Active-U-Datalog: Integrating Active Rules in a Logical Update Language

Elisa Bertino[1], Barbara Catania[1], Vincenzo Gervasi[2], and Alessandra Raffaetà[2]

[1] Dipartimento di Scienze dell'Informazione
University of Milano
Via Comelico, 39/41
20135 Milano, Italy
{bertino,catania}@dsi.unimi.it
[2] Dipartimento di Informatica
University of Pisa
Corso Italia, 40
56125 Pisa, Italy
{gervasi,raffaeta}@di.unipi.it

Abstract. Deductive database technology represents an important step towards the goal of developing highly-declarative database programming languages. In order to make deductive databases a practical technology, deductive rules have to be extended to provide a dynamic behavior. In particular, current applications require not only a support for updates and transactions but also the ability to automatically react to the occurrence of particular events. This is possible by integrating typical deductive rules, whose execution is user-dependent, with *active* rules, whose execution is event-dependent. Current solutions to this problem are not completely satisfactory. In particular, they often lack a clear semantics, guaranteeing termination, confluence and efficient evaluation. The aim of this paper is to propose a new language for integrating active rules, deductive rules and updates in a uniform logical context. The language we propose is based on the U-Datalog language [9], and extends it with support for active rules, modeled according to the PARK semantics [23]. The resulting language allows the representation of several dynamic aspects, such as transaction execution, reactive behavior and update propagation, in a uniform logical framework, admitting a clear and flexible semantics.

1 Introduction

It is today well recognized that the use of high-level, declarative languages greatly reduces the application development time. Deductive database technology represents an important step towards the goal of developing highly-declarative database programming languages. An important difference that characterizes a deductive database system with respect to a relational one is that the former stores not only data, but also *rules*, that allow additional information to be derived from the data stored into the database. Moreover, a deductive database system offers a uniform, declarative paradigm, based on formal logic, for the static

B. Freitag et al. (Eds.): Transactions and Change in Logic DBs, LNCS 1472, pp. 107–133, 1998.

management of the database, providing more powerful languages for defining data, rules, integrity constraints, and for expressing queries.

In order to make deductive databases a practical technology, their formal foundations have to be extended to deal with dynamic aspects, proper of any real database system. Dynamicity concerns at least two different topics:

- An update and a transaction language should be provided and typical atomicity, consistency, isolation, and durability (ACID) properties have to be guaranteed [24].
  This requirement has led to the definition of several languages integrating logic and updates [1,2,9,10,15,16,31,32,35]. In general, all those proposals are based on including special atoms denoting updates in (typically Datalog) rules. The proposed approaches differ for several aspects, such as the semantics assigned to the resulting language (declarative vs. operational), the evaluation techniques, the update position (head vs. body of the rules), the ability to model non-determinism and parallelism, the conflict resolution policy, and the ability to model transactions.
- Consistency of the database has to be guaranteed.
  This is possible by specifying appropriate integrity constraints, to be checked after any database change. In a more advanced setting, one could ask that a semantic-aware database should be able to *react* correctly to changes to the data it contains, by accordingly updating other data, in order to maintain consistency. Besides maintaining integrity constraints, these capabilities are desirable in order 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 (e.g., a query, an update) may cause the firing of so-called *active rules*.

Many proposals for using active rules in databases have appeared, both commercial [3,4,19,26,36] and academic [11,14,18,21,25,30,33,37,38,40,41,42] (the latter usually being more flexible than the former). In order to assign a clear semantics to active rules, several approaches have been proposed to integrate active rules in a deductive framework. In assigning a semantics to these active-deductive languages, two main approaches emerged. The first approach is based on *unifying* the different paradigms under a common semantics (often by using compilative techniques) [13,29,34,39,43]. The second one is based on *integrating* specific, different semantics [20,22]. Current solutions are however not completely satisfactory. Indeed, some of the proposed approaches do not guarantee termination, some others do not guarantee confluence; moreover, most languages do not have polynomial complexity. All these characteristics are fundamental in order to efficiently execute user and system requests as well as to analyze the rule behavior. The price to pay for such missing requirements is a more complex language semantics and, when dealing with a large number of rules, a significant difficulty in predicting their behavior at run-time and therefore in performing useful compile-time optimization.

The aim of this paper is to propose a new language for integrating active rules, deductive rules and updates in a uniform logical context. The language we propose is based on the U-Datalog language [9], and extends it with support for active rules, in the style of the PARK semantics [23].

U-Datalog has been introduced with the aim of providing a set-oriented logical update language, guaranteeing update parallelism in the context of a Datalog-like language. In U-Datalog, update atoms appear in bodies of Datalog rules. The execution of a goal (also called a *transaction*) in U-Datalog is based on a deferred semantics, by which several updates are generated from predicate evaluation, but not executed; rather, they are collected and are executed only at the end of the query-answering process. In U-Datalog, updates are expressed by using constraints. For example, $+p(a)$ states that in the new state $p(a)$ must be true whereas $-p(a)$ states that in the new state $p(a)$ must be false. Each atom solution generates a set of updates.

The deferred semantics of U-Datalog permits a uniform integration of U-Datalog rules with active rules, logically modeled and interpreted according to the PARK semantics.

