Logic Programming

Proceedings of the Seventh International Conference

edited by David H. D. Warren and Peter Szeredi

<7, 1990, Jerusalem>

The MIT Press
Cambridge, Massachusetts
London, England

This book was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Logic programming: proceedings of the seventh international conference/
edited by David H. D. Warren and Peter Szeredi.

p. cm. — (Logic programming)
Papers from the Seventh International Conference on Logic Programming, held in Jerusalem, June 18-20, 1990.
Includes index.
ISBN 0-262-73090-1
1. Logic programming—Congresses. I. Warren, David H. D. II. Szeredi, Peter, 1949- . III. International Conference on Logic Programming (1990: Jerusalem) IV. Series.
QA76.63.L643 1990
006.3—dc20 90-6074
CIP

Contents

Program Committee

Intensional Updates: Abduction via Deduction

François Bry
ECRC, Arabellastraße 17, D-8000 München 81, West Germany
fb@ecrc.de

Abstract

Because they are extra-logical and procedural, the conventional updating facilities of Logic Programming are not convenient for all applications. In particular, it is desirable to give database and expert system users the possibility to specify updates without having to define their execution. We propose the concept of 'intensional update' as a formal basis for declarative update languages. Intensional updates express in first-order logic changes of the consequences of logic programs. They generalize database 'view updates'. The modifications of the logic program that actually realize an intensional update are derived through abductive reasoning. We show that this form of abduction can be reduced to deduction in a non-disjunctive, definite theory, thus giving rise to implementations in Logic Programming. First, we formalize abduction as deduction in a disjunctive theory. Second, we apply the theorem prover Satchmo [22], which express deduction in disjunctive theories through definite meta-clauses. This approach gives rise to efficiently taking static and dynamic integrity constraints into account and to achieving completeness.

1 Introduction

The conventional updating facilities of Logic Programming do not have a semantics independent from the proof procedure. The effect of Prolog 'assert' and 'retract' built-in predicates depends on their positions in bodies of clauses and on the ordering of clauses in the program. Similarly, the changes resulting from the updating primitives proposed for bottom-up Logic Programming in [1, 2] depends on the processing order of clauses. Though useful in certain contexts — e.g., in programming languages —, such extra-logical, procedural facilities are not always convenient. In particular, database, knowledge base, or expert system users often require more declarative tools in order to specify an update without having to define an execution mode. In this article, we propose the notion of 'intensional update' as a formal basis for proof-procedure independent, declarative update languages.

Instead of defining updating operations, and explicitly or implicitly specifying how to perform them, it is possible to describe properties that the facts derivable from the updated program have to satisfy. Doing so, the update of the logic program is not specified, as usual, extensionally as changes to perform, but intensionally by the expected effect. We call 'intensional update' the latter notion. Intensional updates are not restricted to ground atomic formulas but can express more general properties, for example statements involving quantifiers, or referring to the current logic program as well as to its updated version. Hence, they extend the database notion of 'view update' — see, e.g., [10, 21, 28] — and the updates considered in [17, 11, 25, 19].

Consider for example a database of university employees. Assume that the professors that are qualified for tenure are defined in a 'view', i.e., by means of non-factual clauses. A view update permits one to express that a given professor should become qualified for tenure. In contrast, the more general concept of 'intensional update' gives rise to requiring a modification of the database such that, say, all professors currently teaching Logic Programming should qualify for tenure in the new, updated database. So, intensional updates extend view updates by allowing on the one hand quantified updating intentions, on the other hand intentions referring to the current program and to its updated version.

We rely on a meta-predicate for expressing intensional updates in logic. More precisely, we express statements that describe the desired, updated logic program within a meta-predicate called 'new'. Thus, they are distinguished from assertions on the current, non-updated logic program, giving rise to expressing properties related to both states of a logic program in a same intensional update. In this formalism, the intensional update considered above could be expressed as:

$$\forall x \quad \text{professor}(x) \land \text{teach}(x, \text{lp}) \Rightarrow \text{new}(\text{qualified-for-tenure}(x))$$

This formalism also gives rise to declaratively expressing dynamic integrity constraints. A classical example is a constraint requiring that the salary (denoted x) of an employee (denoted y) never decreases:

$$\forall x x_1 y \quad \text{salary}(x, y) \land \text{new}(\text{salary}(x_1, y)) \Rightarrow x_1 \geq x$$

Although they are defined with a meta-predicate, one can show that intensional updates and dynamic integrity constraints have a first-order logic semantics.

Intensional specifications of updates must be translated into actual changes of the logic program. In other words, it is necessary to generate, from an intensional update, 'extensional updates' that realize it. With the aim of automatizing this generation, we investigate the reasoning it involves. Consider for example the following logic program:

$$\begin{array}{ll} \text{p}(x) \leftarrow \text{q}(x) \land \text{r}(x) & \qquad \text{r(a)} \\ \text{q}(x) \leftarrow \text{s}(x) & \qquad \text{s(a)} \end{array}$$

and assume that the updating intention is to make the atom p(a) no longer derivable, i.e., in our formalism new(¬p(a)). This intention can be achieved by realizing new(¬q(a)), or by realizing new(¬r(a)) — disjunctive reasoning. Assuming that it is realized through new(¬q(a)) — hypothesis formation — the second clause yields the intensional update new(¬s(a)), which is realized by removing the fact s(a) from the program.

Although performing backward chaining like Linear and SLD Resolution do, the reasoning involved in such a process is not deductive. Instead of drawing conclusions from given premises, it infers possible causes for the intensional update. It performs disjunctive reasoning and relies on hypothesis formation. Following Peirce [24], we call this kind of inference 'abductive reasoning'.

Abductive reasoning has been primarily applied to diagnostic tasks [26]. Finding possible explanations for malfunctions or diseases indeed consists of making hypotheses that permit to justify the observed symptoms. Abductive reasoning has also been applied to natural language understanding [8, 23, 18], to design synthesis [15], and to formalizing analogical reasoning [27].

