\ & & \Updownarrow \\ \Updownarrow & & S(\Pi^{+}(\Delta)) \\ & & \uparrow\downarrow \\ SEM^{wf}(\Pi^{+}(\Delta)) & \subseteq & S_c(\Pi^{+}(\Delta)) \end{array}$$

Indeed, looking at the comparison between well-founded semantics and stable models semantics, considering that our approach is an extension of the well-founded model and that the approach of [GL90a] is an extension of stable models, we might imagine that $SEM(\Delta) \subseteq S(\Pi(\Delta))$, *i.e.,* $SEM(\Delta) \subseteq \bigcap(\mathcal{S}(\Pi(\Delta)))$. However, *this is not always the case.* In fact, the situation can be illustrated by our Running Example.

Running Example 9.5 In Running Example 4.2, following our approach, we found the following semantics for the database Δ :

$SEM(\Delta) = \{math(Bob), sad(Bob), cs(Bob), not\ sci(Bob), not\ crazy(Bob), not\ happy(Bob)\}$.

On the other hand, in Running Example 9.1 following the approach proposed in [GL90a], we found that the semantics of the program $\Pi(\Delta)$, which is the logic program that corresponds to the database Δ, was:

$$S(\Pi(\Delta)) = \{math(Bob), sad(Bob), cs(Bob), \neg crazy(Bob), \neg happy(Bob)\}.$$

In both models $math(Bob)$, $sad(Bob)$ and $cs(Bob)$ are true and $crazy(Bob)$ and $happy(Bob)$ are false. We note however that $sci(Bob)$ is false in $SEM(\Delta)$ while it is *unknown* with respect to $S(\Pi(\Delta))$. $\square$

The following theorem shows that, sometimes, an atom can be false in the well-founded semantics and unknown in the answer sets semantics of [GL90a].

Theorem 9.1 *Let* $\Delta = (\mathcal{L}, QR, UR)$ *be a consistent database over an alphabet* $\mathbf{A}$. *Let* $\Pi(\Delta)$ *be the corresponding extended logic program. If* $SEM(\Delta)$ *is the semantics of* Δ *in our approach and* $\mathcal{S}(\Pi(\Delta))$ *is the set of all consistent answer sets of* $\Pi(\Delta)$ *then*

- $pos(SEM(\Delta)) \subseteq pos(\bigcap(\mathcal{S}(\Pi(\Delta))))$
- $neg(SEM(\Delta)) \not\subseteq neg(\bigcap(\mathcal{S}(\Pi(\Delta))))$ $\square$

Proof : We consider that p is a predicate in PRED and that $p(\tilde{a}) \in SEM(\Delta)$. To prove $pos(SEM(\Delta)) \subseteq pos(\bigcap(\mathcal{S}(\Pi(\Delta))))$, we have to show that, for every answer set $S(\Pi(\Delta))$, $p(\tilde{a}) \in S(\Pi(\Delta)$. Without loss of generality we can consider an answer set $S(\Pi(\Delta))$ of $\Pi(\Delta)$. Let $\Pi^+(\Delta)$ be the general logic program obtained from $\Pi(\Delta)$ and let $SEM^{wf}(\Pi^+(\Delta))$ be the well-founded model of $\Pi^+(\Delta)$.

If $p(\tilde{a}) \in SEM(\Delta)$, then, by Theorem 5.1, $p(\tilde{a}) \in SEM^{wf}(\Pi^+(\Delta))$ (see also item **(B)** and the diagram). If $p(\tilde{a}) \in SEM^{wf}(\Pi^+(\Delta))$ then $p(\tilde{a})$ belongs to every completed stable model of $\Pi^+(\Delta)$ [GRS91]. In particular, $p(\tilde{a}) \in S_c(\Pi^+(\Delta))$, *i.e.,* $p(\tilde{a}) \in S(\Pi^+(\Delta))$ since $p(\tilde{a})$ is a positive literal. If $p(\tilde{a}) \in S(\Pi^+(\Delta))$ then, according to [GL90a], $p(\tilde{a}) \in S(\Pi(\Delta))$ (see item **(A)**). Every answer set of a general logic program $\Pi^+(\Delta)$ corresponds to an answer set of an extended logic program $\Pi(\Delta)$. Thus, we proved that if $p(\tilde{a}) \in SEM(\Delta)$, then $p(\tilde{a}) \in S(\Pi(\Delta))$, for every answer set $S(\Pi(\Delta))$. Therefore, we proved that $pos(SEM(\Delta)) \subseteq pos(\bigcap(\mathcal{S}(\Pi(\Delta))))$.

We now consider that $not\ p(\tilde{a}) \in SEM(\Delta)$. To prove $neg(SEM(\Delta)) \not\subseteq neg(\bigcap(\mathcal{S}(\Pi(\Delta))))$ we have to show that there are some situations where $not\ p(\tilde{a}) \not\in neg(\bigcap(\mathcal{S}(\Pi(\Delta))))$. In fact, Running Example 9.5 is the proof that this situation can happen. $\square$

It would be interesting now to discuss briefly the reasons why the inclusion of $neg(SEM(\Delta))$ in $neg(\bigcap(\mathcal{S}(\Pi(\Delta))))$ does not hold. Indeed, if $not\ p(\tilde{a})$ belongs to $SEM(\Delta)$ then we have:

- $not\ p(\tilde{a}) \in SEM^{wf}(\Pi^+(\Delta))$ and
- $not\ p(\tilde{a}) \in S_c(\Pi^+(\Delta))$, for every completed stable model $S_c(\Pi^+(\Delta))$.

On the other hand, if $not\ p(\tilde{a})$ is in $S_c(\Pi^+(\Delta))$ then, $p(\tilde{a})$ is in no answer set of $\Pi(\Delta)$, and, moreover, one the following two cases occur:

1. For every answer set $S(\Pi(\Delta))$ of $\Pi(\Delta)$, $\neg p(\tilde{a}) \in S(\Pi(\Delta))$.
2. There exists an answer set $S(\Pi(\Delta))$ of $\Pi(\Delta)$ such that $p(\tilde{a})$ in unknown in $S(\Pi(\Delta))$ (*i.e.,* neither $p(\tilde{a})$ nor $\neg\ p(\tilde{a})$ is in $S(\Pi(\Delta))$).

We now investigate further each of these two cases:

1. Assume that, for every answer set $S(\Pi(\Delta))$ of $\Pi(\Delta)$, we have $\neg\, p(\tilde{a}) \in S(\Pi(\Delta))$.

   Then, given an answer set $S(\Pi(\Delta))$ of $\Pi(\Delta)$, we have by item **(A)**, $nonp(\tilde{a}) \in S(\Pi^+(\Delta))$. As $S(\Pi(\Delta))$ is consistent, we have $p(\tilde{a}) \notin S(\Pi(\Delta))$ and $p(\tilde{a}) \notin S(\Pi^+(\Delta))$ [GL90a]. It is obvious that if $p(\tilde{a}) \notin S(\Pi^+(\Delta))$ then, when completing the stable model, we have *not* $p(\tilde{a}) \in S_c(\Pi^+(\Delta))$. In other words, this situation shows that every negative literal (classical negation) in an answer set $S(\Pi(\Delta))$ is also a negative literal in the corresponding $S_c(\Pi^+(\Delta))$.

   Moreover, as $\neg p(\tilde{a})$ is in $\bigcap(\mathcal{S}(\Pi(\Delta)))$, $nonp(\tilde{a})$ belongs to $\bigcap(\mathcal{S}(\Pi^+(\Delta)))$ and *not* $p(\tilde{a})$ belongs to $\bigcap(\mathcal{S}_c(\Pi^+(\Delta)))$. In this case, it is true that $neg(SEM(\Delta)) \subseteq neg(\bigcap(\mathcal{S}(\Pi(\Delta))))$ since *not* $p(\tilde{a}) \in SEM(\Delta)$ corresponds to $\neg p(\tilde{a}) \in neg(\bigcap(\mathcal{S}(\Pi(\Delta))))$.

