Integrating Active Rules in U-Datalog

Vincenzo Gervasi and Alessandra Raffaetà
Dipartimento di Informatica
Università di Pisa
{gervasi,raffaeta}@di.unipi.it

Abstract

The marriage between logic programming and databases has given rise to the definition of *deductive databases*. These systems allow the users to express data manipulations and queries in a declarative way, and permit efficient storage and retrieval of intensional knowledge.

Another improvement in the database field has come from the use of *active rules*, linking the occurrence of certain events to a reaction (e.g., updates of some data). This kind of rules has proven to be very useful to ensure integrity constraints and to automatize common procedures.

This paper presents an integration of active rules in U-Datalog, which is an extension to Datalog supporting declarative specification of updates based on a nonimmediate update semantics. The resulting language, called Active-U-Datalog, extends the semantics of U-Datalog in a conservative way, introducing a PARK-like semantics for active rules activation and firing, and for handling conflicting update requests.

Keywords : Deductive databases, Active rules, Semantics.

1 Introduction

The evolution of database languages in recent years has been driven by two main goals.

The first one has been to avoid any *procedural* description of the operations to perform on the database. The advantages of abstracting from the actual implementation of the data store are obvious: there is no need to know anything about the underlying data representation, thus allowing for easier "program" expression and separation of database engines evolution from linguistic issues. The widespread use of SQL and its derivatives, both as actual query and data manipulation languages and, more recently, as high-level protocols to interact with heterogeneous data repositories, is the most remarkable success of *declarative* languages in this field. The natural evolution of this research has led to the use of *logic-based* languages, like Datalog [1, 23], $\mathcal{LDL}$++ [29] or RDL [13], that add *deductive* power to the simple databases we were used to work with. These deductive databases can efficiently handle vast amounts of derived knowledge, moving most semantic issues from (diffuse) "applicative logic" to (centralized) database design.

The second main goal has been to make the databases *smart* about the data they contain. This need is particularly evident when dealing with consistency issues; one could well ask that a semantic-aware database could *react* correctly to changes to the data it contains, updating other data accordingly, in order to maintain consistency. Besides maintaining integrity constraints, these capabilities are desirable to automatize common procedures (e.g., having an order shipped when the stock is low, or applying a discount when the stock is too high). This need has led to the design of *active* databases, in which *events* of various kinds (both internally- and externally-generated) may cause the firing of so-called "active rules". Also in this case knowledge about the data semantics and the design of the database is moved into the database itself, thus simplifying applicative software and reducing the chance for consistency-breaking data manipulations. The field has produced various results, both commercial [2, 14, 19] and academic [7, 9, 12, 16, 20, 21, 26, 27] (the latter usually being more flexible than the former). However, the experience with these languages has often exposed the lack of a clear, unified and formal foundation for active rules [25].

These two approaches to improve database management systems are not in opposition, but rather they can be composed almost freely to attain the desired functionality. In giving a semantics to these active-deductive languages, two main approaches emerged: to *unify* the different paradigms under a common semantics (often by using compilative techniques) [6, 15, 18, 22, 28], or to *integrate* specific semantics [8].

In this paper, we follow the latter approach. We present an integration of deductive and active databases by extending U-Datalog [4, 17] with a support for active rules in the style of the PARK semantics [11]; we call the resulting language Active-U-Datalog. U-Datalog is an extension to Datalog supporting declarative specification of updates based on a nonimmediate update semantics. U-Datalog is well suited to such an extension thanks to its clean design, that concentrates on deductive issues without polluting the definition with unrelated needs. On the other hand, the PARK semantics is very flexible in its approach to conflicting updates, a subject that U-Datalog does not handle in a satisfactory way. Moreover, the PARK semantics leaves wide freedom in the choice of a conflict resolution policy, a virtue that can prove useful in implementing different strategies.

In section 2, we give an overview of U-Datalog and of Active-U-Datalog. In section 3 we give the formal syntax of Active-U-Datalog, and in section 4 its semantics. Some final remarks follow.

2 Overview of U-Datalog and Active-U-Datalog

We present here an overview of U-Datalog and we refer to [17] for a complete description.

Let us consider pairwise disjoint sets of *constants* Σ, *predicate names* Π and *variables* V. *Terms* range over $\Sigma \cup V$ and *predicates* are defined as usual on Π. Π is divided into two disjoint sets: *extensional* (Π^e) and *intensional* (Π^i) predicates. Update operations are expressed by extensional atoms prefixed with $+$ and $-$ to denote respectively insertions or deletions.

A U-Datalog program (or database) consists of an extensional database (EDB), that is a set of ground extensional atoms, and an intensional database (IDB), that is a set of rules of the following form:

$$H \leftarrow U_1, \ldots, U_i, B_{i+1}, \ldots, B_n. \qquad n \geq 0$$

where U_j is an update and B_k is a $(\Pi^e \cup \Pi^i, \Sigma, V)$-atom and H is an intensional atom. [17] considers databases which are safe through invocation by a query. A rule is *safe* if each variable in the head appears in a $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom in the body; it is *safe with respect to a query* if it is safe or rendered safe by bindings from the query.

A *simple transaction* T is a rule with no head. A *complex transaction* is a sequence of transactions, denoted by $T_1; T_2$.

The semantics of U-Datalog is essentially given in three steps: in the first one, the *marking phase*, the updates found during the evaluation process are collected, without executing them. Then, in the *update phase*, the requested updates are checked for consistency, that is all the requested updates must be ground and they must not be opposite ($+p(a)$ and $-p(a)$ must not be both requested). If the set of updates is not consistent the transaction is *aborted* and all the updates discarded. The third step is related to the execution of complex transactions.

