Declarative Updates in Deductive Object Bases

Mengchi Liu

Department of Math & Computer Science
University of Prince Edward Island
Charlottetown, Prince Edward Island
Canada C1A 4P3
MLIU@upei.ca

John Cleary

Department of Computer Science
University of Calgary
Calgary, Alberta
Canada T2N 1N4
cleary@cpsc.ucalgary.ca

Abstract

Updates are important database operations, but there has not yet been a uniform logical framework that accounts for them. This paper presents an approach to deal with updates in the deductive and object-oriented database setting. It incorporates temporal information into update rules, so that different versions of objects may be created at different time points and can be identified by the temporal information. The proposed update language has a simple and clear Herbrand-like declarative semantics, which can be computed by a bottom-up evaluation using a temporal stratification.

1 Introduction

Deductive and object-oriented databases are two important extensions of traditional database technology. Deductive databases extend the expressive power of traditional databases by means of deduction and recursion. Object-oriented databases extend the modelling power of the traditional databases by using concepts such as object identity, complex objects, classes, and inheritance. The integration of deductive and object-oriented databases has received considerable attention over the past few years and several deductive and object-oriented database languages have been proposed [AK89, Abi90, AG91, Mai86, CW89, KW89, KL89, KLW90].

The theory of deductive databases without updates is well established and a declarative semantics is characterized as one of its most important features. Similar declarative semantics have also been given object-oriented database languages without updates.

Updates are important database operations, how to incorporate them into deduction has been the focus of active research during the last few years and various approaches have been proposed [Abi88, Bry90, dMS88, KM90, KLS92, Man89, NK88]. However, only a few of them take object-orientation into account. Until now, there has been no uniform logical framework that accounts for database updates. The major difficulty is that updates require control features that deviate from a pure declarative semantics.

Because of this, several deductive database languages, including DLP [Man89] and LDL [NT89], directly provide explicit procedural update constructs and resort to dynamic logic to give procedural semantics to the update part of the languages.

Recently, Kramer et al. [KLS92] presented an update language for deductive and object-oriented databases based on object versioning. For a version v of an object, an update on v creates a new version of the object represented by $ins(v), mod(v)$, or $del(v)$ depending on the type of updates (insert, modify, or delete). Every version corresponds to a certain time step of the entire update process. The version of an object therefore represents the history of the updates on the object. For example, the version $ins(del(mod(ins(o))))$ indicates that the object o first experienced insertion, then modification, deletion and finally insertion, and this version may then be used by other parts of the update program to generate new versions. At the end of the program, the last version of an object represents the final updated objects. In this way, the user exerts explicit control over the update process and update programs have Herbrand-like declarative semantics. There are two major problems with this approach. First, it is the user's responsibility to make sure they refer to the proper versions in each update time step. Second versions of the same base object may be created that are not linearly ordered with each other.

In this paper, we present an approach which directly uses temporal information to identify object versions. There are several advantages of using temporal information. First, time points are naturally linearized. Second, updates are naturally associated with times. Using temporal information to identify object versions is easier for the user than using the method proposed by Kramer et al. Most importantly, the proposed update language has a simple and clear declarative semantics. A similar proposal is advanced in [LC93] for pure deductive databases.

The update mechanism uses "updates in heads", in contrast to "updates in bodies" [Abi88]. That is, a positive literal in the head of the rule is interpreted as insertion or modification and a negative one as a deletion. Updates in heads were used in [Mai86, dMS88, CCCR+90] for snapshot databases, but explicit control is used in [dMS88], manual control is used in [CCCR+90], and a well-defined semantics is not provided in [Mai86].

The importance of incorporating temporal information into databases has long been recognized. Many techniques for modelling and managing temporal databases have been introduced (see [MS91] for a survey). Most of them are based on the relational data model. There are also a few, such as OSAM*/T[SC91], which take object-orientation and deduction into account, but well-defined logic-based semantics is still lacking. The work presented here provides a clear logical account for such a temporal database system. It can also be effectively used for snapshot databases.

