Declarative Updates in Deductive Databases

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

This paper proposes an update language called Datalog/UT which extends Datalog by incorporating update rules with temporal information. Programs to update example Databases are given. The semantics of Datalog/UT are described. Datalog/UT has a declarative semantics using local stratification which is a natural and direct extension of the traditional Datalog semantics and reduces to it in the special case. This is achieved by using explicit temporal information to control the update operations implicitly.

1 Introduction

A deductive database not only contains a set of base relations to which rules can be applied to deduce intensional relations, but also undergoes updates to absorb new information. The theory of deductive databases without updates has a well established basis in logic programming. However, a database language without updates is incomplete. How

to incorporate updates in deductive databases is currently receiving considerable attention [Abi88, Bry90, KM90, Man89, NK88].

A major difficulty is that to support updates normally requires some kind of explicit procedural constructs, contrary to the declarative nature of deductive database languages. Several languages, including DLP [Man89], LDL [NT89], sacrifice declarative semantics, to provide explicit procedural update constructs and use dynamic logic to give a procedural semantics to the languages.

This paper presents a different approach to handling update semantics. It proposes an update language called Datalog/UT which extends Datalog by incorporating update rules with temporal information. By using temporal information, the update language has a clear declarative semantics which is a natural and direct extension of the Datalog semantics and reduces to it in the special case.

This paper is organized as follows. Section 2 introduces the syntax of the language. Section 3 gives several motivating examples. Section 4 discuss the model theory and describes the bottom-up computation. Section 5 provides the conclusions.

2 Syntax of Datalog/UT

We assume knowledge of the basic concepts related to logic programming, relational and deductive databases [Llo87]. We recall some definitions relevant to our needs and present our notation.

The alphabet of the update language Datalog/UT consists of a universe C of constants, an ordered set T of times taken from C (for the sake of example we will take this set to be the

positive integers), a set V of variables, a set *Pred* of predicate names including *assign* and *delete*, and no function symbols.

A normal term is either a variable or a constant or a time. A temporal term is either a variable or a time. If p is a predicate symbol with arity n, and each S_i, for $i = 1, ..., n$, is a normal term, then $p(S_1, ..., S_n)$ is a normal atom. A ground normal atom is a normal atom which contains only constants as arguments. If $p(S_1, ..., S_n)$ is a normal atom and T is a temporal term, then $p(S_1, ..., S_n)@T$ is a temporal atom. A ground temporal atom is defined in a similar way.

A temporal literal is a temporal atom (*positive* temporal literal) or a negated temporal atom (*negative* temporal literal).

A temporal rule is of the form $A \leftarrow L_1, ..., L_n$, $n \geq 0$, where the temporal atom A is the head of the rule, and the literals $L_1, ..., L_n$ form its body.

A Datalog/UT program P consists of a set of temporal rules. The intended interpretation of this is that a Datalog database is a subset of a Datalog/UT database at a particular instant in time. Thus all the base facts in the Datalog database correspond to Datalog facts at a particular time.

There is also a subset of the Datalog/UT program that corresponds to temporal rules of the form: $A@T \leftarrow L_1@T, ..., L_n@T$, where the head and all goals in the body refer to the same time. Such rules correspond to simple Datalog rules of the form: $A \leftarrow L_1, ..., L_n$. We will refer to this set of facts and rules as the Datalog database embedded in the Datalog/UT program at a particular time.

Similar to Datalog, the temporal Herbrand base H is the set of all positive ground temporal

atoms that can be formed using predicate symbols in *Pred*, constants in C, and times in T.

3 Illustrative Examples of Programs

Before giving a formal semantics of Datalog/UT programs, we present several examples of programs.