In integrating active rules in U-Datalog, we want to extend its syntax and semantics in a conservative way [10]. In doing so, we augment U-Datalog programs with a set of active (Event-Condition-Action) rules. In this paper, we consider as events only *updates*, but the extension to other kind of events (temporal, system-defined...) is not difficult. Conditions are actually *queries on the deductive part* of the program, and are evaluated with regard to the *intermediate* state of the computation. This is needed to allow active rules to react to the current (possibly inconsistent) state and take appropriate actions to assure desired properties of the final state.

Conflict resolution is handled by a *parametric conflict resolution policy*; this assures that no conflicting updates will still be present during the update of the database. This approach allows us not to abort a transaction in case of conflicts (as happens in U-Datalog): we try to solve conflicts and in the worst case, when the computation requires non-ground updates, we do not modify the database, but we do not need to abort.

Our actions are *sets of updates*. We allow for more than one update in an action so that the database designer can express atomic sets of updates (we could call them sub-transactions), with the intended meaning that, if the conflict resolution policy forbids one of the updates in the set, none of the updates in the set will be requested anymore.

3 Syntax of Active-U-Datalog

We consider a fixed (Π, Σ, V)-language. Π is partitioned into three sets of symbols: Π^e the extensional predicate symbols, Π^i the intensional predicate symbols and Π^u the update predicate symbols defined as $\Pi^u = \{+p, -p \mid p \in \Pi^e\}$.

An Active-U-Datalog program can be seen as a U-Datalog program (EDB $\cup$ IDB) to which we have added a set of active rules AR. We call *deductive* part the U-Datalog program and *active* part the set of active rules.

Definition 1 (Active-U-Datalog syntax) *An* Active-U-Datalog program $P = IDB \cup EDB \cup AR$ *consists of the extensional database EDB, of the intensional database IDB and of a set of active rules AR. The EDB is a set of ground extensional atoms. The IDB is a set of rules of the form*

$$H \leftarrow U_1, \ldots, U_n, B_1, \ldots, B_m$$

where $B_1, \ldots, B_m$ is the query part (B_i is a $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom), $U_1, \ldots, U_n$ is the update part (U_i is a (Π^u, Σ, V)-atom), and the head H is a (Π^i, Σ, V)-atom. The update and query part cannot be both empty.
The AR is a set of rules of the form

$$E_1, \ldots, E_n, B_1, \ldots, B_m \rightarrow U_1, \ldots, U_k$$

where $E_1, \ldots, E_n$ is the event part (E_i is a (Π^u, Σ, V)-atom), $B_1, \ldots, B_m$ is the condition part (B_i is a positive or negative $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom) and $U_1, \ldots, U_k$ is the action part (U_i is a (Π^u, Σ, V)-atom) which cannot be empty.

A *simple transaction* or *query* T is a rule of the deductive part with no head and with a non-empty query part; a *complex transaction* is a sequence of transactions $T_1; T_2$.

We will always assume that our programs are safe with respect to a query.

Example 1 *In a corporation, employees are assigned to a single department; each employee is paid according to his or her level. When a department is deleted, all its "chieves" (that is, all employees of certain levels) are moved to a special "available personnel" list, while the others are simply fired. We want to express this policy by using active rules.*

```
EDB: dep(toys).      chieflevel(3).      chieflevel(4).
     emp(john).      level(john,2).      assign(john,toys).
     emp(mary).      level(mary,4).      assign(mary,toys).
IDB: chief(E) ← level(E,L), chieflevel(L).
AR:  −dep(D), assign(E,D) → −emp(E).
     −emp(E), chief(E) → +avail(E).
     −emp(E) → −assign(E,_), −level(E,_).
```

In response to the query "$-$dep(toys)", the first active rule asks for the deletion of every employee working in "toys". This request activates the third rule, that deletes the assignment and level of john and mary, and the second one, that puts mary in the available list.

4 Semantics

We present the semantics of Active-U-Datalog in three steps. In the first step, we compute the model of the IDB given the initial EDB and collect the set of bindings that satisfies the query and the requested updates. This step corresponds to the marking phase in U-Datalog. In the second step, we compute the semantics of the active part of the program, according to the model and the updates collected in the first step. The result of this step is the set of updates requested either from the deductive and/or the active part, in which any conflict has been solved by a parametric policy. Finally, we describe how the two semantics fit together and how we apply the computed updates to the EDB, thus obtaining the new database state.

The observable properties of the transaction we want to model are the set of bindings (the answer), the new database state and the indication of success (commit/abort) of the transaction itself.

4.1 Deductive part semantics

The semantics for the deductive part is given by defining a bottom-up operator T analogous to the immediate consequences operator of standard logic programming. This operator is defined over an extended Herbrand Base $\mathcal{B}^{\leftarrow}$containing atoms of the form $H \leftarrow \overline{U}$ where H is a $(\Pi^i \cup \Pi^e)$-atom and $\overline{U}$ is a set of updates. The meaning of such an atom is that H is true, and its evaluation requests the updates in $\overline{U}$.

Definition 2 (Immediate consequence operator) *Given an Active-U-Datalog program P, we define the* immediate consequence operator $T_P : 2^{\mathcal{B}^{\leftarrow}} \rightarrow 2^{\mathcal{B}^{\leftarrow}}$ *as follows:*

$$\begin{aligned} T_P(I) \; = \{ \;\; & A \leftarrow \overline{U} \mid A \leftarrow \overline{U} \in \mathcal{B}^{\leftarrow}, \\ & H \leftarrow U_1, \ldots, U_n, B_1, \ldots, B_m, \;\; \textit{is a renamed apart clause of } P \\ & B'_r \leftarrow \overline{U}_r \in I, \quad 1 \leq r \leq m \\ & \theta = \; mgu\ ((B_1, \ldots, B_m), (B'_1, \ldots, B'_m)) \\ & A = H\theta \\ & \overline{U} = (U_1, \ldots, U_n, \overline{U}_1, \ldots, \overline{U}_m)\theta^1 \\ \} \; & \end{aligned}$$

where the function mgu returns an idempotent mgu.