In previous studies, links between abductive reasoning and Logic Programming have been investigated in several directions. It has been proposed to take advantage of the fact that abduction performs backward chaining for expressing it as a — significantly — modified Linear or SLD Resolution procedure. This approach was first described in [27] with Linear Resolution, more recently in [17, 11, 25] with

SLDNF Resolution for the special task of realizing certain intensional updates. Abduction has also been considered for extending Logic Programming [12, 13, 9].

We propose to bring abduction and Logic Programming closer together in a way that differs from these approaches. First, we apply abduction to Logic Programming for giving formal bases to logical, proof-procedure independent update languages. Second, we rely on Logic Programming for expressing abduction via deduction. We apply the Prolog meta-interpreter Satchmo [22] for processing disjunctions and performing hypothetical reasoning. Therefore, we argue that there is no need to extend Logic Programming with abduction, nor to modify SLD Resolution for achieving abductive reasoning.

An intensional update which is realized by certain transformations of the logic program is often also realized by more substantial changes. We introduce the concept of 'minimal realization'. In order to be effective, a method for realizing intensional updates has to be complete. Intuitively, this means that each modification of the logic program that yields a program satisfying a given intensional update must be represented by one of the extensional updates generated by the method. We rely on results we established in the framework of the Satchmo project [7] for establishing the soundness and completeness of our approach.

The article consists of seven sections, the first of which is this introduction. We give definitions and make working hypotheses in Section 2. In Section 3, we formally introduce the notion of 'intensional update'. We formalize intensional update realization as deduction in non-Horn logic in Section 4. In Section 5, we describe the Satchmo approach to deduction in non-Horn theories. We apply this approach to realizing intensional updates in Section 6. In Section 7, we outline salient characteristics of our approach and we indicate directions for further research.

2 Definitions and Hypotheses

We consider clauses of the form $H \leftarrow L_1 \wedge \ldots \wedge L_n$ where H is an atom and the L_is are literals. Abusing the terminology, we call such a clause *definite* for emphasizing that H is not a disjunction. A clause is a *fact* if $n = 0$. If p is the predicate occurring in H, the clause is said to *define* p.

We consider *logic programs* P consisting of finite sets Cl(P) of clauses and of finite sets of *integrity constraints*. We do not assume any particular semantics for logic programs with non-positive rule's bodies. The formalization of intensional updates we give is independent from the semantics. It applies in particular to sets of clauses with a binary semantics — e.g., stratified, locally stratified, or constructively consistent set of clauses [3] — as well as to clauses with a ternary semantics — e.g., well-founded sets of clauses [30].

Given a logic program or theory P, $\mathcal{L}(P)$ denotes the language of P, i.e., the set of predicate, function, variable, and constant symbols occurring in P. $\mathcal{H}(P)$ denotes the Herbrand base of P, i.e., the set of ground atoms that can be constructed with the symbols in $\mathcal{L}(P)$. $\mathcal{F}(P)$ denotes the set of well-formed formulas expressed in the language $\mathcal{L}(P)$ of P. If P is a logic program, then $\mathcal{T}(P)$ denotes the set of ground literals that are provable from Cl(P) according to the considered semantics. If F is a closed formula, we shall write $P \models F$ if $\mathcal{T}(P) \vdash F$ where $\vdash$ denotes the provability relationship of classical logic. Italic lower case letters denote variables, roman lower case letters and words are used for constants, predicates, and function symbols. Capital letters denote formulas, logic programs, and theories.

In this study, we consider *static* and *dynamic* integrity constraints. Static integrity constraints are closed formulas in the same language as clauses. They are used for stating properties that are not expressible by means of clauses, e.g., disjunctive information. Classically, they are not used for generating answers to query, but as specifications of the logic program. A logic program P is said to be *consistent* if, for all static integrity constraints F of P we have P $\models$ F. Dynamic integrity constraints are defined in Section 3.

A dependency relationship on the predicates of a logic program is inductively defined as follows. A predicate p *depends* on each predicate occurring in the body of a clause defining p, and on each predicate on which one of these body predicates depends. A predicate which depends on itself is said to be *recursive*. A logic program is *recursive* if one of its predicates is recursive.

In Section 5, we consider implications $A_1 \wedge \ldots \wedge A_n \Rightarrow C_1 \vee \ldots \vee C_m$ with non-atomic, disjunctive conclusions. The predicate of a conclusion literal C_j *depends* on the predicates of premise literals A_is. In a set of implications, a predicate p depending on another predicate q also depends on the predicates on which q depends. p is *recursive* in a set of implications S if it depends on itself in S. A set of implications is *recursive* if one of its predicate is recursive.

We consider quantified queries, in particular for expressing integrity constraints. We assume that the quantifications are *restricted* [5], i.e., roughly quantified expressions have the following forms $\forall x$ R $\Rightarrow$ F, $\forall x$ $\neg$R, $\exists x$ S $\wedge$ F, or $\exists x$ S where F is a formula and where R and S are atoms or conjunctions of atoms containing x, more generally, R and S are ranges for x [5]. In order to ensure termination of constructive proofs of universally quantified expressions, we assume moreover that the R expressions admit finitely many answers. Restricted quantifications can be evaluated with the following meta-programs:

```
forall(X, R => F)  :-   not (R, not F).          exists(X, G)  :-   G.
```

Assuming that 'false' is an undefined atom — its evaluation always fails — permits us to represent formulas of the form $\forall x$ $\neg$R, where R is a range for x, as forall(X, R => false).

We constrain the syntax of logic programs on which intensional updates can be defined by precluding extra-logical, "side-effect" predicates such as '!', 'assert', or 'retract'. This hypothesis should not be considered as a restriction, since this study aims to free Logic Programming from extra-logical aspects.