The paper is organized as follows. Section 2 introduces the syntax of the update language. Section 3 gives several motivating examples. Section 4 contains the semantics. Section 5 describes the bottom-up computation. Section 6 gives a brief concluding discussion on how the update language can be used for traditional snapshot databases.

2 Syntax of the Update Language for Objects

The alphabet of the update language contains the following infinite and pairwise disjoint sets of symbols:

(1) a set $\mathcal{O}$ of object identifiers,

(2) a set $\mathcal{V}$ of variables

(3) a set $\mathcal{C}$ of classes,

(4) a set $\mathcal{A}$ of attributes,

(5) a set $\mathcal{T}$ of time points.

Object identifiers are used to denote objects and are grouped together into classes. Attributes are functions to express properties of objects of classes. The value of an attribute of an object is an object. For formal simplicity, we consider values as specific object identifiers in $\mathcal{O}$. The set $\mathcal{T}$ is interpreted as a linear set, that is, there is a least element and for every pair of distinct elements t_i and t_j in $\mathcal{T}$, either $t_i > t_j$ or $t_i < t_j$. Without losing generality, we assume t_0 to be the least element of $\mathcal{T}$. In the following examples, the set $\mathcal{T}$ is taken to be the set of positive integers.

Similar to Kramer et al's approach, we distinguish extensional classes and attributes from intensional classes and attributes. Only extensional classes and attributes can be updated.

There are two kinds of object terms: normal object terms which do not include time points and temporal object terms which do.

Let $O, O_1, ..., O_n$ be variables or object identifiers, p a class, and $a_1, ..., a_n$ attributes, then $O : p(a_1 \rightarrow O_1, ..., a_n \rightarrow O_n)$ is a normal object term, where $n \geq 0$.

Let $O, O_1, ..., O_n$ be variables or object identifiers, t a time point, p a class, and $a_1, ..., a_n$ attributes, then $O : p(a_1 \rightarrow O_1, ..., a_n \rightarrow O_n)@t$ is a temporal object term, where $n \geq 0$.

An arithmetic comparison expression is an expression using $+$, $-$, $*$, $/$, $<$, $>$, $=$, etc. in the standard way.

Corresponding to the two kinds of object terms, we have two kinds of rules: normal rules and update rules.

A normal rule is an expression of the form $A \Leftarrow L_1, ..., L_n$, $n \geq 1$, where the head A is a normal object term, and the body L_1,...,L_n is a conjunction of normal object terms, negated normal object terms, or arithmetic comparison expressions.

An update rule is of the form $T \Leftarrow T_1, ..., T_n$, $n \geq 0$, where the head T is a temporal object term, or a negated temporal object term, and the body T_1,...,T_n is a conjunction of temporal object terms, negated temporal object terms, or arithmetic comparison expressions.

If the head of an update rule is positive (not negated), then it is either an insertion rule used to insert objects into an extensional class or insert values of extensional attributes of an object, or a modification rule used to modify values of extensional attributes. If the head is negative, then it is a deletion rule which is used to delete either the values of extensional attributes of objects or delete objects from their classes. An update fact is an update rule with an empty body. For syntactic clarity positive (temporal) terms in the head of an update rule are indicated with a + and negative ones with a −.

As usual, we require that rules be safe in the sense that all variables which occur in the head also occur in the body [Ull88].

A program P consists of two sets $P = \langle R_N, R_U \rangle$, where R_N is the set of normal rules and R_U the set of update rules.

A query has the form ?– $T_1, ..., T_n$, where T_1,...,T_n are temporal object terms, negated temporal object terms, or arithmetic comparison expressions.

The language introduced so far can be considered as a restricted form of first order logic augmented with temporal information. Classes correspond to unary predicates and attributes to binary predicates. For reasons of simplicity, we treat all attributes as multi-valued so that we do not have to consider consistency problems with respect to functionality of attributes [AH88]. Also, we do not consider object creation, schema and inheritance they are the topic of a separate paper [Liu92].