Notice that the rules involved in this definition are only of the deductive part because the head H is a $(\Pi^i \cup \Pi^e)$-atom. The operator T is continuous on the lattice $(2^{\mathcal{B}^{\leftarrow}}, \subseteq)$, and we use it to define the fixpoint semantics for the deductive part.

Definition 3 (Fixpoint semantics) *Given an Active-U-Datalog program P, we define the* fixpoint semantics $\mathcal{F}$ *of P as follows:*

$$\mathcal{F}(P) = T_P^n(\emptyset)$$

Since the domain is finite, the least fixpoint is reached in a finite number of steps (n) [5].

Example 2 *Let us consider the following Active-U-Datalog program P without active rules:*

```
EDB: t(a).    r(b).
IDB: p(X) ← q(X),+r(X)
     q(X) ← s(X)
     q(X) ← r(X)
     s(X) ← t(X),−r(X)
```

The least fixpoint computation is as follows:

$$\begin{aligned} T_P^0(\emptyset) &= \emptyset \\ T_P^1(\emptyset) &= \{t(a),\ r(b)\} \\ T_P^2(\emptyset) &= \{t(a),\ r(b),\ q(b),\quad s(a) \leftarrow -r(a)\} \\ T_P^3(\emptyset) &= \{t(a),\ r(b),\ q(b),\quad s(a) \leftarrow -r(a),\quad q(a) \leftarrow -r(a)\} \\ T_P^4(\emptyset) &= \{t(a),\ r(b),\ q(b),\quad s(a) \leftarrow -r(a),\quad q(a) \leftarrow -r(a),\quad p(a) \leftarrow -r(a), +r(a), \\ &\quad\ p(b) \leftarrow +r(b)\} \end{aligned}$$

$T_P^5(\emptyset) = T_P^4(\emptyset)$, *so we reached the least fixpoint* $\mathcal{F}(P)$.

[1] As a shorthand, we write $(U_1, \ldots, U_n, \overline{U}_1, \ldots, \overline{U}_m)\theta$ for $\{U_1\theta, \ldots, U_n\theta\} \cup \overline{U}_1\theta \cup \ldots \cup \overline{U}_m\theta$.

A set of updates $\overline{U}$ is *consistent* if it contains no opposite updates, i.e. if $+r(\tilde{t})$ and $-r(\tilde{t})$ are not both in $\overline{U}$.

It is worth remarking that the evaluation of an atom can lead to inconsistent updates as shown in the previous example, where (for instance) the evaluation of $p(a)$ requests the updates $-r(a), +r(a)$. Differing from [17], where inconsistent updates are never introduced in the model, we include them and resolve the conflicts during the active rules evaluation (section 4.2).

Before introducing the set-oriented semantics we give two auxiliary definitions:

Definition 4 *Given a set of bindings b and a transaction T, we define*

$$b_{|T} = \{(X = t) \in b \mid X \textit{ occurs in } T\}$$

Definition 5 *Given a substitution $\theta = \{V_1 \leftarrow t_1, \ldots, V_n \leftarrow t_n\}$ we define*

$$eqn(\theta) = \{V_1 = t_1, \ldots, V_n = t_n\}$$

We define the (set-oriented) semantics of a simple transaction T with regard to a program P, using its model built by $\mathcal{F}$.

Definition 6 (Query answers) *Given an Active-U-Datalog program P and a simple transaction $T = (\overline{U}_0, B_1, \ldots, B_n)$, we define the operator* Set *as follows:*

$$\begin{aligned}\mathsf{Set}((\overline{U}_0, B_1, \ldots, B_n), P) \quad = \{ \quad & \langle b, \overline{U}\rangle \mid A_i \leftarrow \overline{U}_i \in \mathcal{F}(P), \quad 1 \le i \le n \\ & \theta = mgu((B_1, \ldots, B_n), (A_1, \ldots, A_n)) \\ & b = eqn(\theta)_{|T} \\ & \overline{U} = (\overline{U}_0, \overline{U}_1, \ldots, \overline{U}_n)\theta \quad \}\end{aligned}$$

In U-Datalog, only pairs with consistent updates are inserted in Set. On the contrary, we release this restriction, deferring again the conflict resolution at the next step.

Example 3 *Let $p(X), q(X)$ be a query to the program P of example 2. Then,*

$$\mathsf{Set}((p(X), q(X)), P) \quad = \{ \quad \langle\{X = a\}, \{-r(a), +r(a)\}\rangle, \ \langle\{X = b\}, \{+r(b)\}\rangle \quad \}$$

4.2 Active part semantics

During this phase, we follow the PARK semantics proposed in [11]. This semantics is particularly suited to a deferred-update approach, like the one we used in the previous step, and adds considerable flexibility in that it uses a parametric policy to resolve conflicts.

This semantics builds an auxiliary model, in which we are interested only in the update atoms needed to trigger active rules and to obtain the new EDB. To this end, we define a bottom-up operator whose domain is $\mathcal{B}^{\pm} = \{a, +a, -a \mid a \in \mathcal{B}\}$ where $\mathcal{B}$ is the standard Herbrand Base. A subset of $\mathcal{B}^{\pm}$ is an *i-interpretation* (where the "i" stands for intermediate). An i-interpretation is *consistent* if it does not contain any opposite atoms, i.e. $+a$ and $-a$.