3 Intensional Updates as Logical Theories

A logic program P and an updated version P_{new} of it can be viewed as two distinct logical theories. According to this "timeless" view, a declarative expression of an update of P resulting in P_{new} is a logical formula, or a finite set of logical formulas defining P_{new} in terms of P. In other words, it is a set of formulas in the language $\mathcal{L}(P) \cup \mathcal{L}(P_{new})$. In order to express such formulas, it is necessary to distinguish $\mathcal{L}(P)$ from $\mathcal{L}(P_{new})$. We propose to rely on a unary meta-predicate for defining a language for intensional updates. More precisely, we define a set $\mathcal{F}_{new}(P)$ of formulas as an extension of $\mathcal{F}(P)$ based on a unary meta-predicate 'new'.

Definition 3.1 *Given a logic program* P, *the set of formulas* $\mathcal{F}_{new}(P)$ *is recursively defined as follows:*

- F $\in \mathcal{F}_{new}(P)$ *if* F $\in \mathcal{F}(P)$
- new(F) $\in \mathcal{F}_{new}(P)$ *if* F $\in \mathcal{F}(P)$

- $\neg F \in \mathcal{F}_{\text{new}}(P)$ *if* $F \in \mathcal{F}_{\text{new}}(P)$
- $F\ \theta\ G \in \mathcal{F}_{\text{new}}(P)$ *if* $F \in \mathcal{F}_{\text{new}}(P)$, $G \in \mathcal{F}_{\text{new}}(P)$, *and* θ *is a logical connective*
- $Qx\ F \in \mathcal{F}_{\text{new}}(P)$ *if* $F \in \mathcal{F}(P)$, x *is a variable in* $\mathcal{L}(P)$ *free in* F, *and* Q *denotes* $\forall$ *or* $\exists$

The 'new' meta-predicate can be viewed as "prefixing" assertions related to the new, updated logic program. Note that no quantifications of meta-variables are allowed in $\mathcal{F}_{\text{new}}(P)$. This restriction ensures that intensional updates have a first-order logic semantics.

Definition 3.1 also gives a language for dynamic integrity constraints. Such constraints are considered in databases for defining legal updating transactions. They relate the new, updated database to the previous one. Like intensional updates, they can be defined as closed formulas in $\mathcal{F}_{\text{new}}(P)$.

Definition 3.2 *An* integrity constraint *of a logic program* P *is a closed formula in* $\mathcal{F}_{\text{new}}(P)$. *An integrity constraint is said to be* static *if it belongs to* $\mathcal{F}(P)$. *Otherwise, it is called a* dynamic *integrity constraint.*

Intuitively, an intensional update is a formula that defines 'new' expressions. It might seem reasonable to restrict intensional updates to sets of implicative formulas like

$$\forall x \quad \text{professor}(x) \land \text{teach}(x, \text{lp}) \Rightarrow \text{new}(\text{qualified-for-tenure}(x))$$

in which the 'new' expression appears only in the consequence. However, a natural expression of some updating intention may require a more general syntax, as shows the classical example of dynamic integrity constraint:

$$\forall x x_1 y \quad \text{salary}(x, y) \land \text{new}(\text{salary}(x_1, y)) \Rightarrow x_1 \geq x$$

In fact, the only necessary restriction is that 'new' must occur in the formulas defining the intensional update, i.e., the formulas must belong to the difference set $\mathcal{F}_{\text{new}}(P) \setminus \mathcal{F}(P)$.

Definition 3.3 *Let* P *be a logic program with set of integrity constraints* IC. *An* intensional update I *of* P *is a logical theory consisting of:*

- *a set of closed formulas* $U \subseteq \mathcal{F}_{\text{new}}(P) \setminus \mathcal{F}(P)$ *called the* definition *of* I
- $\mathcal{T}(P) \cup \{\text{new}(F) \mid F \in IC \cap \mathcal{F}(P)\} \cup \{F \mid F \in IC \setminus \mathcal{F}(P)\}$

The second point of Definition 3.3 ensures that the static integrity constraints, i.e., the elements of $IC \cap \mathcal{F}(P)$, and the dynamic integrity constraints, i.e., the elements of $IC \setminus \mathcal{F}(P)$, are satisfied in the theory defining 'new'. The intensional updates of a given program are uniquely characterized by their definitions. Therefore, we shall sometimes name them after their definitions.

4 Abduction via Deduction

In this section, we formalize the notion of 'realization' of an intensional update in logic. Intuitively, a realization is a set of updating operations, or 'extensional update', that yield a logic program satisfying the updating intention. We consider updating operations consisting of fact insertions and removals. Fact insertion is a simple way to extend the intensional definition of a predicate. Consider for example the following program:

$$\begin{array}{ll} p(x) \leftarrow q(x) \land \neg r(x) & q(a) \\ s(x) \leftarrow q(x) \land u(x) & u(a) \end{array}$$

The intensional update defined by new(p(b)) can be realized by inserting the fact p(b), or by inserting the fact q(b).

In some cases, the first of these insertions is excluded. In databases for example, it is often desired to keep certain relations virtual, i.e., defined only by non-factual clauses. The following definition provides us with a language for expressing dynamic integrity constraints imposing such a condition.

Definition 4.1 *Let* P *be a logic program. An* extensional update *E of P is a set of expressions* 'remove(A)' *or* 'insert(A)' *where* A $\in \mathcal{H}$(P). *An extensional update is* consistent *if it does not contain* 'remove(A)' *and* 'insert(A)' *for a same atom* A. *The logic program obtained by updating* P *according to a consistent extensional update* E *is* $P_E = (P \cup \{A \mid \text{insert}(A) \in E\}) \setminus \{A \mid \text{remove}(A) \in E\}$

It is possible to give a semantics to inconsistent extensional updates by defining priorities between insertions and removals. For example, relying on the definition of P_E given in Definition 4.1, one can interpret $E = \{\text{remove(A), insert(A)}\}$ like {remove(A)}. Alternatively, one could define P_E as $(P \setminus \{A \mid \text{remove}(A) \in E\}) \cup \{A \mid \text{insert}(A) \in E\}$ and interpret E as {insert(A)}. More sophisticated priorities are sometimes considered — e.g., in [29, 28]. Priority relationships between insertions and removals compromise the declarativity of the update language. For this reason, we do not consider inconsistent extensional updates.