2. Assume that there exists an answer set $S(\Pi(\Delta))$ of $\Pi(\Delta)$ in which $p(\tilde{a})$ is unknown.

   Then $p(\tilde{a}) \notin S(\Pi(\Delta))$ and $\neg p(\tilde{a}) \notin S(\Pi(\Delta))$. Clearly, $p(\tilde{a}) \notin S(\Pi^+(\Delta))$, for the corresponding stable model of $\Pi^+(\Delta)$ (see item **(A)**). Therefore, when completing the stable model we have *not* $p(\tilde{a}) \in S_c(\Pi^+\ (\Delta))$. In other words, this situation shows that every unknown literal in an answer set $S(\Pi(\Delta))$ is a negative literal in the corresponding $S_c(\Pi^+(\Delta))$. Moreover, in this case, $\neg p(\tilde{a}) \notin \bigcap(\mathcal{S}(\Pi(\Delta))$ and $p(\tilde{a}) \notin \bigcap(\mathcal{S}(\Pi(\Delta))$. Thus, *not* $p(\tilde{a}) \in \bigcap(\mathcal{S}_c(\Pi^+(\Delta)))$, and so, it is *not* true that $neg(SEM(\Delta)) \subseteq neg(\bigcap(\mathcal{S}(\Pi(\Delta))))$, since *not* $p(\tilde{a}) \in SEM(\Delta)$ corresponds to $p(\tilde{a}) \notin \bigcap(\mathcal{S}(\Pi(\Delta)))$ and $\neg p(\tilde{a}) \notin \bigcap(\mathcal{S}(\Pi(\Delta)))$. Therefore, $neg(SEM(\Delta)) \not\subseteq neg(\bigcap(\mathcal{S}(\Pi(\Delta))))$.

10 Concluding Remarks

We have seen a model for Datalog^{neg} databases which contain query rules as well as update rules. The essential difference between these two types of rules lies in the fact that update rules have higher priority than query rules. Facts generated by update rules preserve their status (true or false) in the updated database, thus acting as exceptions to derivations by query rules.

An important contribution of our approach is that it provides a *deterministic* way to insert or delete any fact in a database. In this respect, update rules work as constraints which trigger the computation of *side effects* that are necessary in order to maintain automatically the database consistent.

Two different types of negation appear in our approach. Thus, to compute a semantics for our database, we were led to transform one type of negation into the other. We have seen, however, that another choice is possible, namely, to use the answer set semantics proposed in [GL90a], where extended logic programs are considered.

We have shown that we can transform our database Δ into an extended logic program $\Pi(\Delta)$ or into a Datalog program $\Pi^+(\Delta)$. An important point that can be mentioned concerning the translation of Δ into $\Pi(\Delta)$ is that our database model can be defined in terms of non-monotonic formalisms, since the results found in [GL90a] can be applied here.

One aspect of our approach that is not dealt with in this paper is time and space complexity. In this respect, we believe that an incremental approach to the computation of exceptions (see [HLS95a]) and of derived true/false facts can contribute significantly to improve performance.

Finally, we are currently investigating the relationship between our approach and that of [MT94, MT95] where changes on databases are specified in a declarative way, using *revision programs.* It turns out that, translating updates and update rules in terms of revision programming allows to show that our approach actually enjoys the property of minimal change. On the other hand, it can be shown that this translation allows to extend the class of revision programs having nice computational properties, as defined in [MT95]. We refer to [HLSS95] for more details on this topic.

References

[AP92] J. J. Alferes and L. M. Pereira. On logic of program semantics with two kinds of negation. In *Proceedings of the Joint International Conference and Symposium of Logic Programming*, 1992.

[BF91] N. Bidoit and Ch. Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78(1), 1991.

[Bid91] N. Bidoit. Negation in rule-based database languages: a survey. *Theoretical Computer Science*, 78(1), 1991.

[CGT90] S. Ceri, G.Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, 1990.

[Fit85] M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 4, 1985.

[GL90a] M. Gelfond and V. Lifschitz. Logic programming with classical negation. In *Proceedings of the Seventh International Conference of Logic Programming*, 1990.

[GL90b] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium of Logic Programming*, 1990.

[GL92] M. Gelfond and V. Lifschitz. Representing actions in extended logic programming. In *Proceedings of the Joint International Conference and Symposium of Logic Programming*, 1992.

[GPP89] M. Gelfond, H. Przymusinka, and T. Przymusinki. On the relationship between circumscription and negation as failure. *Artificial Inteligence*, 38(1), 1989.

[GRS91] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3), 1991.

[HLS94a] M. Halfeld Ferrari Alves, D. Laurent, and N. Spyratos. Passive and active rules in deductive databases. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, number 841 in LNCS - Lecture Notes in Computer Science. Springer-Verlag, 1994.

[HLS94b] M. Halfeld Ferrari Alves, D. Laurent, and N. Spyratos. Update driven rules in Datalog^{neg} databases. Technical Report 94-06, LIFO, Université d'Orléans, 1994.

[HLS95a] M. Halfeld Ferrari Alves, D. Laurent, and N. Spyratos. Passive and active rules in deductive databases. Technical Report 968, LRI - Université de Paris-Sud, 1995.

[HLS95b] M. Halfeld Ferrari Alves, D. Laurent, and N. Spyratos. Update rules in Datalog programs. Technical Report LRI, Université de Paris-Sud, 1995.

[HLSS95] M. Halfeld Ferrari Alves, D. Laurent, D. Stamate, and N. Spyratos. Update rules and revision programs, 1995. LIFO, Université d'Orléans.

[Kon88] K. Konolige. On the relation between default and autoepistemisc logic. *Artificial Inteligence*, 35, 1988. (Errata, volume 41, 1989/90).

[KS90] R. A. Kowalski and F. Sadri. Logic programs with exceptions. In *Proceedings of the Seventh International Conference of Logic Programming*, 1990.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, second extended edition, 1987.

[LPS93] D. Laurent, V. Phan Luong, and N. Spyratos. Updating intensional predicates in deductive databases. In *9th IEEE ICDE*, Vienna (Austria), 1993.

[MT94] V. W. Marek and M. Truszczyński. Revision specifications by means of revision programs. In *JELIA '94*, LNAI - Lecture Notes in Artificial Intelligence. Springer-Verlag, 1994.

[MT95] V. W. Marek and M. Truszczyński. Revision programming, database updates and integrity constraints. In *International Conference on Database Theory, ICDT*, number 893 in LNCS - Lecture Notes in Computer Science. Springer-Verlag, 1995.

[Prz88] T. C. Przymusinki. On the relationship between logic programming and non-montonic reasoning. In *The seventh National Conference of Artificial Intelligence*, Saint Paul, Minnesota, 1988.

[Rei78] R. Reiter. On closed word databases. In H. Gallaire and J. Minker, editors, *Logic and data bases.* Plenum Press, New York, 1978.

[She88] J. C. Shepherdson. Negation in logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, 1988.

[Ull89] J. D. Ullman. *Principles of Databases and Knowledge Base Systems*, volume I and II. Computer Science Press, 1989.

Appendix A - Proof of Theorem 5.1

Theorem 5.1 *Let Δ be a consistent database and let Δ^+ be the associated Datalog database. Then, for every predicate p in* PRED*:*

- $p(\tilde{a}) \in SEM(\Delta)$ *iff* $p(\tilde{a}) \in SEM^{wf}(\Delta^+)$ *and*
- *not* $p(\tilde{a}) \in SEM(\Delta)$ *iff not* $p(\tilde{a}) \in SEM^{wf}(\Delta^+)$. □

Proof: Let us call $SEM'(\Delta) = T' \cup not.U'$ the set of those literals in $SEM^{wf}(\Pi^+(\Delta)) = T^{wf} \cup not.U^{wf}$ that involve a predicate in PRED. Therefore, we have to show that $SEM(\Delta)$ and $SEM'(\Delta)$ are equal. For this, recall that $SEM(\Delta)$ and $SEM^{wf}(\Pi^+(\Delta))$ are inductively computed through the sequences $[SEM_i]_{i\geq 0}$ and $[SEM_i^{wf}]_{i\geq 0}$, respectively. Finally, for every i, we denote by SEM'_i the set of those literals in $SEM_i^{wf}(\Pi^+(\Delta))$ that involve a predicate in PRED. We prove successively the inclusions (1) $SEM'(\Delta) \subseteq SEM(\Delta)$ and (2) $SEM(\Delta) \subseteq SEM'(\Delta)$.