Definition 7 (Validity) *The validity of a ground literal a in an i-interpretation I is defined as follows:*

$$\mathsf{valid}(a, I) = \begin{cases} I \cap \{a, +a\} \neq \emptyset & \textit{if } a = p(\tilde{t}) \\ I \cap \{a, +a\} = \emptyset \textit{ or } -a \in I & \textit{if } a = \neg p(\tilde{t}) \\ a \in I & \textit{otherwise} \end{cases}$$

A positive $(\Pi^e \cup \Pi^i, \Sigma, V)$-atom is valid if it belongs to I or if an update inserting it belongs to I. A negative $(\Pi^e \cup \Pi^i, \Sigma, V)$-atom is valid if an update deleting it is in I, or if its positive atom is not valid. A (Π^u, Σ, V)-atom is valid if it belongs to I.

Definition 8 (Purification) *Let IDB be the intensional database of an Active-U-Datalog program P; we define its* purified *version $\widehat{IDB}$ as follows:*

$$H \leftarrow U_1, \ldots, U_n, B_1, \ldots, B_m \in IDB \quad \Longrightarrow \quad B_1, \ldots, B_m \rightarrow H \in \widehat{IDB}$$

It is worth noting that a query is provable in $\widehat{\text{IDB}} \cup \text{EDB}$ if and only if it is provable in $\text{IDB} \cup \text{EDB}$, with the same computed answers. The purification only avoids the side effects of the query evaluation. Also notice that we reversed the direction of the arrow in order to have a uniform notation with active rules. No ambiguity can arise since active rules have only update atoms in their heads, while purified ones have (Π^i, Σ, V)-atoms. In the following, we will generically refer to active and purified rules as "rules". Notice that definitions 9, 10, and 11 only consider active rules, while subsequent ones work on both kinds of rules.

Definition 9 (Conflicts) *A pair (r, θ), where r is a rule and θ is a ground substitution for r is called a* rule grounding*.*

Let P be a set of rules and I an i-interpretation for P. Then $\mathsf{conflicts}(P,I)$ *is a set of maximal triples of the form $(a, \mathsf{ins}, \mathsf{del})$ such that a is a ground atom and* ins *and* del *are sets of rule groundings. For each such triple the following conditions must hold:*

1. $\exists\ r = l_1, \ldots, l_n \to u_1, \ldots, u_k, \quad r' = l'_1, \ldots, l'_m \to u'_1, \ldots, u'_s, \quad r, r' \in P$, *and* $\exists\ \theta, \theta'$ *ground substitutions such that*
   - $\forall\ 1 \leq i \leq n,\ \mathsf{valid}(l_i\theta, I)$,
   - $\forall\ 1 \leq i \leq m,\ \mathsf{valid}(l'_i\theta', I)$,
   - $\exists\ i, j.\ 1 \leq i \leq k\ u_i = +l_0,\ 1 \leq j \leq s\ u'_j = -l'_0$ *and* $a = l_0\theta = l'_0\theta'$.
2. *For all possible r, r' and θ, θ', satisfying condition 1 above,* $(r, \theta) \in \mathsf{ins}$ *and* $(r', \theta') \in \mathsf{del}$.

A triple $(a, \mathsf{ins}, \mathsf{del}) \in \mathsf{conflicts}(P, I)$ is called a *conflict.*

The previous step (definitions 2 and 6) could have produced conflicts; we are now going to resolve them via a parametric policy.

Definition 10 (Conflict resolution policy) *Given an extensional database EDB, a set of rules P, an i-interpretation I and a conflict c, we define* $\mathsf{sel}(EDB, P, I, c)$ *as a total function with codomain* $\{\mathsf{insert}, \mathsf{delete}\}$.

The intended meaning of $\mathsf{sel}(\text{EDB}, P, I, (a, \mathsf{ins}, \mathsf{del}))$ is to choose if the atom a, object of the conflict, should be inserted in or deleted from I, thus effectively choosing which of the conflicting update requests should prevail. This is obtained by *preventing* the rule instances in one of the two sets from firing.

[11] presents a number of common policies, and discusses their advantages and disadvantages. We point out here just two of the most common ones. The *principle of inertia* states that *both* of the conflicting updates should be discarded, effectively leaving EDB in the same state as before with regard to a (in our framework, this can be obtained by returning insert if a was already in EDB, delete otherwise). The *rule priority* policy, found in systems such as Ariel [12], Postgres [20] and Starburst [27], assumes that each rule has a (static or dynamic) priority associated with it; sel returns insert or delete as needed to preserve the update requested by the highest-priority rule. Other policies, like voting schemes and user queries, are also reasonable, but the final choice is better left to the particular application.