3 Illustrative Examples

Before giving formal semantics, we present several examples in this section. First, let us look at several update facts. At time 1, insert an object *tom*, into the extensional class *employee*, and assign 3000 and *shoe* to the extensional attributes *salary* and *works_in* respectively.

$$+tom : employee(salary \rightarrow 3000, works\_in \rightarrow shoe)@1.$$

At time 3, modify the value of the extensional attribute *works_in* of the object *tom*, from *shoe* to *toy*.

$$+tom : employee(works\_in \rightarrow toy)@3.$$

At time 4, delete the value of the extensional attribute *salary* of the object *tom*

$$-tom : employee(salary \rightarrow 3000)@4.$$

Delete the object *tom*, at the time point 5, from the extensional class *employee*

$$-tom : employee@5.$$

Notice that the deletion of the object *tom* from the extensional class *employee* also deletes all values of the attributes defined on the class *employee*, such as *salary*.

Now let us look at several update rules. At some time point T, give each employee a 10% salary increase and those in a managerial position an extra 200 [1]. The new values in the database at time T are calculated from the values existing in the database at the preceeding time T_0. The object $salary : update$ indicates the time at which the update should occur and is injected either by some other rule or externally by a user interface.

$$\begin{aligned} +E : employee(salary \rightarrow S_2)@T \Leftarrow\ & salary : update@T, \\ & T = T_0 + 1, \\ & E : employee(salary \rightarrow S_1, works\_in \rightarrow D)@T_0, \\ & D : dept(manager \rightarrow E)@T_0, \\ & S_2 = S_1 * 1.1 + 200. \end{aligned}$$

$$\begin{aligned} +E : employee(salary \rightarrow S_2)@T \Leftarrow\ & salary : update@T, \\ & T = T_0 + 1, \\ & E : employee(salary \rightarrow S_1, works\_in \rightarrow D)@T_0, \\ & \neg D : dept(manager \rightarrow E)@T_0, \\ & S_2 = S_1 * 1.1. \end{aligned}$$

Afterwards (one time point later) all employees who make more than their bosses are fired.

$$\begin{aligned} -E : employee@T \Leftarrow\ & salary : update@T_0, \\ & T = T_0 + 1, \\ & E : employee(boss \rightarrow B, salary \rightarrow S_1)@T_0, \\ & B : employee(salary \rightarrow S_2)@T_0, \end{aligned}$$

[1] This example is from [KLS92]

$S_1 > S_2.$

The following are two normal rules which are used to define the attribute *boss* of the class *employee*, the class *highPaidEmpl*.

$$
\begin{aligned}
E : employee(boss \rightarrow B) \Leftarrow\ & E : employee(works\_in \rightarrow D), \\
& D : dept(manager \rightarrow B), \\
& B \neq E.
\end{aligned}
$$

$$
\begin{aligned}
E : highPaidEmpl \Leftarrow\ & E : employee(salary \rightarrow S), \\
& S \geq 3000.
\end{aligned}
$$

Based on the rules above, we add the following update facts to make them a complete program which will be used as a running example through out this paper.

$+toy : dept(manager \rightarrow henry)@1.$

$+henry : employee(salary \rightarrow 2800, works\_in \rightarrow toy)@1.$

$+tom : employee(salary \rightarrow 3000, works\_in \rightarrow toy)@3.$

$+salary : update@4.$

4 Semantics

Let P be a program. As in traditional logic programming, we are interested in Herbrand-like interpretations.

Let P be a program. The *Base* B_P of P is the set of all possible ground temporal object terms formed from the class symbols and attribute symbols in P, time points in $\mathcal{T}$ and object identifiers in $\mathcal{O}$ together with all such terms preceeded by $+$ and $-$. So, for the example program all of $tom : employee@1$, $+tom : employee@1$ and $-tom : employee@1$ are part of the base. We will say that a term such as $+tom : employee@1$ is true (w.r.t. the interpretation I) iff it is a member of I.

A (Herbrand) interpretation I of a program P is a subset of the base B_P. The set $I[t]$ is the set I restricted to the time point t. More precisely: $I[t] = \{B : B@t \in I\}$. This can be treated as an interpretation of the normal rules at the time t.