Assume that *employee* is a base relation giving the monthly salaries and departments of employees. Consider the following update facts.

$$assign(employee, tom, shoe, 3000)@1.$$
$$assign(employee, tom, toy, 3300)@4.$$
$$delete(employee, tom)@6.$$

The intention is that the first fact inserts a new employee Tom at time 1. He is assigned to the shoe department with a salary of $3000. The second fact causes Tom to be shifted from the shoe department to the toy department with a new salary of $3300 at time 4. The third fact causes Tom to be deleted from the base relation *employee* at time 6.

The meaning of *assert* and *delete* is described by the following temporal rule. Note that these fundamental operations are not built into the language as primitives but can be expressed directly in a temporal logic.

$$\begin{aligned} employee(Emp, Dept, Sal)@T_2 \leftarrow\ & T_2 \geq T_1, \\ & assign(employee, Emp, Dept, Sal)@T_1, \\ & \neg(assign(employee, Emp, \_, \_)@T_x, T_2 \geq T_x, T_x > T_1), \\ & \neg(delete(employee, Emp)@T_x, T_2 \geq T_x, T_x > T_1). \end{aligned}$$

This makes the employee relation no longer a base relation but derives it from *assert* and *delete*. Informally the rule states that a particular employee (Emp) has values for department and salary (resp. $Dept$ and Sal) which derive from the most recent assignment. The goal

$T_2 \geq T_1$ says that these values are present for times after the assignment occurs. The two negations say that this particular entry is not present after the next assignment or deletion of this employees information. (To be more precise, if T_x is the time the next assignment or deletion occurs then the employee tuple ceases being true at times greater than or equal to T_x ($T_2 \geq T_x$). The condition $T_x > T_1$ is necessary to prevent an assignment from invalidating itself!).

The next example is an update program which includes the base relation *employee* together with its update rule above and a base relation *manager* which gives the managers of each department. The update first gives all employees a 10% salary increase, and those in managerial positions an extra \$200. Following this all employees that earn more than their managers after the salary adjustment are fired. The base relation *update* is used to specify the time at which the update is done. That is, injection of this fact into the database at a particular time triggers these rules and the consequent assignments and deletions.

$$\begin{aligned} assign(employee, E, D, S_2)@T_2 \leftarrow\ & update@T_2, \\ & T_2\ is\ T_1 + 1 \\ & employee(E, D, S_1)@T_1 \\ & manager(D, E)@T_1, \\ & S_2\ is\ S_1 * 1.1 + 200. \end{aligned}$$

$$\begin{aligned} assign(employee, E, D, S_2)@T_2 \leftarrow\ & update@T_2 \\ & T_2\ is\ T_1 + 1 \\ & employee(E, D, S_1)@T_1, \\ & \neg manager(D, E)@T_1, \\ & S_2\ is\ S_1 * 1.1. \end{aligned}$$

$$\begin{aligned} delete(employee, E)@T_2 \leftarrow\ & T_2\ is\ T_1 + 1, \\ & update@T_1 \\ & employee(E, D, S_1)@T_1, \\ & manager(D, B)@T_1, \\ & employee(B, D, S_2)@T_1, \\ & S_1 > S_2 \end{aligned}$$

Figure 1 shows the evolution of an example database after the injection of the fact *update* at time 11.