Definition 11 (Blocked rule instances) *Given an extensional database EDB, a set of rules P, a conflict resolution policy* sel*, and an i-interpretation I, let*

$$\begin{aligned} X &= \{\mathsf{del} \mid (a, \mathsf{ins}, \mathsf{del}) \in \mathsf{conflicts}(P, I) \text{ and } \mathsf{sel}(EDB, P, I, (a, \mathsf{ins}, \mathsf{del})) = \mathsf{insert}\} \\ Y &= \{\mathsf{ins} \mid (a, \mathsf{ins}, \mathsf{del}) \in \mathsf{conflicts}(P, I) \text{ and } \mathsf{sel}(EDB, P, I, (a, \mathsf{ins}, \mathsf{del})) = \mathsf{delete}\} \end{aligned}$$

We define then $\mathsf{blocked}(EDB, P, I, \mathsf{sel}) = \left(\bigcup_{x \in X} x\right) \cup \left(\bigcup_{y \in Y} y\right)$

Example 4 *Let us consider the following extensional database EDB and the set of rules P:*

```
EDB:    t(a,a).  t(a,b).  r(a).
P:   r1 t(a,X) → p(X)
     r2 s(X), r(X) → p(X)
     r3 +s(X),t(X,Y) → +q(X)
     r4 +s(X),t(X,Y) → −q(Y)
     r5 +s(X) → +q(b)
     r6 +q(X),p(X) → −t(X,X)
```

Given an i-interpretation $I = \{t(a,a), t(a,b), r(a), +s(a)\}$, we have that

$$\begin{array}{rl} \mathsf{conflicts}(P, I) = \{ & (q(a), \{(r_3, \{X \leftarrow a, Y \leftarrow a\})\}, \{(r_4, \{X \leftarrow a, Y \leftarrow a\})\}), \\ & (q(b), \{(r_5, \{X \leftarrow a\})\}, \{(r_4, \{X \leftarrow a, Y \leftarrow b\})\}) \quad \} \end{array}$$

Now, let us suppose that the conflict resolution policy sel *is such that*

$$\begin{array}{rcl} \mathsf{sel}(EDB, P, I, (q(a), \{(r_3, \{X \leftarrow a, Y \leftarrow a\})\}, \{(r_4, \{X \leftarrow a, Y \leftarrow a\})\})) & = & \mathsf{insert} \\ \mathsf{sel}(EDB, P, I, (q(b), \{(r_5, \{X \leftarrow a\})\}, \{(r_4, \{X \leftarrow a, Y \leftarrow b\})\})) & = & \mathsf{delete} \end{array}$$

Then, we have that

$$\mathsf{blocked}(EDB, P, I, \mathsf{sel}) = \{(r_4, \{X \leftarrow a, Y \leftarrow a\}), (r_5, \{X \leftarrow a\})\}$$

To solve the condition part of an active rule, we want to use the knowledge in the deductive part. However, in exploiting this knowledge we work under different conditions with respect to what we did in section 4.1:

- the resolution of a condition should not affect the state of the EDB. To obtain this, we forbid any side effect (i.e., updates) during the resolution process of a condition;
- conditions should be checked against the *current* state of the EDB, in order to take into account the updates requested by the active rules.

Definition 12 (Immediate consequence operator) *Given a set of rules P, a set of blocked rule instances B, and an i-interpretation I, we define $\Gamma_{P,B}(I)$ as the smallest set U satisfying the following conditions:*

- *$I \subseteq U$;*
- *If $r = l_1, \ldots, l_n \rightarrow u_1, \ldots, u_k \in P$ and θ is a ground substitution for r such that*
  - $(r, \theta) \notin B$
  - $\forall\ 1 \leq i \leq n,\ \mathsf{valid}(l_i\theta, I)$

  then $\{u_1\theta, \ldots, u_k\theta\} \subseteq U$

$\Gamma_{P,B}$ is continuous on the lattice $(\mathcal{B}^{\pm}, \subseteq)$.

Example 5 *Given the definitions of example 4, with $B = \mathsf{blocked}(EDB, P, I, \mathsf{sel})$, we obtain*

$$\Gamma_{P,B}(I) \quad = \quad \{t(a,a), t(a,b), r(a), +s(a), p(a), p(b), +q(a), -q(b)\}$$

Notice how the use of blocked *has prevented the insertion of any conflict in $\Gamma_{P,B}(I)$.*

Definition 13 (Bi-structures) *A bi-structure $\langle B, I\rangle$ consists of a set B of blocked rule instances and an i-interpretation I. We define an order relation on bi-structures as follows:*

$$\langle B, I\rangle \prec \langle B', I'\rangle \quad \Leftrightarrow \quad \left\{ \begin{array}{ll} B \subset B' & \textit{or} \\ B = B' \textit{ and } I \subset I' & \end{array} \right.$$

Moreover, given $\mathcal{A}$ and $\mathcal{B}$ bi-structures, $\mathcal{A} \preceq \mathcal{B} \quad \equiv \quad (\mathcal{A} = \mathcal{B} \vee \mathcal{A} \prec \mathcal{B})$

Definition 14 (Δ operator) *Given a set of rules P, an extensional database EDB, a bi-structure $\mathcal{A} = \langle B, I\rangle$, a set of ground $(\Pi^e \cup \Pi^i, \Sigma, V)$-atoms $\overline{I} \subseteq I$, and a conflict resolution policy* sel*, we define*

$$\Delta_{P,\mathsf{sel},\overline{I}}(\mathcal{A}) = \begin{cases} \langle B, \Gamma_{P,B}(I)\rangle & \text{if } \Gamma_{P,B}(I) \text{ is consistent} \\ \langle B \cup \mathsf{blocked}(EDB, P, I, \mathsf{sel}), \overline{I}\rangle & \text{otherwise} \end{cases}$$

Notice that the poset of bi-structures, ordered by $\preceq$, is a cpo, and that $\Delta_{P,\mathsf{sel},\overline{I}}$ is continuous on this domain. Moreover, the least fixed point can be obtained in a finite number of steps because the cpo itself is finite.