We treat complex object descriptors as follows: if $o : p(a_1 \to o_1, ..., a_n \to o_n)@t \in I$ where $n \geq 1$, then it is synonymous with $o : p@t \in I$ and $o : p(a_j \to o_j)@t \in I$, $1 \leq j \leq n$. In other words, $o : p(a_1 \to o_1, ..., a_n \to o_n)@t$ stands for the conjunction of $o : p@t$, $o : p(a_1 \to o_1)@t, ..., o : p(a_n \to o_n)@t$. The truth value of arithmetic comparison expressions are defined in the standard way.

The truth of normal rules is defined in the usual way as follows. A normal rule is true iff all ground instances of the rule $A \Leftarrow L_1, ..., L_n$ are true. That is, if all ground temporal instances $L_1@T, ..., L_n@T$ are true then $A@T$ is true.

The following two constraints on models give the semantics of updates. The positive update constraint holds iff for every ground term $A@t$ whenever $+A@t$ is true and $-A@t$ is false then $A@t$ is true. The frame update constraint holds iff for every ground term $A@t$ whenever $-A@t$ is false and there exists $t_1 < t$ such that for all t_2 where $t_1 \leq t_2 < t$, $A@t_2$ is true then $A@t$ is true. (The frame update constraint can be simplified in the case that $\mathcal{T}$ is the positive integers to: the frame update constraint holds iff for every ground term $A@t$ whenever $-A@t$ is false and $A@(t-1)$ is true then $A@t$ is true.)

Let P be a program. An interpretations I is a *model* of P iff every rule in P is true and the two update constraints hold.

It is easily verified that for a model M each of the projections $M[t]$ is a model (in the usual sense) of the normal rules in the database.

The update rules are worthy of some comment. In standard cases they conform to intuitions about a database. If $+A@t$ is true then $A@t$ is to be added to the database. If neither $-A@t$ nor $+A@t$ is true then the truth or falsity of $A@t$ remains unchanged. In the case when $A@t$ is false then $-A@t$ leaves the database undisturbed. Because we will seek minimal models as the preferred ones a deletion $-A@t$ functions by removing the necessity for $A@t$ to be true. The singular case when both $+A@t$ and $-A@t$ are true has been resolved arbitrarily to delete $A@t$.

It is possible to state these update constraints as if they were encoded in a constructive rule (this constructive form will be used in the next section for a monotone mapping which leads to a least fixpoint which is a minimal model): $A@t$ must be true if there is no deletion $-A@t_1$ since the last addition $+A@t_0$. To be more precise: $A@t$ must be true if there is some $+A@t_0, t \geq t_0$ and no other additions since $(+A@t_1, t \geq t_1 > t_0)$ and no other deletions at the same time or since $(-A@t_2, t \geq t_2 \geq t_0)$. In the language described in [LC93] it is possible to write such update rules directly in the language.

The following interpretation M broken into the sequence of point interpretations $M[0]$, ..., $M[6]$, ... can be verified as a model of the example program in the last section.

$M[0] = \{\}$

$M[1] = \{+henry : employee(salary \rightarrow 2800, works\_in \rightarrow toy),$
$\quad henry : employee(salary \rightarrow 2800, works\_in \rightarrow toy),$
$\quad +toy : dept(manager \rightarrow henry),$
$\quad toy : dept(manager \rightarrow henry)\}$

$M[2] = \{henry : employee(salary \rightarrow 2800, works\_in \rightarrow toy),$
$\quad toy : dept(manager \rightarrow henry)\}$