Definition 4.1 provides us with a language for expressing dynamic integrity constraints precluding certain modifications of the program. For example, the formula $\forall x\ \neg$insert(p(x)) forbids to extend the definition of a unary predicate p with facts.

In the following definition, we define the 'new' meta-predicate in terms of the extensional update 'insert' and 'remove' meta-predicates. We rely on a 'clause' meta-predicate which is assumed to range over the non-factual clauses of the logic program under consideration. Similarly, we use a 'fact' meta-predicate ranging over the facts of the considered logic program.

Definition 4.2 *Let* P *be a logic program and* I *an intensional update.* $N_{P,I}$ *is the theory consisting of the following formulas, where* A *denotes an atomic formula in* $\mathcal{F}$(P), *and* F *and* G *denotes formulas in* $\mathcal{F}$(P)*:*

$$\begin{array}{llcl}
(1) & \text{new(A)} & \Leftrightarrow & [\exists B\ \text{clause}(A \leftarrow B) \land \text{new}(B)] \\
 & & & \quad \lor [\text{fact}(A) \land \neg\text{remove}(A)] \lor \text{insert}(A) \\
(2) & \text{new}(F \lor G) & \Leftrightarrow & \text{new}(F) \lor \text{new}(G) \\
(3) & \text{new}(F \land G) & \Leftrightarrow & \text{new}(F) \land \text{new}(G) \\
(4) & \text{new}(\forall x\ F) & \Leftrightarrow & \forall x\ \text{new}(F) \\
(5) & \text{new}(\exists x\ F) & \Leftrightarrow & \exists x\ \text{new}(F) \\
(6) & \text{new}(\neg A) & \Leftrightarrow & [\forall B\ \text{clause}(A \leftarrow B) \Rightarrow \text{new}(\neg B)] \\
 & & & \quad \land [\text{fact}(A) \Rightarrow \text{remove}(A)] \land \neg\text{insert}(A)) \\
(7) & \text{new}(\neg(F \lor G)) & \Leftrightarrow & \text{new}(\neg F) \land \text{new}(\neg G) \\
(8) & \text{new}(\neg(F \land G)) & \Leftrightarrow & \text{new}(\neg F) \lor \text{new}(\neg G) \\
(9) & \text{new}(\neg\forall x\ F) & \Leftrightarrow & \exists x\ \text{new}(\neg F) \\
(10) & \text{new}(\neg\exists x\ F) & \Leftrightarrow & \forall x\ \text{new}(\neg F)
\end{array}$$

The following proposition establishes the correctness of Definition 4.2.

Proposition 4.1 *Let* P *be a logic program,* E *an extensional update of* P, *and* F *a formula in* $\mathcal{F}$(P). $P_E \models F$ *if and only if* $(\mathcal{T}(P) \cup E \cup N_{P,I}) \vdash \text{new}(F)$.

Corollary 4.1 *Let* P *be a logic program and* E *an extensional update of* P. $(\mathcal{T}(P) \cup E \cup N_{P,I})$ *is consistent and has exactly one Herbrand model.*

Intuitively, an extensional update E realizes an intensional update I if the formulas defining I are true in the updated program P_E. The following definition formalizes this intuition. We recall that $IC \cap \mathcal{F}(P)$ and $IC \setminus \mathcal{F}(P)$ are respectively the sets of static and dynamic integrity constraints.

Definition 4.3 *Let* P *be a logic program, let* IC *be its set of integrity constraints, and let* I *be an intensional update of* P *defined by a set* U *of formulas. A* realization *of* I *is a consistent extensional update* E *such that:*

$$\begin{array}{lll} (\mathcal{T}(P) \cup E \cup N_{P,I}) \vdash & F & \textit{for all } F \in U \\ (\mathcal{T}(P) \cup E \cup N_{P,I}) \vdash & F & \textit{for all } F \in IC \setminus \mathcal{F}(P) \\ (\mathcal{T}(P) \cup E \cup N_{P,I}) \vdash & \mathrm{new}(F) & \textit{for all } F \in IC \cap \mathcal{F}(P) \end{array}$$

If the intensional update I is defined by a formula new(F), by Proposition 4.1 a realization of I is an extensional update E such that $P_E \models F$. The more complex definition given above is needed since general intensional updates may refer to the logic program prior to the update.

Some realizations can be "subsumed" by others and only "minimal" realizations are of interest. Consider for example the following logic program P:

$$\begin{array}{llll} p(x) \leftarrow r(x) \wedge s(x) & r(a) & s(a) & t(a) \\ q(x) \leftarrow t(x) \wedge u(x) & r(b) & s(b) & u(a) \end{array}$$

The intensional update defined by $\mathrm{new}(\forall x\ p(x) \Rightarrow q(x))$ is realized by the following extensional updates E_1 and E_2. Intuitively, E_1 "subsumes" E_2.

$$\begin{array}{l} E_1 = \{\mathrm{insert}(t(b)), \mathrm{insert}(u(b))\} \\ E_2 = \{\mathrm{insert}(t(b)), \mathrm{insert}(u(b)), \mathrm{insert}(t(c)), \mathrm{insert}(u(c))\} \end{array}$$

Definition 4.4 *Let* P *be a logic program and* I *an intensional update of* P. *An extensional update* E *of* P *is a* minimal realization *of* I *if and only if none of the strict subsets of* E *realize* I. *A procedure for generating the realizations of intensional updates is said to be* complete *if it exhaustively generates all* minimal realizations.

We conclude this section by citing some properties of minimal realizations.

An intensional update may have no, or several minimal realizations. This follows directly from the fact that intensional updates are, in general, non-Horn theories. Some minimal realizations of intensional updates may be infinite, even if the considered logic program is free of function symbols. Finally, some intensional updates may have infinitely many minimal realizations.