(1) We prove by induction that $SEM'(\Delta) \subseteq SEM(\Delta)$.

1. We first have $SEM'_0 \subseteq SEM_0$, since $SEM'_0 = \emptyset$ and $SEM_0 = \xi_{\mathcal{L}}$.

2. We assume now that $SEM'_{i-1} \subseteq SEM_{i-1}$, and we want to prove that $SEM'_i \subseteq SEM_i$. Recalling that $SEM'_i = T'_i \cup not.U'_i$ and $SEM_i = T_i \cup not.U_i$, the proof is done by contradiction, *i.e.,* we assume that there is a literal such that $l \in SEM'_i$ ($l \notin SEM'_{i-1}$) and $l \notin SEM_i$.

   Firstly, we consider that l is a **positive literal** ($l = p(\tilde{a})$), therefore, l is a consequence of a rule in $\Pi^+(\Delta)$. This means that $p(\tilde{a})$ is in SEM'_i because there is a rule r' in $\Pi^+(\Delta)$ such that:

   $$p(\tilde{a}) \longleftarrow L_1, L_2, \ldots, L_m, not\ nonp(\tilde{a}).$$

   On the other hand, by construction of $\Pi^+(\Delta)$, we know that rule r' corresponds to a query-driven rule r in Δ such that:

   $$p(\tilde{a}) \longleftarrow L_1, L_2, \ldots, L_m.$$

   As $p(\tilde{a})$ is in SEM'_{i-1}, then all literals in the body of r' are in $SEM_i^{wf}(\Pi^+(\Delta))$. Thus, all literals in the body of r' over predicate $PRED$ (*i.e.,* literals $L_1, L_2, \ldots, L_m$) are in SEM'_{i-1}. By induction hypothesis, $L_1, L_2, \ldots, L_m$ are also in SEM_{i-1} and, using rule r, it would be possible to obtain $p(\tilde{a}) \in SEM_i$. The only way to avoid this, *i.e.,* the only way to have $p(\tilde{a}) \notin SEM_i$, is to consider $p(\tilde{a}) \in neg(\xi_{\mathcal{L}})$. However, in this case, by construction of $\Pi^+(\Delta)$, the literal $nonp(\tilde{a})$ is in SEM'_{i-1}. Obviously, in this situation, rule r' cannot be used to derive $p(\tilde{a})$. That is, $p(\tilde{a}) \notin SEM'_i$ which is a contradiction to the initial assumption.

   Briefly, if $p(\tilde{a}) \in neg(\xi_{\mathcal{L}})$ then $p(\tilde{a}) \notin SEM'_i$ and $p(\tilde{a}) \notin SEM_i$. However, if $p(\tilde{a}) \notin neg(\xi_{\mathcal{L}})$ and $p(\tilde{a}) \in SEM'_i$ then $p(\tilde{a}) \in SEM_i$.

   Secondly, we consider that l is a **negative literal** ($l = not\ p(\tilde{a})$). If *not* $p(\tilde{a})$ is in SEM'_i then $p(\tilde{a})$ is not a potentially founded fact[2]. Therefore, $p(\tilde{a})$ belongs to $SPF'_{\Pi^+(\Delta)}(SEM_{i-1}^{wf})$ and, as a consequence, $p(\tilde{a}) \in U'_i$. On the other hand, if *not* $p(\tilde{a})$ is not in SEM_i then $p(\tilde{a})$ is a potentially founded fact, *i.e.,* $p(\tilde{a}) \in SPF_\Delta(SEM_{i-1})$ and, as a consequence, $p(\tilde{a}) \notin U_i$.

[2] We call $SPF'_{\Pi^+(\Delta)}(SEM_{i-1}^{wf})$ the set of those facts in $SPF^{wf}_{\Pi^+(\Delta)}(SEM_{i-1}^{wf})$ that involve predicates in $PRED$.

To prove that such a situation is *not* possible we will show that $U'_i \subseteq U_i$.

We remember that the sets of potentially founded facts for SEM_i and SEM_i^{wf} are computed through the sequences $[SPF_k(SEM_{i-1})]_{k\geq 0}$ and $[SPF_k(SEM_{i-1}^{wf})]_{k\geq 0}$, respectively. Therefore, if HB_Δ is the Herbrand Base with respect to the database Δ (only with predicates in $PRED$), then we have:

$$\begin{array}{rcl} U_i & = & [(HB_\Delta \setminus SPF_\Delta(SEM_{i-1})) \setminus pos(\xi_\mathcal{L})] \cup neg(\xi_\mathcal{L}) \\ & = & [HB_\Delta \setminus (SPF_\Delta(SEM_{i-1}) \cup pos(\xi_\mathcal{L}))] \cup neg(\xi_\mathcal{L}) \\ U'_i & = & HB_\Delta \setminus SPF'_{\Pi^+(\Delta)}(SEM_{i-1}^{wf}) \end{array}$$

In other words, we want to prove that

$$[SPF_\Delta(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L}) \subseteq SPF'_{\Pi^+(\Delta)}(SEM_{i-1}^{wf}).$$

The inclusion is proved by induction on k.

- We consider $k = 0$, $p(\tilde{a}) \notin SPF'_0(SEM_{i-1}^{wf})$ and $p(\tilde{a}) \in [SPF_0(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$. Obviously, since $p(\tilde{a}) \notin SPF'_0(SEM_{i-1}^{wf})$, by construction, $p(\tilde{a}) \notin pos(\xi_\mathcal{L})$.

  Therefore, there are rules r, in Δ, and r', in $\Pi^+(\Delta)$, such that:

  $$\begin{array}{rlcl} r: & p(\tilde{a}) & \longleftarrow & L_1, L_2, \ldots, L_m \\ r': & p(\tilde{a}) & \longleftarrow & L_1, L_2, \ldots, L_m, not\ nonp(\tilde{a}) \end{array}$$

  where L_i $(1 \leq i \leq m)$ are negative literals, with $(not\ L_i) \notin SEM_{i-1}$. Since $SEM'_{i-1} \subseteq SEM_{i-1}$, we know that $(not\ L_i) \notin SEM'_{i-1}$.

  Thus, the only way to obtain $p(\tilde{a}) \notin SPF'_0(SEM_{i-1}^{wf})$ is to have $nonp(\tilde{a}) \in SEM_{i-1}^{wf}$. However, this is possible only when $p(\tilde{a}) \in neg(\xi_\mathcal{L})$ due to the fact that *nonp* is an extensional predicate (by construction, only if $p(\tilde{a}) \in neg(\xi_\mathcal{L})$ we have $nonp(\tilde{a}) \in SEM_{i-1}^{wf}$).

  But, if $p(\tilde{a}) \in neg(\xi_\mathcal{L})$ then $p(\tilde{a}) \notin [SPF_0(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$. A contradiction! Thus, we proved that $[SPF_0(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L}) \subseteq SPF'_0(SEM_{i-1}^{wf})$.

- We suppose now that $[SPF_{k-1}(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L}) \subseteq SPF'_{k-1}(SEM_{i-1}^{wf})$ and we want to prove that $[SPF_k(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L}) \subseteq SPF'_k(SEM_{i-1}^{wf})$. The proof is done by contradiction as follows:

  Assume that $p(\tilde{a})$ is in $[SPF_k(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$ but that $p(\tilde{a})$ is not in $SPF'_k(SEM_{i-1}^{wf})$. In this case, as shown when $k = 0$, $p(\tilde{a}) \notin pos(\xi_\mathcal{L})$ and we have rules r, in Δ, and r', in $\Pi^+(\Delta)$:

  $$\begin{array}{rlcl} r: & p(\tilde{a}) & \longleftarrow & L_1, L_2, \ldots, L_m \\ r': & p(\tilde{a}) & \longleftarrow & L_1, L_2, \ldots, L_m, not\ nonp(\tilde{a}) \end{array}$$

  Since L_i $(1 \leq i \leq m)$ are positive or negative literals, we distinguish the cases where (i) L_i is positive and where (ii) L_i is negative.