Theorem 1 *Given a set of rules P, a conflict resolution policy* sel*, a bi-structure $\mathcal{A} = \langle B, I\rangle$, and $\overline{I} = I \setminus \{\pm a \mid \pm a \in I\}$, the following statements hold:*

1. $\mathcal{A} \preceq \Delta_{P,\mathsf{sel},\overline{I}}(\mathcal{A})$
2. $\Delta^{\omega}_{P,\mathsf{sel},\overline{I}}(\mathcal{A})$ *is a least fixed point of* $\Delta_{P,\mathsf{sel},\overline{I}}$
3. *if* $\Delta^{\omega}_{P,\mathsf{sel},\overline{I}}(\mathcal{A}) = \langle B', I'\rangle$, *then* $I' = \mathsf{lfp}(\Gamma_{P,B'})$

4.3 Integrating deductive and active semantics

In this section we show how the two semantics presented above fit together and how the result of a transaction is computed.

We are interested in modeling as observable property of a transaction the following information: the set of answers, the database state, and the result of the transaction itself (i.e. Commit and Abort).

Definition 15 (Observables) *An* observable *Oss is a triple $\langle \mathsf{Ans}, EDB, \mathsf{Res}\rangle$ where* Ans *is a set of bindings, EDB is an extensional database and* Res $\in$ {Commit, Abort}*. The set of observables is* Oss*.*

We now define a function which takes an i-interpretation and the current extensional database and returns the new extensional database obtained by executing the updates in the i-interpretation.

Definition 16 (Updates incorporation) *Given an i-interpretation I and an extensional database EDB, we define*

$$\mathsf{incorp}(I, EDB) = (EDB \setminus \{a \mid -a \in I\}) \cup \{a \mid +a \in I\}$$

Finally, we define an auxiliary operator that transforms the requested updates collected during the first step into active rules.

Definition 17 (Updates as rules) *Given a set of rules P and a set of updates U, we define*

$$P_U = P \cup \{ \rightarrow \pm a \mid \pm a \in U\}$$

Now, we can give the Active-U-Datalog semantics.

Definition 18 (Active-U-Datalog semantics) *Given an Active-U-Datalog program P (with EDB extensional database, IDB intensional database, and AR active rules), a transaction T and a conflict resolution policy* sel*, the semantics of a transaction T is denoted by the function Sem defined as*

$$Sem_{P,\mathsf{sel}}(T) = \mathcal{S}^{\mathcal{DA}}_{IDB,AR,\mathsf{sel}}(T)(\langle \emptyset, EDB, \mathsf{Commit}\rangle)$$

where the function $\mathcal{S}^{\mathcal{DA}}_{IDB,AR,\mathsf{sel}}(T) : \mathsf{Oss} \rightarrow \mathsf{Oss}$ is defined as follows:

If T is a simple transaction, then

$$\mathcal{S}^{\mathcal{DA}}_{IDB,AR,\mathsf{sel}}(T)(Oss_i) = \begin{cases} \langle \mathsf{Ans}, \mathsf{incorp}(I, EDB), \mathsf{Commit}\rangle & \textit{if OK} \\ \langle \emptyset, EDB, \mathsf{Commit}\rangle & \textit{otherwise} \end{cases}$$

where

$$\begin{aligned} \mathsf{Ans} &= \{b_j \mid \langle b_j, \overline{u}_j\rangle \in \mathsf{Set}(T,P)\} \\ \overline{U} &= \bigcup \overline{u}_j \\ \langle B, I\rangle &= \Delta^{\omega}_{\widehat{(IDB \cup AR)}_{\overline{U}},\mathsf{sel},EDB}(\langle \emptyset, EDB\rangle) \\ OK &\equiv \overline{U} \textit{ is ground} \end{aligned}$$

If T is a complex transaction $T_1;T_2$, then

$$\mathcal{S}^{\mathcal{DA}}_{IDB,AR,\mathsf{sel}}(T_1;T_2)(Oss_i) = \mathcal{S}^{\mathcal{DA}}_{IDB,AR,\mathsf{sel}}(T_2)(Oss_{i+1})$$

where

$$\mathsf{Oss}_{i+1} = \mathcal{S}^{\mathcal{DA}}_{IDB,AR,\mathsf{sel}}(T_1)(Oss_i)$$

It is worth remarking that our semantics never produces aborts caused by conflicts, differing from [3], augmenting, therefore, the successful computations. This is possible due to the presence of the active part, and especially of the conflict resolution policy. Moreover, note that the "otherwise" case discards all changes to the EDB if a non-ground update is requested. The semantics of a complex transaction is simply given by the sequential composition of the updates of its components, since none of them can abort. Besides, this semantics discards the answer to the first transaction, to stay close to the approach in [3].

Example 6 *Let us return to the corporation we met in example 1. We want that, when a department is closed, all its chieves are moved to a different department, while the remaining personnel is fired. This policy is expressed by the following Active-U-Datalog program P:*

```
EDB: dep(toys).     chieflevel(3).    chieflevel(4)
     emp(john).     level(john,2).    assign(john,toys).
     emp(mary).     level(mary,4).    assign(mary,toys).
IDB: chief(X) ← level(X,L), chieflevel(L).
     chief_of(X,D) ← assign(X,D), chief(X).
     close_move(D,B) ← −dep(D), +emp(X), +assign(X,B), chief_of(X,D).
AR:  −dep(D), assign(E,D) → −emp(E).
     −emp(E), assign(E,D), level(E,L) → −assign(E,D), −level(E,L).
     +assign(X,D), assign(X,B), ¬(B=D) → −assign(X,B)
     +assign(X,D), ¬dep(D) → +dep(D)
```

Let us compute the semantics of the query $T =$ `close_move(toys, shoes)`, using the inertial *conflict resolution policy* sel *(that is, always leave the extensional database untouched when a conflict arises).*

First, we compute the fixed-point semantics of the deductive part of P (definition 3):