The following proposition gives a sufficient condition for finiteness properties of intensional update realizations.

Proposition 4.2 *Let* P *be a logic program and* I *an intensional update of* P. *If* P *and* I *are not recursive, then* I *has finitely many minimal realizations and they are all finite.*

5 Definite Clauses for Disjunctive Reasoning

In this section, we outline the Satchmo theorem prover [22] which implements deduction in disjunctive, indefinite theories by means of definite meta-clauses.

We assume that the formulas are in implicative form. For ground formulas, this form can be derived from the clausal form as follows. A clause $\neg A_1 \vee \ldots \vee \neg A_n \vee C_1 \vee \ldots \vee C_m$ with positive literals C_is is represented by the implication $A_1 \wedge \ldots \wedge A_n \Rightarrow C_1 \vee \ldots \vee C_m$. Completely positive clauses ($n = 0$) and completely negative clauses ($m = 0$) are respectively expressed as true $\Rightarrow C_1 \vee \ldots \vee C_m$ and $A_1 \wedge \ldots \wedge A_n \Rightarrow$ false.

Satchmo computes from a set of implications I the minimal sets of atomic formulas that are logically consistent with I. For example, the set {q $\Rightarrow$ false, true $\Rightarrow$ p, p $\Rightarrow$ q $\vee$ r $\vee$ s} gives rise to derive two such sets of atoms: {p, r} and {p, s}. If there are no such sets, Satchmo fails: This reports inconsistency. Satchmo performs disjunctive and hypothetical reasoning.

The disjunctive and hypothetical reasoning is implemented in Prolog by the following program. Calling 'satchmo([], M)' successively binds the variable M to the minimal models — represented as lists of atoms — of the set of implications defined with the infixed binary predicate '=>'. Conjunctions and disjunctions are expressed à la Prolog with ',' and ';', respectively. The atom 'true' is the Prolog built-in which is always satisfied, while 'false' is an undefined atom: Its evaluation therefore always fails.

```
satchmo(M1, M2) :-
        P => C,
        evaluate(M1, P),
        not evaluate(M1, C), !,
        not (C = false),
        satisfy(A, C),
        satchmo([A | M1], M2).
satchmo(M, M).
```

```
evaluate(M, (A1, A2)) :- !,
        evaluate(M, A1),
        evaluate(M, A2).
evaluate(M, (A1 ; A2)) :- !,
        ( evaluate(M, A1)
        ; evaluate(M, A2) ).
evaluate(M, A) :-
        member(A, M).
evaluate(M, true).
```

```
satisfy(A, (A ; D)).
satisfy(A, (B ; D)) :- !,
        satisfy(A, D).
satisfy(A, A).
```

The procedure 'satchmo' first searches the implications P => C. When an implication is found the body of which is satisfied by the atoms in the list M1 (test 'evaluate(M1, P)'), its head is evaluated over M1 ('evaluate(M1, C)'). If C is not satisfied by M1 and is not the atom 'false', then one of its components A is selected ('satisfy(A, C)') and added to M1. The search for unsatisfied implications is pursued through the recursive call ('satchmo([A | M1], M2)'). As soon as no unsatisfied conclusions can be generated, i.e., as soon as the test

evaluate(M1, P), not evaluate(M1, C)

fails, the second clause for 'satchmo' succeeds and returns as second argument the constructed list of atoms.[1]

This program has been published in [22] in a slightly different style. It is the basic program of the theorem prover Satchmo. It is applicable to proving theorems in non-ground theories, provided their formulas are expressed as Skolemized and range-restricted implications [22]. Range-restriction can be achieved with any set of formulas by introducing a new predicate 'dom' describing the domain of the already constructed set of atomic formulas, and by adding auxiliary implications [22].

Skolemization is acceptable in refutation theorem proving because a theory T is inconsistent if and only if a Skolemized form Sk(T) of T is inconsistent. However, Skolemization may have an undesirable side-effect if Satchmo is used for generating the minimal sets of atomic formulas that are logically compatible with a consistent theory. Consider for example the following logic program, the intensional update new(p(a, b)), and the constraint $\neg$ insert(p(a, b)):

$$\mathrm{p}(x, y) \leftarrow \mathrm{q}(y) \wedge \mathrm{p}(x, z) \qquad\qquad \mathrm{q(b)}$$

By Definition 4.2, we have: $\mathrm{new}(\mathrm{p}(x, y)) \Rightarrow \exists z\ \mathrm{new}(\mathrm{q}(y)) \wedge \mathrm{new}(\mathrm{p}(x, z))$. If the existential variable z is represented the Skolem function f(x, y), new(p(a, b)) implies new(p(a, f(a, b)) which in turn implies new(p(a, f(a, f(a,b))), etc. No finite realizations are found. Such realizations however exist, e.g., {p(a, a)}.

We first define the implicative form for formulas with explicit existential quantifications. Then, we give a version of the procedure Satchmo for such implications.

Definition 5.1 *A formula* F *is in* implicative form *if* $\mathrm{F} = \mathrm{A}_1 \wedge \ldots \wedge \mathrm{A}_n \Rightarrow \mathrm{C}_1 \vee \ldots \vee \mathrm{C}_n$ *where the* A_i*s are atoms (possibly* 'true' *if* n = 1*) or existential formulas in conjunctive form, and where the* C_j*s are atoms (possibly* 'false' *if* m = 1*) or existential formulas in conjunctive form. An existential formula* G *is in* conjunctive form *if* $\mathrm{G} = \exists x\ \mathrm{A}_1 \wedge \ldots \wedge \mathrm{A}_n$ *or* $\mathrm{G} = \exists x\ \mathrm{A}_1 \wedge \ldots \wedge \mathrm{A}_n \wedge \mathrm{F}$ *where the* A_i*s are atoms or existential formulas in conjunctive form containing the variable* x *and where* F *is a formula in implicative form.*

One can show that syntactical transformations permit to rewrite any first-order theory as a set of formulas in implicative form.