  (i) If L_i is a positive literal then $L_i \in SPF_{k-1}(SEM_{i-1})$ (and $L_i \notin neg(\xi_\mathcal{L})$). Thus, by induction hypothesis, we have that L_i is in $SPF'_{k-1}(SEM_{i-1})$.

  (ii) If L_i is a negative literal then $(not\ L_i) \notin (SEM_{i-1})$, and as $SEM'_{i-1} \subseteq SEM_{i-1}$, we can conclude that $(not\ L_i)$ is not in (SEM'_{i-1}).

  Thus, the only way to obtain $p(\tilde{a}) \notin SPF'_k(SEM_{i-1})$ is to have $nonp(\tilde{a}) \in SEM_{i-1}^{wf}$. However, this is possible only when $p(\tilde{a}) \in neg(\xi_\mathcal{L})$ due to the fact that *nonp* is an extensional predicate. But, if $p(\tilde{a}) \in neg(\xi_\mathcal{L})$ then $p(\tilde{a}) \notin [SPF_0(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$. A contradiction!

Thus, we proved that $[SPF_{\Delta}(SEM_{i-1}) \cup pos(\xi_{\mathcal{L}})] \setminus neg(\xi_{\mathcal{L}}) \subseteq SPF'_{\Pi^+(\Delta)}(SEM^{wf}_{i-1})$. Therefore, $U'_i \subseteq U_i$, and thus, if *not* $p(\tilde{a}) \in SEM'_i$ (*i.e.*, $p(\tilde{a}) \in U'_i$) then *not* $p(\tilde{a}) \in SEM_i$ (*i.e.*, $p(\tilde{a}) \in U_i$). This proves that $SEM'_i \subseteq SEM_i$, which is a contradiction to our hypothesis of induction that l belongs to SEM'_i but not to SEM_i.

Thus, we showed that $SEM'(\Delta) \subseteq SEM(\Delta)$, finishing the first part of our proof.

(2) To prove $SEM(\Delta) \subseteq SEM'(\Delta)$, we show by induction that for every i, there exists j such that $SEM_i \subseteq SEM'_{i+j}$.

1. It is clear that $SEM_0 = \xi_{\mathcal{L}}$. On the other hand, the construction of $\Pi^+(\Delta)$ guarantees that: (i) every positive literal $l = p(\tilde{a})$ in $\xi_{\mathcal{L}}$ is a fact in $\Pi^+(\Delta)$, and thus l is in $SEM^{wf}_1(\Pi^+(\Delta))$, and ($ii$) for every negative literal $l =$ *not* $p(\tilde{a})$ in $\xi_{\mathcal{L}}$, where p is an intensional predicate, the fact, $nonp(\tilde{a})$, is put in $\Pi^+(\Delta)$.
   In case (ii), *not* $p(\tilde{a})$ appears in $SEM^{wf}_1(\Pi^+(\Delta)$. Moreover, in $\Pi^+(\Delta)$, there are rules whose head is $p(\tilde{a})$ and that have *not* $nonp(\tilde{a})$ in their bodies. Since $nonp(\tilde{a})$ appears in $SEM^{wf}_1(\Pi^+(\Delta))$, the fact $p(\tilde{a})$ will be derived as unfounded. In other words, *not* $p(\tilde{a})$ is in $SEM^{wf}_j(\Pi^+(\Delta))$, for $j > 1$. Thus, we proved that $SEM_0 \subseteq SEM'_{0+j}$.

2. We assume now that there is a j for which $SEM_{i-1} \subseteq SEM'_{i-1+j}$, and we want to prove that there is a j' for which $SEM_i \subseteq SEM'_{i+j'}$.
   The proof is done by contradiction and, so, we consider that there is a literal l such that $l \in SEM_i$ ($l \notin SEM_{i-1}$) and $l \notin SEM'_{i+j'}$.

   Firstly, we consider that l is a **positive literal** ($l = p(\tilde{a})$), therefore, l is a consequence of a rule in Δ. This means that $p(\tilde{a})$ is in SEM_i because there is a rule r in Δ such that:

   $$p(\tilde{a}) \longleftarrow L_1, L_2, \ldots, L_m$$

   On the other hand, we know that there is a rule r', in $\Pi^+(\Delta)$, that corresponds to the query-driven rule r :

   $$p(\tilde{a}) \longleftarrow L_1, L_2, \ldots, L_m, not\ nonp(\tilde{a}).$$

   As $p(\tilde{a})$ is in SEM_i then all literals in the body of r are in SEM_{i-1}. By induction hypothesis, there is a j', for which $L_1, L_2, \ldots, L_m$ are also in $SEM'_{i-1+j'}$. Therefore, to obtain $p(\tilde{a}) \notin SEM'_{i+j'}$, we have to consider $p(\tilde{a}) \in neg(\xi_{\mathcal{L}})$. Indeed, if $p(\tilde{a}) \in neg(\xi_{\mathcal{L}})$ then the literal $nonp(\tilde{a})$ is in $SEM'_{i-1+j'}$. Obviously, in this situation, rule r' cannot be used to derive $p(\tilde{a})$. That is, $p(\tilde{a}) \notin SEM'_{i+j'}$. However, in this case, it is clear (by the computation of SEM_i) that $p(\tilde{a}) \notin SEM_i$. A contradiction!

   Briefly, if $p(\tilde{a}) \in neg(\xi_{\mathcal{L}})$ then $p(\tilde{a}) \notin SEM_i$ and $p(\tilde{a}) \notin SEM'_{i+j'}$. However, if $p(\tilde{a}) \notin neg(\xi_{\mathcal{L}})$ and $p(\tilde{a}) \in SEM_i$ then $p(\tilde{a}) \in SEM'_{i+j'}$.

   Secondly, we consider that l is a **negative literal** ($l =$ *not* $p(\tilde{a})$). If *not* $p(\tilde{a})$ is in SEM_i then $p(\tilde{a})$ is not a potentially founded fact. Therefore, $p(\tilde{a})$ is not in $SPF_{\Delta}(SEM_{i-1})$ and, as a consequence, $p(\tilde{a}) \in U_i$. On the other hand, if *not* $p(\tilde{a})$ is not in $SEM'_{i-1+j'}$ then $p(\tilde{a})$ is a potentially founded fact, *i.e.*, $p(\tilde{a})$ belongs to $SPF'_{\Pi^+(\Delta)}(SEM^{wf}_{i-1+j'})$ and, as a consequence, $p(\tilde{a}) \notin U'_{i+j'}$.

   To prove that such a situation is *not* possible we will show that $U_i \subseteq U'_{i+j'}$.

As in the first part of this proof, if HB_Δ is the Herbrand Base with respect to the database Δ, then we have:

$$\begin{array}{rcl} U_i & = & [(HB_\Delta \setminus SPF_\Delta(SEM_{i-1})) \setminus pos(\xi_\mathcal{L})] \cup neg(\xi_\mathcal{L}) \\ & = & [HB_\Delta \setminus (SPF_\Delta(SEM_{i-1}) \cup pos(\xi_\mathcal{L}))] \cup neg(\xi_\mathcal{L}) \\ U'_{i+j} & = & HB_\Delta \setminus SPF'_{\Pi^+(\Delta)}(SEM^{wf}_{i-1+j}) \end{array}$$

In other words, we want to prove that

$$SPF'_{\Pi^+(\Delta)}(SEM^{wf}_{i-1+j}) \subseteq [SPF_\Delta(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L}).$$

The inclusion is proved by induction on k.