$M[3] = \{henry : employee(salary \rightarrow 2800, works\_in \rightarrow toy),$
$\quad +tom : employee(salary \rightarrow 3000, works\_in \rightarrow toy, boss \rightarrow henry),$
$\quad tom : employee(salary \rightarrow 3000, works\_in \rightarrow toy, boss \rightarrow henry),$

$$
\begin{aligned}
&+salary : update,\\
&salary : update,\\
&toy : dept(manager \rightarrow henry),\\
&tom : highPaidEmpl\}
\end{aligned}
$$

$$
\begin{aligned}
M[4] = \{&+tom : employee(salary \rightarrow 3300),\\
&tom : employee(salary \rightarrow 3300, works\_in \rightarrow toy, boss \rightarrow henry),\\
&+henry : employee(salary \rightarrow 3280),\\
&henry : employee(salary \rightarrow 3280, works\_in \rightarrow toy),\\
&toy : dept(manager \rightarrow henry),\\
&tom : highPaidEmpl, henry : highPaidEmpl\}
\end{aligned}
$$

$$
\begin{aligned}
M[5] = \{&henry : employee(salary \rightarrow 3280, works\_in \rightarrow toy),\\
&-tom : employee,\\
&toy : dept(manager \rightarrow henry),\ henry : highPaidEmpl\}
\end{aligned}
$$

$$
\begin{aligned}
M[6] = \{&henry : employee(salary \rightarrow 3280, works\_in \rightarrow toy),\\
&toy : dept(manager \rightarrow henry),\ henry : highPaidEmpl\}
\end{aligned}
$$

A program P may have an infinite numbers of models. By making proper restrictions on the program similar to those in traditional logic programs, we can guarantee that the program has a model. One distinguished minimal and supported model can be chosen as the intended semantics of the program. We discuss this in the next section.

5 Bottom-Up Computation

The computation of a model of a logic program is usually done bottom-up by repeatedly applying a monotone mapping until a least fixpoint is reached. In the presence of negation, and in our case additions and deletions, it is not always possible to construct such a mapping. A solution to this problems can be achieved by constructing a local stratification on the program [Prz88]. The aim of such stratifications is to partition the base into strata; bottom-up computation then is done stratum by stratum. The results of lower strata are the input to the respective next higher stratum. After having processed all strata, a fixpoint of the program is reached.

First we define a mapping T_P on the program which we will show under an appropriate local stratification will give a least fixpoint. The mapping is the same as the one traditionally given for logic programs plus extra terms to account for the update rules.

$$
\begin{aligned}
T_P(I) = \{A@t : {} & A@t \Leftarrow L_1@t, ...L_n@t \text{ is a ground instance of a normal rule} \\
& \text{and each } L_i@t,\ 1 \leq i \leq n, \text{ is true w.r.t. } I\ \} \cup \\
\{F@t : {} & F@t \Leftarrow L_1@t_1, ...L_n@t_n \text{ is a ground instance of an update rule} \\
& \text{and each } L_i@t_i,\ 1 \leq i \leq n, \text{ is true w.r.t. } I\ \} \cup \\
\{A@t : {} & A@t \text{ is a ground term and} \\
& \text{there is some } +A@t_0 \text{ true w.r.t. to } I,\ t \geq t_0, \text{ and} \\
& \text{there is no term } +A@t_1 \text{ true w.r.t. } I,\ t \geq t_1 > t_0, \text{ and} \\
& \text{there is no term } -A@t_2 \text{ true w.r.t. } I,\ t \geq t_2 \geq t_0\ \}.
\end{aligned}
$$

It is easily verified that any fixpoint of T_P is a model in the sense defined above.

Given this mapping we now wish to construct a local stratification to ensure that a suitable monotonic sequence is available to arrive at a least fixpoint. A local stratification is defined here as a countable sequence of disjoint subsets of B_P, $H_0, H_1, ...$ which satisfy the following conditions:

1. For each ground instance of a normal rule $A \Leftarrow B_1, ...B_n, \neg C_1, ..., \neg C_m$.

   $A@t \in H_l$ implies that

   a) for each i, $1 \leq i \leq n$ that $B_i@t \in H_k$ for some $k \leq l$

   b) for each i, $1 \leq i \leq m$ that $C_i@t \in H_k$ for some $k < l$

2. For each ground instance of an update rule $F@t \Leftarrow B_1@t_1, ...B_n@t_n, \neg C_1@s_1, ..., \neg C_m@s_m$

   $F@t \in H_l$ implies that

a) for each i, $1 \leq i \leq n$ that $B_i@t_i \in H_k$ for some $k < l$

b) for each i, $1 \leq i \leq m$ that $C_i@s_i \in H_k$ for some $k < l$

3. For each ground term $A@t \in H_l$, then there is some $m_+ < l$ such that $+A@t \in H_{m_+}$ and there is some $m_- < l$ such that $+A@T \in H_{m_-}$.