A procedure constructing all minimal sets of atomic formulas consistent with a set of formulas in implicative form is obtained by first trying to instantiate the existential variables over the already constructed domain and with a new domain value. So does the 'satchmo_1' procedure given below. In contrast with 'satchmo', 'satchmo_1' stores the formulas in a list which is dynamically modified.

Calling 'satchmo_1(S, [], [], M)' successively binds the variable M to the finite models — represented as lists of atoms — of a list S of range-restricted formulas in implicative form. We assume that calling 'new_value(X)' instantiate X with a term which is not in the current domain.

In [22], we have given a leveled version of Satchmo which achieves completeness with respect to inconsistency: If a set of Skolemized formulas in implicative form is inconsistent 'leveled Satchmo' will report it in finite, indefinite time. Although it is complete with respect to inconsistency, 'leveled Satchmo' may fail to construct the minimal sets of atoms consistent with a recursive set of implications. This is because, in presence of Skolem functions, a consistent set of formulas has always infinite Herbrand models (Skolem-Löwenheim Theorem). In such a case, 'leveled Satchmo' can very well start the construction of an infinite model before building finite models.

As shown in [7], an exhaustive procedure for constructing minimal models — i.e., a procedure generating each minimal, finite model in finite but indefinite time — requires to abandon the depth-first search strategy of Prolog for a breath-first strategy.

Proposition 5.1 *When evaluated under a breadth-first strategy,* 'satchmo_1' *is a sound and exhaustive procedure for generating minimal, finite models.*

```
satchmo_1(S1, S3, M1, M3) :-
    member(P => C, S1),
    evaluate(M1, P),
    not evaluate(M1, C), !,
    not (C = false),
    satisfy(S1, S2, M1, M2, C),
    satchmo_1(S2, S3, M2, M3).
satchmo_1(S1, S3, M1, M3) :-
    member(exists(X, F), S1),
    not evaluate(M1, F),
    satisfy(S1, S2, M1, M2, exists(X, F)),
    satchmo_1(S2, S3, M2, M3).
satchmo_1(S, S, M, M).
```

```
evaluate(M, exists(X, F)) :- !,
    evaluate(M, F).
evaluate(M, (A1, A2)) :- !,
    evaluate(M, A1),
    evaluate(M, A2).
evaluate(M, (A1 ; A2)) :- !,
    ( evaluate(M, A1)
    ; evaluate(M, A2) ).
evaluate(M, A) :-
    member(A, M).
evaluate(M, true).
```

```
satisfy(S, [F | S], M, M, exists(X, F)) :-
    member(dom(X), M).
satisfy(S, [F | S], M, [dom(X) | M], exists(X, F)) :- !,
    new_value(X).
satisfy(S, S, M, [A | M], (A ; D)).
satisfy(S, S, M1, M2, (B ; D)) :- !,
    satisfy(S, S, M1, M2, D).
satisfy(S, S, M, [A, | M], A).
```

6 Intensional Update Realization

In this section, we propose an implementation of Definition 4.2 by means of formulas in implicative form to be processed by 'satchmo_1'. We rely on the meta-predicates 'ground_atom', 'base_atom', and 'satisfied'. A call 'ground_atom(A)' succeeds when A is an ground atom. 'base_atom(A)' holds if A is a ground atom the literal of which does not appear in the head of any rule. 'satisfied' refers to a meta-interpreter which, given a logic program P and and an extensional update M, simulates evaluation against the updated program P_M. We do not give these programs here, since they are rather simple (an implementation of 'satisfied' is given under the name 'new' in [6]). We assume moreover that the procedure 'evaluate' is extended with the clause evaluate(M, F) :- satisfied(M, F).

For the sake of simplicity and without loss of generality, we assume that the existential quantifications are explicit in the bodies of rules in the object logic program: If a variable x does occur only in the body of a program clause, then this body must have the form $\exists x$ F.

```
new((F, G)), not (F, G) => new(F).
new((F, G)), not (F, G) => new(G).
new((F ; G)), not (F ; G) => new(F) ; new(G).
new(forall(X, R => F), R, not F => new(not R) ; new(F).
new(exists(X, F)), not F => exists(X, new(F)).
new(A), base_atom(A), not A => insert(A).
new(A), ground_atom(A),
        not A, clause(A, B) => insert(A) ; new(B).
new(not (F, G)) => new((not F ; not G)).
new(not (F ; G)) => new((not F, not G)).
new(not forall(X, R => F)) => new(exists(X, (R, not F))).
new(not exists(X, (R, F)) => new(forall(X, R => not F)).
new(not A), ground_atom(A), A => remove(A).
new(not A), ground_atom(A),
        clause(A, B), not (B = true), B => new(not B).
insert(A), remove(A) => false.
new(false) => false.
```

The procedure 'satchmo_1' must be constrained such that only 'insert', 'remove', and 'new' atoms are collected in the list representing the set of atoms under construction. It is also necessary to restrict the instantiation of existential variables over new constants to those variables that occur in 'new' expressions. The first restriction is enforced by an additional test, the second by distinguishing two kinds of existential quantifications, depending whether the variable occurs in a 'new' atom or not. We do not give the program specialized in this way: It follows easily from 'satchmo_1'.

Since the implications given above implement Definition 5.1, we have by Proposition 4.2:

Corollary 6.1 *The finite minimal realizations of an intensional update to a logic program are exhaustively generated when the above-defined implications are processed with (specialized)* 'satchmo_1' *under a breadth-first evaluation strategy.*

The meta-interpreter 'satisfied' reflects the evaluations of the underlying system, and therefore conveys its restrictions. If the object rules are recursive, Prolog cannot be used because the evaluation may never stop. If the object logic program contains negative premise literals and is not hierarchical, a Prolog-like evaluation may be incorrect. However, it suffices to consider, in place of Prolog, a top-down reasoning method that correctly handles recursive or non-hierarchical programs for obtaining a correct evaluation of 'satisfied' expressions and hence a correct generation of intensional update realizations — such procedures are described, e.g., in [4, 3].