- We consider $k=0$, $p(\tilde{a}) \in SPF'_0(SEM^{wf}_{i-1+j'})$ and $p(\tilde{a}) \notin [SPF_0(SEM_{i-1})) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$.

  Therefore, we know that there are rules r, in Δ, and r', in $\Pi^+(\Delta)$, such that:

  $$\begin{array}{rlcl} r: & p(\tilde{a}) & \longleftarrow & L_1, L_2, \ldots, L_m \\ r': & p(\tilde{a}) & \longleftarrow & L_1, L_2, \ldots, L_m, not\ nonp(\tilde{a}) \end{array}$$

  where L_i $(1 \le i \le m)$ are negative literals, with $(not\ L_i) \notin SEM'_{i-1+j'}$. Since $SEM_{i-1} \subseteq SEM'_{i-1+j'}$, we know that $(not\ L_i) \notin SEM_{i-1}$.

  Thus, the only way to obtain $p(\tilde{a}) \notin [SPF_0(SEM_{i-1})) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$ is to have $p(\tilde{a}) \in neg(\xi_\mathcal{L})$. But, in this case, $nonp(\tilde{a}) \in \Pi^+(\Delta)$ and $p(\tilde{a})$ will be unfounded in $SEM'_{i+j'}$ for some j'. And this is a contradiction to our assumption. Thus, we proved that $SPF'_0(SEM^{wf}_{i-1+j'}) \subseteq [SPF_0(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$.

- We suppose now that $SPF'_{k-1}(SEM^{wf}_{i-1+j'}) \subseteq [SPF_{k-1}(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$, and we want to prove that $SPF'_k(SEM^{wf}_{i-1+j'}) \subseteq [SPF_k(SEM_{i-1})) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$. The proof is done by contradiction as follows:

  Assume that $p(\tilde{a})$ is not in $[SPF_k(SEM_{i-1})) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$ but that $p(\tilde{a})$ is in $SPF'_k(SEM^{wf}_{i-1+j'})$. In this case, we have rules r, in Δ, and r', in $\Pi^+(\Delta)$:

  $$\begin{array}{rlcl} r: & p(\tilde{a}) & \longleftarrow & L_1, L_2, \ldots, L_m \\ r': & p(\tilde{a}) & \longleftarrow & L_1, L_2, \ldots, L_m, not\ nonp(\tilde{a}) \end{array}$$

  Since L_i $(1 \le i \le m)$ are positive or negative literals, we distinguish the cases where (i) L_i is positive and where (ii) L_i is negative.

  (i) If L_i is a positive literal then $L_i \in SPF^{wf}_{k-1}(SEM'_{i-1+j'})$, and therefore, by induction hypothesis, we can conclude that L_i is in $[SPF_{k-1}(SEM_{i-1})) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$.

  (ii) If L_i is a negative literal then $(not\ L_i) \notin (SEM'_{i-1+j'})$, and as $SEM_{i-1} \subseteq SEM'_{i-1+j'}$, we can conclude that $(not\ L_i)$ is not in (SEM_{i-1}).

  Thus, here, the only way to obtain $p(\tilde{a}) \notin [SPF_k(SEM_{i-1})) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$ is to have $p(\tilde{a}) \in neg(\xi_\mathcal{L})$. But, in this case, as showed before for $k=0$, $p(\tilde{a}) \notin SPF'_k(SEM^{wf}_{i-1+j'})$. A contradiction!

Thus, we proved that $SPF'_k(SEM^{wf}_{i-1+j'}) \subseteq [SPF_k(SEM_{i-1}) \cup pos(\xi_\mathcal{L})] \setminus neg(\xi_\mathcal{L})$.

Therefore, $U_i \subseteq U'_{i+j}$, and thus if $not\ p(\tilde{a}) \in SEM_i$ (*i.e.*, $p(\tilde{a}) \in U_i$) then $not\ p(\tilde{a}) \in SEM'_{i+j'}$ (*i.e.*, $p(\tilde{a}) \in U'_{i+j'}$). This proves that $SEM_i \subseteq SEM'_{i+j'}$, which is a contradiction to our hypothesis of induction that l belongs to SEM_i but not to $SEM'_{i+j'}$.

Thus, we showed that $SEM(\Delta) \subseteq SEM'(\Delta)$, finishing the second part of the proof. As a consequence, from (1) and (2), we obtain $SEM(\Delta) = SEM'(\Delta)$, and the proof is complete. $\square$

Appendix B - Literal Rule Consistency

In this appendix, we are interested in the actual computation of all *inconsistent derivation paths* that can be found in a given set of l-rules LR. To this end, we first recall that we assume, as usual in Logic Programming, that each time a rule r is considered in a derivation, all variables occuring in r are *new* variables, occuring nowhere else in the derivation.

Now, in order to find which l-rules in LR can give raise to an inconsistent derivation path, we associate LR with a directed labeled graph $G(\text{LR})$ whose nodes, links and labels are defined as follows:

- *Nodes:* For every literal L, define

$$node(L) = \begin{cases} p & \text{if } L = p(\tilde{t}) \\ not\ p & \text{if } L = not\ p(\tilde{t}) \end{cases}$$

  Thus, for every l-rule $L_1 \leftarrow L_2$ in LR $node(L_1)$ and $node(L_2)$ are nodes of $G(\text{LR})$.

- *Links:* For every l-rule $L_1 \leftarrow L_2$ in LR there is a link from $node(L_2)$ to $node(L_1)$.

- *Labels of Links:* Every link (ν_1, ν_2) of $G(\text{LR})$ has a label, denoted by $\lambda(\nu_1, \nu_2)$ and defined as being the set of all rules $L \leftarrow L'$ in LR such that $node(L') = \nu_1$ and $node(L) = \nu_2$.

Example B.1 Consider the following set LR of l-rules and the associated graph $G(\text{LR})$.

$$\begin{array}{llllllll} \text{LR}: & 1: & q(a) & \leftarrow v(a) & 2: & not\ v(a) & \leftarrow & q(a) \\ & 3: & q(b) & \leftarrow v(b) & 4: & not\ v(b) & \leftarrow & q(x) \\ & 5: & v(x) & \leftarrow t(x) & & & & \end{array}$$

$G(\text{LR})$ contains four nodes, namely t, q, v and $not\ v$, and three links, namely (t, v), (v, q) and $(q, not\ v)$. Moreover, using the number of l-rules instead of the rules themselves, the labels of the links are the following: $\lambda(t, v) = \{5\}$, $\lambda(v, q) = \{1, 3\}$ and $\lambda(q, not\ v) = \{2, 4\}$. Therefore, $G(\text{LR})$ is the following graph:

$$t \xrightarrow{\{5\}} v \xrightarrow{\{1,3\}} q \xrightarrow{\{2,4\}} not\ v$$

Using the labels of the links, we find that the path from v to $not\ v$ in $G(\text{LR})$ corresponds to four different subsets of l-rules in LR, namely $\{1, 2\}$, $\{3, 4\}$, $\{1, 4\}$ and, $\{3, 2\}$. $\square$

Now, let $P = [\nu_1, \nu_2, \ldots, \nu_k]$ be a path in $G(\text{LR})$. We associate P with a set $\Sigma(P)$ containing pairs of the form $\langle \rho, \theta \rangle$ where:

- ρ is a sequence $[r_1, r_2, \ldots, r_{k-1}]$ of $k - 1$ l-rules in LR such that, for every $i = 1, 2, \ldots, k - 1$, $r_i \in \lambda(\nu_i, \nu_{i+1})$.