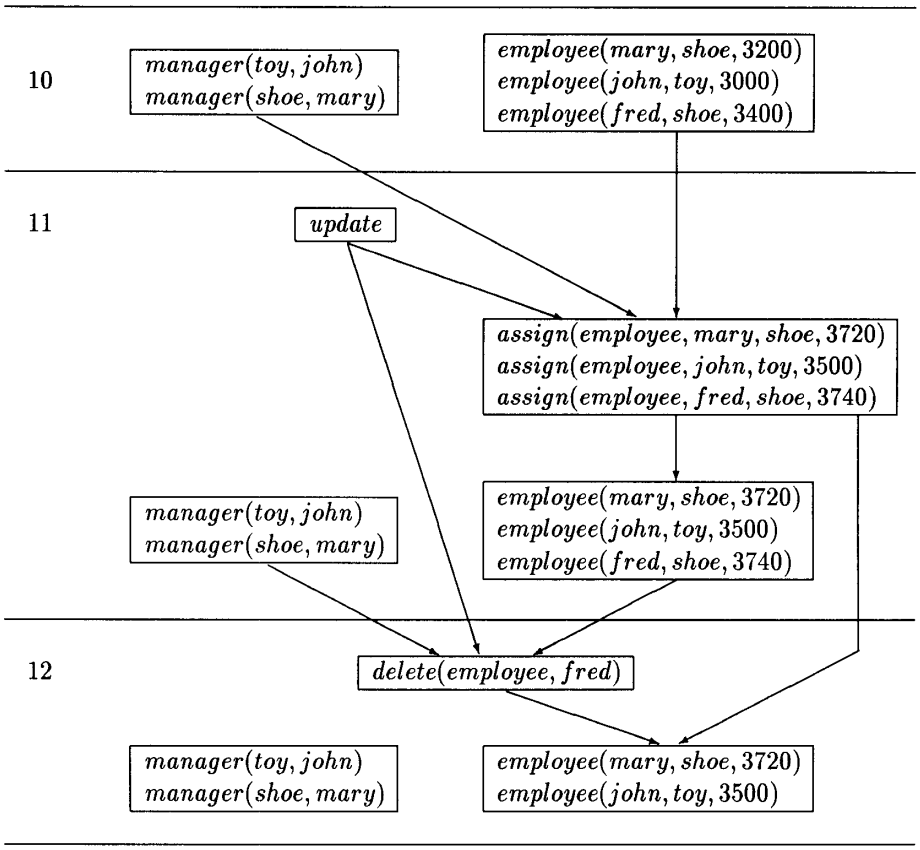

Figure 1: Evolution of a database following injection of *update*@11

4 Semantics of Datalog/UT

Both Datalog and Datalog/UT programs are normal programs and as such inherit the semantics of normal programs [Llo87, Prz88]. What we are interested in is finding minimal models of a program, that is, in finding minimal subsets of the (temporal) Herbrand universe which

are models. Because of the presence of negation in the programs it is possible for more than one minimal model to exist. A preferred unique model can be selected by imposing a priority ordering or local stratification on the Herbrand universe [Prz88]. The difference between Datalog and Datalog/UT is the choice of ordering.

Datalog is commonly stratified by ordering on the names of the predicates. Thus each predicate inhabits a layer or stratum. A predicate may depend positively on other predicates in the same or lower stratum. However, it may depend negatively only on predicates that are in a strictly lower stratum.

In Datalog/UT the ordering is done on the basis of the ordering of the times. That is, each time value determines a stratum. The result is a causality principle that a predicate may depend only on values at the same or an earlier time.

To obtain a local stratification it is necessary to impose a partial ordering on the (temporal) Herbrand universe and then show that rules in the program conform to the stratification. That is that any term in the head of a ground instance of a clause is greater than or equal to all terms in the body of the rule. (Note that we order in the conventional way where later times are higher in the ordering, this is the opposite of the priority ordering used in [Prz88]). As well there may be no cycles in the call graph including a negation which leads back to the same term.

Let the partial order on the temporal Herbrand universe be $<_T$ and the corresponding less than or equal to relation be $\leq_T$. The first requirement for local stratification is that for any ground clause instance: $A@T \leftarrow B_1@T_1, ..., B_m@T_m, \neg B_{m+1}@T_{m+1}, ..., \neg B_n@T_n$, $m \geq 0, m \leq n$. then $B_i@T_i \leq_T A@T$, $1 \leq i \leq n$.