We conclude with an example. The reasoning is illustrated on the figure below. The branchings of this figure express alternative hypotheses that are made during disjunction processing.

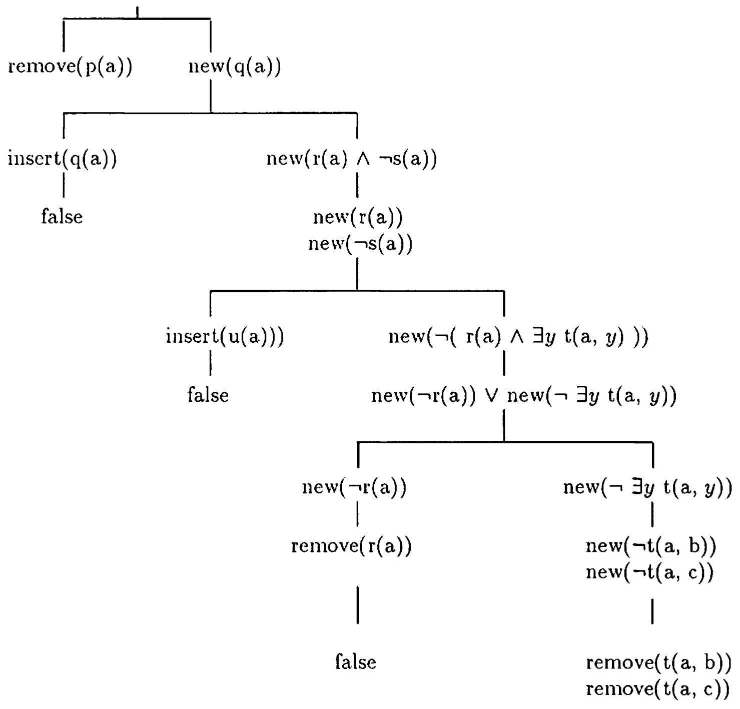
Clauses and Facts:			
$q(x) \leftarrow r(x) \wedge \neg s(x)$	p(a)	r(a)	t(a, b)
$s(x) \leftarrow r(x) \wedge \exists y\ t(x, y) \wedge \neg u(x)$		r(b)	t(a, c)

Intensional Update:	Integrity Constraints:
$\forall x\ p(x) \Rightarrow \text{new}(q(x))$	$\forall x\ \text{insert}(q(x)) \Rightarrow \text{false}$
	$\forall x\ r(x) \wedge \text{insert}(u(x)) \Rightarrow \text{false}$

7 Conclusion

We have investigated an approach aiming to free Logic Programming from procedural updates. We have proposed the concept of 'intensional update' for specifying an update without defining its procedural execution. Thus, intensional updates depart in their principle from the Prolog side-effect 'assert' and 'retract' predicates, and from the update primitives proposed in [1, 2] for bottom-up Logic Programming.

Intensional updates generalize database 'view updates'. While view updates permit one to express ground, atomic updating intentions, intensional updates are more general. They give rise to express updating intentions involving quantifiers. Intensional updates also give rise to relating the updated logic program to the version prior to the update. Therefore, they are more general than the updates considered in [17, 11, 25, 19].



In order to be effective, intensional updates must be translated into changes of the logic program, or 'extensional updates'. In this study, we have considered extensional updates consisting of fact insertions and removals, although the same framework also gives rise to handling 'exceptions' to rules and clause deletions. We have investigated the reasoning involved in generating minimal extensional updates that realize an intensional update. This reasoning is not deductive, but abductive.

We have formalized abduction as deduction in a meta-theory. Then, we have applied a specialized version of the theorem prover Satchmo [22] to process the disjunctive formulas of this formalization. Relying on results we established for

Satchmo [7], we have established the completeness of the abductive intensional update realization method.

This study is related to the large amount of articles on database view updates, that we cannot all cite here. The generation of realizations for view updates has been formalized in [14]. This article does however not describe any implementation. [20] proposes a declarative expression of sequential updates and investigates the hypothetical reasoning they require, which is close to that performed by the meta-interpreter mentioned in Section 6 and [6]. The article [31] investigates the control of concurrent database updates by considering updating intentions instead of procedural definitions.

The methods described in [17, 11, 25, 19] also consider intensional updates. However, they do not handle dynamic integrity constraints, nor do [17, 25, 19] address completeness issues. [17, 11] propose to handle static integrity constraints according to a 'generate-and-test' scheme: Realizations are first generated without considering the constraints. Then, those violating some of the constraints are rejected. In contrast, the method we proposed interupts as soon as possible the construction of constraint violating realizations.

There are interesting links between the method we proposed and logic-based diagnosis. On the one hand, the concept of intensional update realization is extremely close to that of diagnosis: Invalidating a literal is in fact similar to assuming a disease or a malfunction. On the other hand, Satchmo can be used as a — in some respect, improved — truth maintenance system and can therefore serve for consistency-based diagnosis as it is shown in [16][2]. In [26] it is argued that the two diagnosis paradigms — consistency-based and abduction — are "fundamentally different" and require "different ways to think about a domain". However, we have shown that the reasoning involved in both paradigms can be performed with the same inference engine, namely Satchmo. There are nevertheless slight differences: They are expressed through the meta-logical implications of Section 6.

Further research should be devoted to several issues. First, it would be desirable to investigate if efficiency could be improved through other implementations of the same principle, in particular for data intensive applications. Second, the two-state language we have proposed for expressing intensional updates and dynamic constraints could be extended into a n-state language for applications with historical data. We think that the formal treatment of intensional updates given in this article should provide one with the appropriate formal basis for such an extension. Finally, it would be desirable to investigate how preferences between realizations can be declaratively and elegantly expressed. A proposal in this direction has been made in [29].