- Denoting by ϵ the identity substitution, θ is defined as follows:

  1. Let $\theta_1 = \epsilon$
  2. For every $i = 2, \ldots, k - 1$, define θ_i from θ_{i-1} by:
     if θ_{i-1} is undefined **then** $\theta_i = undefined$
     else let $\sigma = mgu[head(r_{i-1})\theta_{i-1}, body(r_i)]$;
       if σ is undefined **then** $\theta_i = undefined$
       else $\theta_i = \theta_{i-1}\sigma$
  3. $\theta = \theta_{k-1}$

In Example B.1 above, if we consider $P = [v, q, not\ v]$, $\Sigma(P)$ contains the following four pairs:

- $\langle \rho_1, \theta_1 \rangle$ with $\rho_1 = [1, 2]$ and $\theta_1 = \epsilon$
- $\langle \rho_2, \theta_2 \rangle$ with $\rho_2 = [3, 4]$ and $\theta_2 = \{x/b\}$
- $\langle \rho_3, \theta_3 \rangle$ with $\rho_3 = [1, 4]$ and $\theta_3 = \{x/a\}$
- $\langle \rho_4, \theta_4 \rangle$ with $\rho_4 = [3, 2]$ and $\theta_4 = undefined$.

It follows from Definition 3.4 that the substitution θ associated to a sequence ρ is built in the same way as the unifier of a rule path. Thus, pairs in $\langle \rho, \theta \rangle$ in $\Sigma(P)$ such that θ is not undefined correspond to derivation paths as defined in Definition 3.3. This is precisely what is stated in the following proposition.

Proposition B.1 *Let* LR *be a set of l-rules and let $P = [\nu_1, \nu_2, \ldots, \nu_k]$ be a path in the graph $G(\text{LR})$. For every $\langle \rho, \theta \rangle$ in $\Sigma(P)$, ρ is a rule path iff θ is not undefined. Moreover, in this case, θ is the unifier associated to ρ.* □

Now, inspired by the notions of inconsistent linear derivation path and of inconsistent fork derivation path (see Definition 3.5), we define the notions of *linear subgraph* and of *fork subgraph* of the graph $G(\text{LR})$ as follows:

Definition B.1 *Let* LR *be a set of l-rules and let $G(\text{LR})$ be the associated graph.*

1. *A* linear subgraph *of $G(\text{LR})$ is a path from node p to node not p or from node not p to node p.*
2. *A* fork subgraph *of $G(\text{LR})$ is a subgraph of $G(\text{LR})$ consisting in two paths, one from node q to node p and one from node q to node not p. Moreover, q is the only node that belongs to both of these paths.* □

Clearly, every inconsistent linear (respectively fork) derivation path in LR can be associated to a linear (respectively fork) subgraph of LR. Indeed, by Lemma 3.2, we know that every derivation using the rules in LR corresponds to a rule path π which itself, corresponds to a path in $G(\text{LR})$. Moreover, in the linear case the derivation starts from a literal L and terminates with the opposite literal *not* L, and we have a similar argument in the fork case. An important consequence of this remark and of Theorem 3.1 is that any set LR containing no linear subgraph and no fork subgraph is consistent. This is precisely what is stated in the following proposition.

Proposition B.2 *Let* LR *be a set of l-rules and let $G(\text{LR})$ be the associated graph. If* LR *is inconsistent then $G(\text{LR})$ contains a linear subgraph or a fork subgraph.* □

However, we note that the converse of Proposition B.2 does not hold. Consider for example the set LR containing the following three rules:

$$1: \quad p(a, x) \leftarrow q(a, b) \qquad 2: \quad not\ p(b, x) \leftarrow q(a, x) \qquad 3: \quad q(a, x) \leftarrow not\ q(b, c).$$

Then, if we use the number of each rule in the labels, the graph $G(\text{LR})$ is the following:

$$\begin{tikzcd} not\ q \arrow[r, "\{3\}"] & q \arrow[r, "\{1\}"] \arrow[d, "\{2\}"] & p \\ & not\ p & \end{tikzcd}$$

Thus, $G(\text{LR})$ contains a linear subgraph (see link labeled $\{3\}$) and a fork subgraph (see links labeled $\{1\}$ and $\{2\}$). However, there is no inconsistent derivation path in LR because:

1. $q(a, x)$, the head of rule 3, and $q(b, c)$, the body of rule 3, are not unifiable, and because

2. although $q(a,x)$ (the body of rule 2) and $q(a,b)$ (the body of rule 1) are unifiable, $p(b,x)$ (the negation of rule 2 head) and $p(a,x)$ (the head of rule 1) are *not* unifiable.

As a consequence of Proposition B.2, in order to compute all inconsistent derivation paths, we only have to consider the linear and fork subgraphs of $G(\textsc{lr})$. The important step to achieve our goal is now to calculate the substitutions associated to the potentially inconsistent derivation paths.

This is what is achieved in the following two algorithms, the first one corresponding to a linear subgraph and the second one corresponding to a fork subgraph. Note that in both algorithms, we use the following additional notation: let $\rho = [r_1, r_2, \ldots, r_k]$ be a sequence of l-rules, $body(r_1)$ and $head(r_k)$ are denoted by $body(\rho)$ and $head(\rho)$, respectively.

Algorithm B.1 - The linear case:

INPUT: A linear subgraph P in $G(\textsc{lr})$.
OUTPUT: Either *consistent* or *inconsistent*. For each case of inconsistency, the algorithm returns a partially instantiated literal.

```
compute Σ(P) ;
for every ⟨ρ, θ⟩ in Σ(P) do
begin
    if θ is not undefined
    then begin
        Θ = (mgu[body(ρ)θ, not head(ρ)θ])θ ;
        if Θ is not undefined
        then return inconsistent and body(ρ)Θ
        //* the set of l-rules in ρ is inconsistent
        //* for body(ρ)Θ
        else return consistent
        //* the set of l-rules in ρ is consistent
    end
    else return consistent
    //*the set of l-rules in ρ is consistent
end
```

We now consider the case of a fork subgraph consisting of the two paths P^1 and P^2.

Algorithm B.2 - The fork case

INPUT: A fork subgraph (P^1, P^2) in $G(\textsc{lr})$.
OUTPUT: Either *consistent* or *inconsistent*. For each case of inconsistency, the algorithm returns a partially instantiated literal.

```
compute Σ(P¹) ; compute Σ(P²) ;
for every ⟨ρ¹, θ¹⟩ in Σ(P¹) do
begin
    for every ⟨ρ², θ²⟩ in Σ(P²) do
    begin
(1)     if θ¹ and θ² are not undefined
        then begin
            Θ₀ = mgu[head(ρ¹)θ¹, not head(ρ²)θ²]
(2)         if Θ₀ is not undefined
            then begin
                Θ₁ = mgu[body(ρ¹)θ¹Θ₀, body(ρ²)θ²Θ₀]
(3)             if Θ₁ is not undefined
```

then return *inconsistent* and $body(\rho^1)\theta^1\Theta_0\Theta_1$
//* the set of l-rules in ρ^1 or in ρ^2 is inconsistent
//* for $body(\rho^1)\theta^1\Theta_0\Theta_1$ (or for $body(\rho^2)\theta^2\Theta_0\Theta_1$)
else return *consistent*
//* the set of l-rules in ρ^1 or in ρ^2 is consistent
end
else return *consistent*
//* the set of l-rules in ρ^1 or in ρ^2 is consistent
end
else return *consistent*
//* the set of l-rules in ρ^1 or in ρ^2 is consistent
end
end

Before stating the theorem which shows that algorithms B.1 and B.2 are correct, we consider the following example.