$\mathcal{F}(P) = \{$ *dep(toys), chieflevel(3), chieflevel(4), emp(john), level(john,2), assign(john,toys), emp(mary), level(mary,4), assign(mary,toys), chief(mary), chief_of(mary,toys), close_move(toys,B)← −dep(toys), +emp(mary), +assign(mary,B)* $\}$

Then, we compute the query answer (definition 6):

$\mathsf{Set}(T,P) = \{ \ \langle\ \emptyset,$ *{−dep(toys), +emp(mary), +assign(mary,shoes)}* $\rangle \quad \}$

The set of requested updates is thus

$\overline{U} = \{ \ -dep(toys),\ +emp(mary),\ +assign(mary,shoes) \quad \}$

and, since it is ground, we are in the "OK" case of the $\mathcal{S}^{\mathcal{DA}}_{IDB,AR,\mathsf{sel}}$ *definition and so we can compute the active part semantics.*

The purified IDB (definition 8) is the following:

$\widehat{IDB}$:
```
level(X,L), chieflevel(L) → chief(X)
assign(X,D), chief(X) → chief_of(X,D)
chief_of(X,D) → close_move(D,B)
```

so the set of rules $P' = (\widehat{IDB} \cup AR)_{\overline{U}}$ *we use in the computation of* Δ^{ω} *is:*

$\widehat{IDB}$:	r_1	`level(X,L), chieflevel(L) → chief(X)`
	r_2	`assign(X,D), chief(X) → chief_of(X,D)`
	r_3	`chief_of(X,D) → close_move(D,B)`
AR:	r_4	`−dep(D), assign(E,D) → −emp(E).`
	r_5	`−emp(E), assign(E,D), level(E,L) → −assign(E,D), −level(E,L).`
	r_6	`+assign(X,D), assign(X,B), ¬(B=D) → −assign(X,B)`
	r_7	`+assign(X,D), ¬dep(D) → +dep(D)`
from U:	r_8	`→ −dep(toys)`
	r_9	`→ +emp(mary)`
	r_{10}	`→ +assign(mary,shoes)`

The computation of $\Delta^{\omega}_{P',\mathsf{sel},EDB}$ *proceeds as follows (we denote with* $\langle B^i, I^i\rangle$ *the bi-structure computed at the i-th step):*

$$\begin{aligned}
\Delta^0_{P',\mathsf{sel},EDB}(\langle\emptyset, EDB\rangle) &= \langle\emptyset, EDB\rangle \\
\Delta^1_{P',\mathsf{sel},EDB}(\langle\emptyset, EDB\rangle) &= \langle\emptyset, I^0 \cup \{+assign(mary, shoes), +emp(mary), \\
&\qquad -dep(toys), chief(mary), chief_of(mary, toys)\}\rangle
\end{aligned}$$

Now, in the computation of $\Gamma_{P',\emptyset}(I^1)$ *a conflict arises:*

$\Gamma_{P',\emptyset}(I^1) = \{$ *dep(toys), chieflevel(3), chieflevel(4), emp(john), emp(mary), level(john,2), level(mary,4), assign(john,toys), assign(mary,toys), +assign(mary,shoes),* **+emp(mary)**, *−dep(toys), chief(mary), chief_of(mary,toys), close_move(toys,...), −emp(john),* **−emp(mary)**, *−assign(mary,toys), +dep(shoes)* $\}$

The conflict is $c = (emp(mary), \{\langle r_9, \emptyset\rangle\}, \langle r_4, \{D = toys, E = mary\}\rangle)$*; since we are using the inertial conflict resolution policy,* $\mathsf{sel}(EDB, P', I^1, c) = \mathsf{insert}$*, thus blocking* $(r_4, \{D \leftarrow toys, E \leftarrow mary\})$*. The computation proceeds without further conflicts and the least fixpoint of* $\Delta_{P',\mathsf{sel},EDB}$ *is*

$$\begin{aligned}
&\mathsf{lfp}(\Delta_{P',\mathsf{sel},EDB}) = \langle\{\langle r_4, \{D \leftarrow toys, E \leftarrow mary\}\rangle\}, \\
&\quad \{dep(toys), chieflevel(3), chieflevel(4), emp(john), emp(mary), level(john, 2), \\
&\quad level(mary, 4), assign(john, toys), assign(mary, toys), +assign(mary, shoes), \\
&\quad +emp(mary), -dep(toys), chief(mary), chief_of(mary, toys), close_move(toys, \ldots), \\
&\quad -emp(john), -assign(mary, toys), +dep(shoes), -assign(john, toys), -level(john, 2)\}\rangle
\end{aligned}$$

We can now incorporate the updates in the EDB, thus obtaining EDB':

$EDB' = \mathsf{incorp}(I, EDB) = \{$ *chieflevel(3), chieflevel(4), emp(mary), level(mary,4), assign(mary,shoes), dep(shoes)* $\}$

Finally,

$$Sem_{P,\mathsf{sel}}(T) = \mathcal{S}^{\mathcal{DA}}_{IDB,AR,\mathsf{sel}}(T)(\langle\emptyset, EDB, \mathsf{Commit}\rangle) = \langle\emptyset, EDB', \mathsf{Commit}\rangle$$

5 Conclusions and future work

In this paper we have shown a conservative extension [10] to U-Datalog with a support for active rules. While the full deductive power of U-Datalog is still intact, active rules allow for clear and efficient expression of integrity constraints and other active policies, moving most logic from applications into the database.

The fact that this extension does not compromise the deductive aspects of U-Datalog let us think that the active and deductive paradigms are more like to orthogonal axes rather than to ends of a spectrum as proposed in [24], and that the two approaches can be mixed in a careful design without hindering each other.