Acknowledgements

This study was partly supported by the European Community in the framework of the ESPRIT Basic Research Action *Compulog* No. 3012. I would like to thank my colleagues of *Compulog* for helpful discussions. I am grateful to Jesper Träff and Mark Wallace who helped me finding some mistakes in an early version of this paper.

Notes

^{1}: In Section 6, 'evaluate' is applied on quantified formulas. A variance test must replace unification in 'member', for preventing undesirable bindings such as X:b and Y:a from 'forall(X, p(X, a) => q(X, a))' and 'forall(Y, p(b, Y) => q(b, Y))'.

^{2}: The program called Momo in [16] is identical with Satchmo up to unsubstantial changes, and not only based on a basic principle described in [22] as it is claimed in [16].

References

[1] S. Abiteboul and V. Vianu. A Transaction Language Complete for Update and Specification. In *Proc. 6th ACM Symp. on Principles of Database Systems (PODS)*, 1987.

[2] S. Abiteboul and V. Vianu. Procedural and Declarative Database Update Languages. In *Proc. 7th ACM Symp. on Principles of Database Systems (PODS)*, 1988.

[3] F. Bry. Logic Programming as Constructivism: A Formalization and its Application to Databases. In *Proc. 8th ACM Symp. on Principles of Database Systems (PODS)*, 1989.

[4] F. Bry. Query Evaluation in Recursive Databases: Bottom-up and Top-down Reconciled. In *Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases (DOOD)*, 1989.

[5] F. Bry. Towards an Efficient Evaluation of General Queries: Quantifier and Disjunction Processing Revisited. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD)*, 1989.

[6] F. Bry, H. Decker, and R. Manthey. A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases. In *Proc. 1st Int. Conf. Extending Data Base Technology (EDBT)*, 1988.

[7] F. Bry and R. Manthey. Proving Finite Satisfiability of Deductive Databases. In *Proc. 1st Workshop on Computer Science Logic (CSL)*, 1987.

[8] E. Charniak. A Neat Theory of Marker Passing. In *Proc. 5th Nat. Conf. on Artificial Intelligence (AAAI)*, 1986.

[9] W. Chen and D. S. Warren. Abductive Logic Programming. Research Report, Dept. of Comp. Sc., State Univ. of New York at Stony Brook, 1989.

[10] C. C. Cosmadakis and C. H. Papadimitriou. Updates of Relational Views. *Journal of the ACM*, 31(4):742–760, 1984.

[11] H. Decker. Drawing Updates from Derivations. Research Report IR-KB-65, ECRC, 1989.

[12] K. Eshgi and R. A. Kowalski. Abduction through Deduction. Research Report, Dept. of Computing, Imperial College od Sc. and Tech., London, 1988.

[13] K. Eshgi and R. A. Kowalski. Abduction Compared with Negation by Failure. In *Proc. 6th Int. Conf. on Logic Programming (ICLP)*, 1989.

[14] R. Fagin, J. Ullman, and M. Vardi. On the Semantics of Updates in Databases. In *Proc. 2nd ACM Symp. Principles of Database Systems (PODS)*, 1983.

[15] J. J. Finger. *Exploiting Constraints in Design Synthesis.* PhD thesis, Dept. of Computer Science, Stanford Univ., 1987.

[16] G. Friedrich and W. Nejdl. MOMO - Model-Based Diagnosis for Everybody. In *Proc. 6th IEEE Conf. on Artificial Intelligence Applications*, 1990.

[17] A. Guessoum and J. W. Lloyd. Updating Knowledge Bases. Research Report TR-89-05, Comp. Sc. Dept., Univ. of Bristol, 1989.

[18] J. R. Hobbs and M. E. Stickel. Interpretation as Abduction. In *Proc. 26th Annual Meeting of the Assoc. for Computational Linguistic*, 1988.

[19] T. Kakas and P. Mancarella. Database Updates through Abduction. Research Report, Dept. of Computing, Imperial College od Sc. and Tech., London, 1990.

[20] S. Manchanda. Declarative Expression of Deductive Database Updates. In *Proc. 8th Symp. on Principles of Database Systems (PODS)*, 1989.

[21] S. Manchanda and D. S. Warren. Towards a Logical Theory of Database View Updates. In *Proc. Workshop on Foundations of Deductive Databases and Logic Programming*, 1986.

[22] R. Manthey and F. Bry. SATCHMO: A Theorem Prover Implemented in Prolog. In *Proc. 9th Int. Conf. on Automated Deduction (CADE)*, 1988.

[23] P. Norvig. Inference in Text Understanding. In *Proc. 6th Nat. Conf. on Artificial Intelligence (AAAI)*, 1987.

[24] C. S. Peirce. *Collected Papers of Charles Sanders Peirce*, volume 2, 1931-1958. Harvard University Press, 1958. pages 272-607.

[25] L. M. Pereira, M. Calejo, and J. N. Aparício. Refining Knowledge Base Updates. Research Report, AI Center, Univ. of Lisbon, 1989.

[26] D. Poole. Normality and Faults in Logic-Based Diagnosis. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 1989.

[27] H. J. Pople, Jr. On the Mechanization of Abductive Logic. In *Proc. 3rd Int. Conf. on Artificial Intelligence*, 1973.

[28] F. Rossi and S. A. Naqvi. Contributions to the View Update Problem. In *Proc. 6th Int. Conf. on Logic Programming (ICLP)*, 1989.

[29] A. Tomasic. View Update: Translation via Deduction and Annotation. In *Proc. 2nd Int. Conf. on Database Theory (ICDT)*, 1988.

[30] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. In *Proc. 7th ACM Symp. on Principles of Database Systems (PODS)*, 1988.

[31] V. Vianu and G. Vossen. Goal-Oriented Concurrency Control. In *Proc. 2nd Symp. on Mathematical Fundamentals of Database Systems (MFDBS)*, 1989.