Note that the stratification requires that all updated terms $A@T$ follow their updating terms $+A@T$, and $-A@T$ in the stratification.

Given such a local stratification it is easily verified that the mapping T_P is monotonic over the appropriately restricted lattices used in the constructions of [Prz88] and [Llo87]. Thus if there is such a stratification a minimal perfect model of P can be constructed.

In general it is not possible to easily decide whether a program has a local stratification. However in the current context if $\mathcal{T}$ is the integers then two simple restrictions on P guarantee that a local stratification exists. It is built on a (finite) stratification of the normal rules and then a further constraint that the update rules are causal. A stratification on the normal rules is defined to be a (finite) ordering $>_N$ on class attribute pairs such that for any ground instance of a normal rule:

$$A \Leftarrow L_1, ..., L_n$$

where $A = o_1 : p_1(a_1 \rightarrow o_2)$ then $L_i = o_3 : p_2(a_2 \rightarrow o_4)$ implies that $\langle p_1, a_1 \rangle \geq_N \langle p_2, a_2 \rangle$ and $L_i = \neg o_3 : p_2(a_2 \rightarrow o_4)$ implies that $\langle p_1, a_1 \rangle >_N \langle p_2, a_2 \rangle$.

Given such a finite ordering it is possible to separate all ground (non-temporal) terms into a finite series of strata $N_1, ..., N_k$.

Each time point t then generates a sequence of strata $H_{t,0}, H_{t,1}, ..., H_{t,k}$ where

$H_{t,0} = \{$ all ground terms $+A@t$, $-A@t\}$ and $H_{t,i} = \{$ ground terms $A@t$ where $A \in H_i\}$.

The entire local stratification is the sequence $H_{1,0}$, $H_{1,1}$, $H_{1,2}$, ... $H_{1,k}$, $H_{2,0}$, $H_{2,1}$, ... $H_{2,k}$, ... This leads to a bottom-up computation where the database is evaluated at all points prior to t. The update terms $+A@t$ and $-A@t$ are then computed and finally the update rules are used to compute the remaining terms $A@t$ at time t.

6 Conclusion

The primary intention of this research is to present an update language for deductive and object-oriented databases with a clear declarative semantics. Such an objective is achieved by separating normal rules from update rules and by introducing temporal information in update rules. The result of this investigation sets the formal foundation for practical implementations of update operations in both deductive and object-oriented temporal databases and traditional snapshot databases.

For snapshot databases, the update rules can be simplified into the form $L \Leftarrow L_1, ..., L_n$, $n \geq 0$ where L is either a normal object term preceded by a $+$ or $-$, L_1, ..., L_n are normal object terms, negated normal object terms, or arithmetic comparison expressions. This simplified form generalizes traditional database update operations. We can translate this rule into the following explicit update rule $L@T \Leftarrow T = T_0 + 1, L_1@T_0, ..., L_n@T_0$. In this way, we could use the database state $M[t]$ to perform the intended update operation and obtain a new database state $M[t+1]$. Similarly, a query could be made only at the current time and thus could be simplified into the form ?- $L_1, ..., L_n$ where each L_i, $i = 1, ..., n$ is a normal object term, a negated normal object term, or an arithmetic comparison expression. Used in this way, temporal information would have no meaning within the database itself. It would just be used to express the semantics of updates in the database. This is the way most database systems are constructed. Note that even though the traditional database updates do not have well-defined semantics, they are well implemented. In many database systems, timestamps have been extensively used to enforce concurrency control [Ull88]. The proposed approach formalizes and provides a clear logical account for such an underlying mechanism.