More work is needed to enrich the set of events to which an Active-U-Datalog program can react (HiPAC [7, 16], for example, can react to database operations, methods invocations, transaction outcomes, temporal events, external messages and to almost arbitrarily complex combinations of these events) but this can be best done in designing a complete DBMS rather than in giving semantics to a language. Another promising extension is the integration of Active-U-Datalog with Object-U-Datalog [3]. Such a language would offer clean modularity in addition to the advantages of deductive and active languages.

Acknowledgments: We thank P. Baldan for his comments on earlier versions of this paper and E. Bertino for pointing us to the integration of active and deductive databases.

References

[1] S. Abiteboul and E. Simon. Fundamental properties of deterministic and nondeterministic extensions of Datalog. *Theoretical Computer Science*, (78):137–158, 1991.

[2] ANSI TC X3H2 and ISO/IEC JTC 1/SC 21/WG 3. Master index for SQL and all its parts, Mar. 1966. Document X3H2-96-066 DBL:MCI-011.

[3] E. Bertino, G. Guerrini, and D. Montesi. Toward deductive object databases. *Theory and Practice of Object Systems*, 1(1), 1995.

[4] E. Bertino and D. Montesi. Modeling database updates with constraint logic programming. In U. W. Lipeck and B. Thalheim, editors, *Proc. of the Fourth International Workshop on Foundations of Models and Languages for Data and Objects*, pages 120–132, 1992.

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

[6] S. Ceri and J. Widom. Deriving incremental production rules for deductive data. *Information Systems*, 19(6):467–490, 1994.

[7] S. Chakravarthy et al. HiPAC: A research project in active, time-constrained database management (final report). Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, Massachussetts, Aug. 1990.

[8] A. A. A. Fernandes, M. H. Williams, and N. W. Patton. A logic-based integration of active and deductive databases. *New Generation Computing*, 1996.

[9] N. Gehani and H. V. Jagadish. Ode as an active database: Constraints and triggers. In *Proc. of the Seventeenth International Conference on Very Large Data Bases*, Sept. 1991.

[10] V. Gervasi and A. Raffaetà. Active-U-Datalog: integrating active rules in a deductive database. Technical Report (in preparation), Dipartimento di Informatica, Pisa, Italy, 1997.

[11] G. Gottlob, G. Moerkotte, and V. S. Subrahmanian. The PARK semantics for active rules. In *Proceedings of the fifth International Conference on Extending Database Technology*, Lecture Notes in Computer Science Nr. 1057, pages 35–55. Springer–Verlag, Mar. 1996.

[12] E. Hanson. Rule condition testing and action execution in Ariel. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 49–58, San Diego, CA, June 1992.

[13] J. Kiernan, C. de Maindreville, and E. Simon. Making deductive databases a practical technology: A step forward. In *Proc. of the ACM SIGMOD International Conference on the Management of Data*, pages 237–246, Atlantic City, New Jersey, May 1990.

[14] Kirkwood. *Sybase Architecture and Administration.* Prentice-Hall, 1993.

[15] B. Ludäscher, W. May, and G. Lausen. Nested Transactions in a Logical Languages for Active Rules. In *Proceedings of the International Workshop on Logic in Databases*, Lecture Notes in Computer Science Nr. 1154, pages 197–222. Springer–Verlag, 1996.

[16] D. R. McCarhy and U. Dayal. The architecture of an active database management system. In *Proc. of the ACM SIGMOD International Conference on Extending Data Base Technology*, Mar. 1989.

[17] D. Montesi. *A Model for Updates and Transactions in Deductive Databases.* PhD thesis, Dipartimento di Informatica – Università di Pisa, 1993.

[18] D. Montesi and R. Torlone. A transaction transformation approach to active rule processing. In *Proc. of the Eleventh International Conference on Data Engineering*, pages 109–116, Mar. 1995.

[19] Oracle Corp. *Oracle 7 Server Concepts Manual, 7.3.* Oracle Corp., Feb. 1996.

[20] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos. On rules, procedures, caching and views in data base systems. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 281–290, 1990.

[21] M. Stonebraker and G. Kemnitz. The POSTGRES next-generation database management system. *Communications of the ACM*, 34(10), Oct. 1991.

[22] L. Tanca. (re-)action in deductive databases. In *Proc. of the Second International Workshop on Intelligent and Cooperative Information Systems*, pages 55–61, Como, Italy, May 1990.

[23] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I and II. Computer Science Press, Rockville, Maryland, 1989.

[24] J. Widom. Deductive and active databases: Two paradigms or ends of a spectrum? In *Proc. of the First International Workshop on Rules in Database Systems*, pages 306–315, Edimburgh, Scotland, Aug. 1993.

[25] J. Widom and S. Ceri, editors. *Active Database Systems - Triggers and rules for advanced database processing.* Morgan Kaupfmann, 1996.

[26] J. Widom, R. J. Cochrane, and B. G. Lindsay. Implementing set-oriented production rules as an extension to Starburst. In *Proc. of the Seventeenth International Conference on Very Large Data Bases*, Sept. 1991.

[27] J. Widom and S. Finkelstein. Set-oriented production rules in relational databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 259–270, 1990.

[28] C. Zaniolo. A unified semantics for active and deductive databases. In *Proc. of the First International Workshop on Rules in Database Systems*, pages 271–287, Edimburgh, Scotland, Aug. 1993.

[29] C. Zaniolo, N. Arni, and K. Ong. Negation and aggregates in recursive rules. In *Proceedings of the third International Conference on Deductive and O-O DBs*, Phoenix, Arizona, 1993.