Example B.2 Consider the following set LR of l-rules and the associated graph $G(\text{LR})$.

$$
\begin{array}{rrcl rrcl}
1: & u(x,y) & \leftarrow & p(x,y) & 2: & not\ p(x,b) & \leftarrow & u(a,y) \\
3: & not\ p(y,d) & \leftarrow & u(y,x) & 4: & not\ p(c,e) & \leftarrow & u(a,b) \\
5: & s(x,y) & \leftarrow & p(x,y) & 6: & s(a,b) & \leftarrow & p(a,b) \\
7: & t(a,y) & \leftarrow & s(a,b) & 8: & q(x,y) & \leftarrow & p(x,y) \\
9: & q(a,b) & \leftarrow & p(b,a) & 10: & not\ t(x,a) & \leftarrow & q(b,a) \\
11: & not\ t(x,y) & \leftarrow & q(x,y) & 12: & not\ q(b,a) & \leftarrow & p(x,y)
\end{array}
$$

The associated graph (where the links are not labeled) $G(\text{LR})$ is the following:

$$
\begin{array}{ccccc}
 & & not\ q & & \\
 & & \uparrow & & \\
not\ p \longleftarrow & u \longleftarrow & p & \longrightarrow s & \longrightarrow t \\
 & & \downarrow & & \\
 & & q & & \\
 & & \downarrow & & \\
 & & not\ t & &
\end{array}
$$

In $G(\text{LR})$, we can distinguish one linear subgraph $SG_1(\text{LR})$ and two fork subgraphs $SG_2(\text{LR})$ and $SG_3(\text{LR})$. These subgraphs with their labeled links are the following:

$$SG_1(\text{LR}) : p \xrightarrow{\{1\}} u \xrightarrow{\{2,3,4\}} not\ p$$

$$
SG_2(\text{LR}) : \begin{array}{ll}
p & \xrightarrow{\{8,9\}} q \\
\downarrow^{\{12\}} & \\
not\ q &
\end{array}
$$

$$
SG_3(\text{LR}) : \begin{array}{lll}
p & \xrightarrow{\{5,6\}} s & \xrightarrow{\{7\}} t \\
\downarrow^{\{8,9\}} & & \\
q & & \\
\downarrow^{\{10,11\}} & & \\
not\ t & &
\end{array}
$$

We first consider the linear subgraph $SG_1(\text{LR})$ and we denote by P_1 the associated path in $G(\text{LR})$. Then, with $\Sigma(P_1)$ as input, Algorithm B.1 computes the following:

1. $\langle \rho_1, \theta_1 \rangle$ where $\rho_1 = [u(x_1, y_1) \leftarrow p(x_1, y_1),\ \mathit{not}\ p(x_2, b) \leftarrow u(a, y_2)]$ and $\theta_1 = \{x_1/a, y_1/y_2\}$. In this case, $\mathit{head}(\rho_1)\theta_1 = \mathit{not}\ p(x_2, b)$ and $\mathit{body}(\rho_1)\theta_1 = p(a, y_2)$, and so, we have $\Theta = \{x_1/a, y_1/b, x_2/a, y_2/b\}$. As a consequence, Algorithm B.1 returns "*inconsistent*" and the literal $p(a, b)$.

2. $\langle \rho_2, \theta_2 \rangle$ where $\rho_2 = [u(x_1, y_1) \leftarrow p(x_1, y_1),\ \mathit{not}\ p(y_2, d) \leftarrow u(y_2, x_2)]$ and $\theta_2 = \{x_1/y_2, y_1/x_2\}$. In this case, $\mathit{head}(\rho_2)\theta_2 = \mathit{not}\ p(y_2, d)$ and $\mathit{body}(\rho_2)\theta_1 = p(y_2, x_2)$ and so, we have $\Theta = \{x_1/y_2, x_2/d, y_1/d\}$. As a consequence, Algorithm B.1 returns "*inconsistent*" and the literal $p(y_2, d)$.

3. $\langle \rho_3, \theta_3 \rangle$ where $\rho_3 = [u(x_1, y_1) \leftarrow p(x_1, y_1),\ \mathit{not}\ p(c, e) \leftarrow u(a, b)]$ and $\theta_3 = \{x_1/a, y_1/b\}$. In this case, $\mathit{head}(\rho_3)\theta_2 = \mathit{not}\, p(c, e)$ and $\mathit{body}(\rho_3)\theta_1 = p(a, b)$, and so, Θ is undefined. As a consequence, Algorithm B.1 returns "*consistent*".

Let us now consider the fork subgraphs $SG_2(\textsc{lr})$ and we denote by P^1 and P^2 the associated paths in $G(\textsc{lr})$. Then, with $\Sigma(P^1)$ and $\Sigma(P^2)$ as input, Algorithm B.2 always computes the identity substitution ϵ for θ^1 and θ^2, since both paths are of length 1. Moreover, we have two possible cases to consider, since $\Sigma(P^1)$ contains one pair and $\Sigma(P^2)$ contains two pairs. We now examine each of these two cases one by one:

1. Consider $\langle \rho_1^1, \theta_1^1 \rangle$ with $\rho_1^1 = [\mathit{not}\ q(b, a) \leftarrow p(x_1, y_1)]$ and $\theta_1^1 = \epsilon$ in $\Sigma(P^1)$ and $\langle \rho_1^2, \theta_1^2 \rangle$ with $\rho_1^2 = [q(x_2, y_2) \leftarrow p(x_2, y_2)]$ and $\theta_1^2 = \epsilon$ in $\Sigma(P^2)$. In this case, $\Theta_0 = \{x_2/b, y_2/a\}$, and so, $\Theta_1 = \mathit{mgu}[p(b, a), p(x_1, y_1)]$. Thus, $\Theta_1 = \{x_1/b, y_1/a\}$ and $\theta_1^1\Theta_0\Theta_1 = \{x_1/b, y_1/a, x_2/b, y_2/a\}$. Therefore, Algorithm B.2 returns "*inconsistent*" and the literal $p(b, a)$.

2. Consider $\langle \rho_1^1, \theta_1^1 \rangle$ as above in $\Sigma(P^1)$ and $\langle \rho_2^2, \theta_2^2 \rangle$ with $\rho_2^2 = [q(a, b) \leftarrow p(b, a)]$ and $\theta_2^2 = \epsilon$ in $\Sigma(P^2)$. In this case, Θ_0 is undefined, because it is impossible to unify $q(b, a)$ with $q(a, b)$ (if statement (2) fails). Therefore, Algorithm B.2 returns "*consistent*".

Let us now consider the fork subgraphs $SG_3(\textsc{lr})$ and we denote by P^1 and P^2 the associated paths in $G(\textsc{lr})$. Then, we have eight possible cases to consider, since $\Sigma(P^1)$ contains two pairs and $\Sigma(P^2)$ contains four pairs. We outline each of these cases as follows:

1. Consider $\langle \rho_1^1, \theta_1^1 \rangle$ with $\rho_1^1 = [s(x_1, y_1) \leftarrow p(x_1, y_1),\ t(a, y_2) \leftarrow s(a, b)]$ and $\theta_1^1 = \{x_1/a, y_1/b\}$ in $\Sigma(P^1)$ and $\langle \rho_1^2, \theta_1^2 \rangle$ with $\rho_1^2 = [q(x_3, y_3) \leftarrow p(x_3, y_3), \mathit{not}\, t(x_4, a) \leftarrow q(b, a)]$ and $\theta_1^2 = \{x_3/b, y_3/a\}$ in $\Sigma(P^2)$. In this case, $\Theta_0 = \{x_4/a, y_2/a\}$ and Θ_1 is undefined, since we have $p(x_1, y_1)\theta_1^1\Theta_0 = p(a, b)$ and $p(x_3, y_3)\theta_1^2\Theta_0 = p(b, a)$ (if statement (3) fails). Therefore, Algorithm B.2 returns "*consistent*".

2. Consider $\langle \rho_1^1, \theta_1^1 \rangle$ as above in $\Sigma(P^1)$ and $\langle \rho_2^2, \theta_2^2 \rangle$ with $\rho_2^2 = [q(x_3, y_3) \leftarrow p(x_3, y_3), \mathit{not}\, t(x_4, y_4) \leftarrow q(x_4, y_4)]$ and $\theta_2^2 = \epsilon$ in $\Sigma(P^2)$. In this case, $\Theta_0 = \{x_4/a, y_4/y_2\}$, and $\Theta_1 = \{x_3/a, y_3/b\}$. Thus, $\theta_1^1\Theta_0\Theta_1 = \{x_1/a, y_1/b, x_3/a, y_3/b, x_4/a, y_4/y_2\}$, and so, Algorithm B.2 returns "*inconsistent*" and the literal $p(a, b)$.