The second requirement is that there is no cycle of ground rule instances such as the following where there is term that depends (possibly indirectly) on its own negation:

$$A@T \leftarrow , \ldots , \neg C@T_i.$$

$$C@T \leftarrow , \ldots , A@T.$$

Given a (standard) stratification on the Datalog portion of a Datalog/UT program which induces an ordering $<_P$ ($\leq_P$) on the (non-temporal) Herbrand universe then a temporal local stratification can be defined as follows: $A@T <_T B@U$ iff $T < U$ or $U = T$ and $A <_P B$. That is terms at the same time are ordered by the Datalog stratification otherwise the time determines the ordering. The example program above is temporally stratified. Even though there is a cycle of calls: *employee* calls $\neg(assign)$ which in turn calls *employee* there is a time advance in the rule that defines assign and delete (notice the constraint that T_2 is $T_1 + 1$ that is $T_2 > T_1$). So the employee relation is used at one time to compute the changes which are to take place and the change takes place later.

5 Implementation

Given such a locally stratified program it is possible to compute the derived relations bottom up. In this case it means first computing the relations at time 0, then at time 1 and so on. Kaushik in [Kau91] presents a technique for optimizing this process drawing on techniques of discrete event simulation and connection graph theorem proving. In particular he shows how to avoid recomputing relations at time points where no changes occur. As well the techniques used in bottom up computation of Datalog programs can still be used to good effect. [Kau91, Cle90, CK91] discuss the application of temporal ordering to the bottom up

computation of general logic programs (including function symbols).

Clearly Datalog/UT is a powerful but low level language (there can be few other languages which have assignment as a composite, not a primitive, operation) and there are some things that might be done to ease its use. For example it is probably too great a burden for the user to write an update rule for every base Datalog relation. However, so long as the primary key for a relation is known an update rule as in the example can be automatically generated.

A second practical problem is what to do in the presence of conflicting *assign* operations. For example $assign(employee, tom, toy, 3000)@11$ and $assign(employee, tom, shoe, 4000)@11$ might both be present. Of course what should be done is dependent on what the user wants. However, we will now show one possible approach to the problem. A rule is added which detects such conflicts:

$$\begin{aligned} &assign\_error@T \leftarrow \\ &\qquad assign(employee, Id, D1, S1)@T, \\ &\qquad assign(employee, Id, D2, S2)@T, \\ &\qquad (D1 \neq D2; S1 \neq S2). \end{aligned}$$

This predicate is then used to prevent the update of the *employee* relation with a simple modification of the update rule from the example above.

$$\begin{aligned} &employee(Emp, Dept, Sal)@T_2 \leftarrow \\ &\qquad T_2 \geq T_1, \\ &\qquad \neg(assign\_error@T_1), \\ &\qquad assign(employee, Emp, Dept, Sal)@T_1, \\ &\qquad \neg(assign(employee, Emp, \_, \_)@T_x, T_2 \geq T_x, T_x > T_1), \\ &\qquad \neg(delete(employee, Emp)@T_x, T_2 \geq T_x, T_x > T_1). \end{aligned}$$

6 Conclusion

This paper has shown that by making the local stratification of programs explicit in time stamps a pure logical theory of updates is possible. This brings updates which have always been problematic and seemingly required a procedural and non-logical semantics firmly into the mainstream of deductive databases. As yet no practical implementation of this approach has been constructed. However, it is known how to avoid recomputing relations except when they are updated and that many of the standard techniques of deductive database implementation can still be used. So there is hope that an efficient implementation is possible. We look forward to this challenge.

7 Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada and by Jade Simulations International.

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.

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

[CK91] J.G. Cleary and V.N. Kaushik. Updates in a Temporal Logic Programming Language. Technical Report 91/427/11, Dept of Computer Science, University of Calgary, 1991.

[Cle90] J. G. Cleary. Colliding Pucks Solved Using a Temporal Logic . In *Proc. Conf. on Distributed Simulation*, San Diego, Jan 1990.

[Kau91] V.N. Kaushik. *Starlog: from Semantics to Interpretation.* MSc. Thesis University of Calgary, 1991.

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

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

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

[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.