References

[Abi88] Serge Abiteboul. Updates, a new database frontier. In *Proc. Intl. Conf. on Data Base Theory*, pages 1–18. Springer-Verlag Lecture Notes in Computer Science 326, 1988.

[Abi90] Serge Abiteboul. Towards a Deductive Object-Oriented Database Language. *Data and Knowledge Engineering*, 5(2):263–287, 1990.

[AG91] S. Abiteboul and S. Grumbash. COL: A Logic-Based Language for Complex Objects. *ACM TODS*, 16(1):1–30, 1991.

[AH88] S. Abiteboul and R. Hull. Data Functions, Datalog and Negation. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 143–153, 1988.

[AK89] S. Abiteboul and P.C. Kanellakis. Object Identity as a Query Language. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 159–173, 1989.

[Bry90] F. Bry. Intensional updates: Abduction via deduction. In *Proc. Intl. Conf. on Logic Programming*, 1990.

[CCCR+90] F. Cacace, S. Ceri, S. Crepi-Reghizzi, L. Tanca, and R. Zicari. Integrating object-oriented data modelling with a rule-based programming paradigm. In *Proc. Intl. Conf. on Very Large Data Bases*, pages 251–261, 1990.

[CW89] W. Chen and D.S. Warren. C-Logic for Complex Objects. In *Proc. ACM Syum. on Principles of Database Systems*, pages 369–378, 1989.

[dMS88] C. de Maindreville and Eric. Simon. Modelling non deterministic queries and updates in deductive databases. In *Proc. Intl. Conf. on Very Large Data Bases*, pages 395–406, 1988.

[KL89] M. Kifer and G. Lausen. F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Schema. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 134–146, 1989.

[KLS92] M. Kramer, G. Lausen, and G. Saake. Updates in a rule-based language for objects. In *Proc. Intl. Conf. on Very Large Data Bases*, pages 225–236, 1992.

[KLW90] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. Technical Report 90/14, Dept of CS, SUNY at Stony Brook, 1990.

[KM90] A.C. Kakas and P. Mancarella. Database updates through abduction. In *Proc. Intl. Conf. on Very Large Data Bases*, pages 650–661, 1990.

[KW89] M. Kifer and J. Wu. A Logic for Object-Oriented Logic Programming (Maier's O-logic Revisited). In *Proc. ACM Syum. on Principles of Database Systems*, pages 379–393, 1989.

[LC93] M. Liu and J. Cleary. Declarative Updates in Deductive Databases. Technical Report 93/500/05, Dept of CS, University of Calgary, 1993.

[Liu92] Mengchi Liu. *NLO: A Deductive Object Base Language.* Ph.D Thesis, University of Calgary, 1992.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2 edition, 1987.

[Mai86] D. Maier. A Logic for Objects. Technical Report CS/E-86-012, Oregon Graduate Center, Beaverton, Oregon, 1986.

[Man89] Sanjay Manchanda. Declarative expression of deductive database updates. In *ACM PODS*, pages 93–100, 1989.

[MS91] L. Edwin McKenzie and Richard T. Snodgrass. Evaluation of Relational Algebras Incorporating the Time. *Computing Surveys*, 23(4):501–543, Dec 1991.

[NK88] S. Naqvi and R. Krishnamurthy. Database Updates in Logic Programming. In *Proc. ACM Syum. on Principles of Database Systems*, pages 261–272, 1988.

[NT89] Shamim Naqvi and Shalom Tsur. *A Logical Language for Data and Knowledge Bases.* Computer Science Press, 1989.

[Prz88] T.C. Przmusinski. *On the Declarative Semantics of Deductive Databases and Logic Programs*, chapter 5, pages 193–216. Morgan Kaufmann Publishers, 1988.

[SC91] S. Y. W. Su and H. M. Chen. A temporal knowledge representation model osam*/t and its query language oql/t. In *Proc. Intl. Conf. on Very Large Data Bases*, pages 431–442, September 1991.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.