3. Consider $\langle \rho_1^1, \theta_1^1 \rangle$ as above in $\Sigma(P^1)$ and $\langle \rho_3^2, \theta_3^2 \rangle$ with $\rho_3^2 = [q(a, b) \leftarrow p(b, a), \mathit{not}\ t(x_4, a) \leftarrow q(b, a)]$ and $\theta_3^2 = \mathit{undefined}$ in $\Sigma(P^2)$. In this case, if statement (1) fails, and so, Algorithm B.2 returns "*consistent*".

4. Consider $\langle \rho_1^1, \theta_1^1 \rangle$ as above in $\Sigma(P^1)$ and $\langle \rho_4^2, \theta_4^2 \rangle$ with $\rho_4^2 = [q(a, b) \leftarrow p(b, a), \mathit{not}\ t(x_4, y_4) \leftarrow q(x_4, y_4)]$ and $\theta_4^2 = \{x_4/a, y_4/b\}$ in $\Sigma(P^2)$. In this case, $\Theta_0 = \{y_2/b\}$, and so, Θ_1 must unify $p(a, b)$ with $p(b, a)$. Thus, Θ_1 is undefined (if statement (3) fails). Therefore, Algorithm B.2 returns "*consistent*".

5. Consider $\langle\rho_2^1,\theta_2^1\rangle$ with $\rho_2^1 = [s(a,b) \leftarrow p(a,b), t(a,y_2) \leftarrow s(a,b)]$ and $\theta_2^1 = \epsilon$ in $\Sigma(P^1)$ and $\langle\rho_1^2,\theta_1^2\rangle$ as in item 1 above in $\Sigma(P^2)$. In this case, $\Theta_0 = \{x_4/a, y_2/a\}$, and Θ_1 is undefined (if statement (3) fails). Therefore, Algorithm B.2 returns "*consistent*".

6. Consider $\langle\rho_2^1,\theta_2^1\rangle$ as above in $\Sigma(P^1)$ and $\langle\rho_2^2,\theta_2^2\rangle$ as in item 2 above in $\Sigma(P^2)$. Since $\theta_2^1 = \theta_2^2 = \epsilon$, we have $\Theta_0 = \{x_4/a, y_4/y_2\}$. Moreover, $\Theta_1 = \{x_3/a, y_3/b\}$. Therefore, $\theta_2^1\Theta_0\Theta_1 = \{x_3/a, y_3/b, x_4/a, y_4/y_2\}$, and so, Algorithm B.2 returns "*inconsistent*" and the literal $p(a,b)$.

7. Consider $\langle\rho_2^1,\theta_2^1\rangle$ as above in $\Sigma(P^1)$ and $\langle\rho_3^2,\theta_3^2\rangle$ as in item 3 above in $\Sigma(P^2)$. Here again, if statement (1) fails, and so, Algorithm B.2 returns "*consistent*".

8. Consider $\langle\rho_2^1,\theta_2^1\rangle$ as above in $\Sigma(P^1)$ and $\langle\rho_4^2,\theta_4^2\rangle$ as in item 4 above in $\Sigma(P^2)$. As in item 4, Θ_1 is undefined, and so, Algorithm B.2 returns "*consistent*". □

We can now present a characterization of inconsistent derivation paths based on the algorithms B.1 and B.2.

Theorem B.1 *Let* LR *be a set of l-rules and let $G(\text{LR})$ be the associated graph. The set* LR *contains an inconsistent derivation path if and only if one of the following holds:*

1. *The graph $G(\text{LR})$ has a linear subgraph for which Algorithm B.1 returns "inconsistent".*
2. *The graph $G(\text{LR})$ has a fork subgraph for which Algorithm B.2 returns "inconsistent".* □

Proof: We first assume that LR contains an inconsistent derivation path. Then, according to Definition 3.5, we have to distinguish two cases: (i) the case of an inconsistent linear derivation path, and (ii) the case of an inconsistent fork derivation path. By Proposition B.2, cases (i) and (ii) correspond to the existence in $G(\text{LR})$ of a linear or a fork subgraph, respectively.
(i) Let $\langle\pi,\sigma\rangle$ be an inconsistent linear derivation path where $\pi = [r_1, r_2, \ldots, r_k]$. Then, denoting by P the associated path in $G(\text{LR})$, by Proposition B.1, $\Sigma(P)$ contains the pair $\langle\pi,\theta(\pi)\rangle$. Therefore, in Algorithm B.1 applied with P as input, Θ is computed, and moreover, we know that $body(r_1)\theta(\pi)\sigma = not\ head(r_k)\theta(\pi)\sigma$. Thus, $mgu[body(\pi)\theta(\pi), head(\pi)\theta(\pi)]$ is not undefined. Therefore, Algorithm B.1 returns "inconsistent".
(ii) Now, let $(\langle\pi^1,\sigma^1\rangle, \langle\pi^2,\sigma^2\rangle)$ be an inconsistent fork derivation path where $\pi^1 = [r_1^1, r_2^1, \ldots, r_{k_1}^1]$ and where $\pi^2 = [r_1^2, r_2^2, \ldots, r_{k_2}^2]$. Then, as above, if we call (P^1, P^2), the associated fork subgraph in $G(\text{LR})$, when applying Algorithm B.2 with (P^1, P^2) as input, we find $\langle\pi^1,\theta(\pi^1)\rangle$ in $\Sigma(P^1)$ and $\langle\pi^2,\theta(\pi^2)\rangle$ in $\Sigma(P^2)$ such that $\theta(\pi^1)$ and $\theta(\pi^2)$ are not undefined. Moreover, as we know that $head(r_{k_1}^1)\theta(\pi^1)\sigma^1 = not\ head(r_{k_2}^2)\theta(\pi^2)\sigma^2$, Θ_0 is not undefined. Since we also have that $body(r_1^1)\theta(\pi^1)\sigma^1 = body(r_2^2)\theta(\pi^2)\sigma^2$, the unifier Θ_1 is not undefined either. Thus, Algorithm B.2 returns "inconsistent".

Conversely, assume that Algorithm B.1 returns "inconsistent" for a linear subgraph $SG_L(\text{LR})$ of $G(\text{LR})$. It is easy to see, in the same way as in case (i) above, that, in this case, that $SG_L(\text{LR})$ corresponds to an inconsistent linear derivation path in LR. It can be shown in a similar way that, in the case of Algorithm B.2, $SG_L(\text{LR})$ corresponds to an inconsistent fork derivation path in LR. Thus, the proof of the theorem is complete. □

As a consequence of theorems 3.1 and B.1, a set of l-rules LR is inconsistent if and only if there exists a linear (or, respectively, a fork) subgraph in $G(\text{LR})$ for which Algorithm B.1 (or, respectively, for which Algorithm B.2) outputs "inconsistent". Moreover, the following proposition shows that the partially instantiated literals returned by all possible calls of algorithms B.1 or B.2 characterize the literals which make the set LR to be inconsistent.

Proposition B.3 *Let* LR *be a set of l-rules. For every ground literal L, ξ_L is inconsistent iff there exists a linear or a fork subgraph in $G(\text{LR})$ for which Algorithm B.1 or Algorithm B.2 outputs "inconsistent" and a literal l having a ground instance that belongs to ξ_L.* □

Proof: By Theorem 3.1, ξ_L is inconsistent iff there exists an inconsistent derivation path in LR involving L. By Theorem B.1, this is equivalent to the fact that a call of Algorithm B.1 or of Algorithm B.2 outputs "inconsistent" together with a literal l. In this case, a similar reasoning as in the proof or Theorem B.1 shows that ξ_L is inconsistent iff ξ_L contains a ground instance of l. Thus the proof is complete. □

Referring back to Example B.2, Proposition B.3 shows that if we consider a database $\Delta = (\mathcal{L}, UR, QR)$ where UR is the *inconsistent* set of l-rules as shown in this example, then the *only* updates that must be rejected are the insertions of $p(a, b)$ or of $p(b, a)$, or of any fact of the form $p(\alpha, d)$ where α is an arbitrary constant.