The PARK semantics has been designed with the intent of overcoming the limitations of most previously defined semantics for active rules. In particular, given a set of ECA (Event-Condition-Action) rules, i.e. rules of the form "ON event IF condition THEN action", the PARK semantics satisfies several properties. First of all, it is non ambiguous, i.e., it always guarantees confluence of the execution. Moreover, it is flexible with respect to conflict resolution. A conflict is a situation where two or more active rules can be fired and one of these rules requests the insertion of an atom a in the database, whereas at least one of the others requests the deletion of a from the database. A conflict resolution policy is a method to determine which actions should be executed in presence of a conflict and which others should be suppressed. Under the PARK semantics, the conflict resolution policy can be chosen according to specific application requirements. A fixpoint semantics is used to determine the result of the application of a set of active rules. The fixpoint semantics has been chosen because it has a clear mathematical foundation and can be directly implemented. The proposed semantics guarantees the termination of the evaluation process. As it has been pointed out in [23], all other proposed semantics fail to satisfy these important requirements.

The integration of active rules in U-Datalog results in a new language that allows several dynamic aspects to be represented, such as transaction execution, reactive behavior and more specifically update propagation, in a uniform logical framework. The semantics we propose for active rules differs from the semantics of most other proposals in that, as the PARK semantics on which it is based, it guarantees termination and polynomial complexity (since the PARK semantics for active rules as well as the U-Datalog semantics guarantee polynomial complexity [7,23]). The use of the PARK semantics allows the system to handle updates generated by deductive rules and updates generated by active rules in a uniform way. For example, with respect to U-Datalog [9], there is no need to

define *a priori* the behavior to be taken when conflicts arise (whereas U-Datalog always reacts to conflicts by aborting the transaction). Thus, our approach can also be seen as an extension of U-Datalog to deal with multiple policies for conflict resolution. As far as we know, no other approach of this kind has been proposed yet in the context of deductive databases.

Active rules and conflict resolution policies address two different but related problems. In fact, active rules provide a means for expressing *integrity* constraints in a simple, centralized way. On the other hand, conflict resolution policies provide a decision mechanism to handle situations where *conflicting* actions are requested.

The paper is organized as follows. In Section 2, the syntax and the semantics of U-Datalog are informally introduced, together with the basic ideas underlying the introduction of active rules in U-Datalog. The syntax of the new language, called Active-U-Datalog, is formally introduced in Section 3 whereas its semantics is described in Section 4. Finally, Section 5 presents some conclusions and outlines future work.

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

In this section, we informally present the basic notions of U-Datalog and Active-U-Datalog. We refer the reader to [9] for a complete description of U-Datalog. Moreover, we assume that the reader is familiar with the basic logic programming concepts [28].

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

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

where U_j, $j = 1, ..., i$, are *update atoms* and B_k, $k = i + 1, ..., n$, are atoms, in the usual logic programming sense. An update atom is an atom preceded by the symbol $+$, to denote an insertion, or by the symbol $-$, to denote a deletion. Predicates defined in the extensional database and predicates defined in the intensional database are disjoint.

A query is a rule with no head. Since, as we will see, U-Datalog queries may generate updates, a U-Datalog query is also called *simple transaction*. A *complex transaction* is a sequence of (simple) transactions, denoted by $T_1; \ldots; T_n$.

For simplicity, it is assumed that U-Datalog databases are safe, that is all its rules are safe. A rule is *safe* if each variable in the head belongs to a non-update atom in the body. Often this condition is relaxed, assuming that the rule is *safe with respect to a query*; in this case, it should be safe when the head is unified with constants present in the query.

Example 1. Consider the following U-Datalog program:

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

The extensional database contains a fact for each of the extensional predicates r and v; the intensional database defines three intensional predicates, p, q and s. Rules for p, q and the last rule for s are safe; on the contrary, the first rule for s is not. But if we consider the transaction $T = s(b)$, then the database is safe with respect to T. Indeed, by evaluating T, all variables contained in update atoms appearing in the body of rules defining s are bound by the constant present in T. ◇

The semantics of U-Datalog is essentially given in three steps: in the first one, the *marking phase*, a set of solutions is generated. Each solution contains a set of bindings and a *consistent* set of updates. A set of updates is consistent if it does not require both the insertion and the deletion of the same fact. The union of the updates collected in all solutions returned by the marking phase are executed only in the *update phase*, if they are *consistent*. If the set of updates is not consistent, the transaction is *aborted* and all the updates are discarded. The third step is related to the execution of complex transactions. In this case, the extensional database is updated after each transaction execution. If a transaction aborts, the whole complex transaction aborts as well, to guarantee atomicity.

Example 2. Consider the U-Datalog database introduced in Example 1 and the transaction $T = q(X)$. When T is evaluated on IDB $\cup$ EDB, it computes the binding $X = a$ and collects the updates $+v(a, a)$, $-r(a)$. Since the requested updates are consistent, the new EDB obtained by executing them is EDB $= \{v(a, a), v(a, b)\}$. If we evaluate the transaction $T = q(X), s(X)$ then we obtain two solutions: one is exactly the previous one, the other has binding $X = a$ and collected updates $+v(a, a)$, $-r(a)$, $+r(a)$. Since the set of updates is not consistent, this solution is not returned to the user. Now consider the transaction $T = s(a)$. When evaluated on IDB $\cup$ EDB, it returns two solutions, both containing an empty set of bindings. The first solution generates the update $+r(a)$ whereas the second solution generates the update $-r(a)$. Each solution contains a consistent set of updates. However, the union of such sets of updates is not consistent and therefore the transaction is aborted. ◇

In integrating active rules in U-Datalog, we extend its syntax and semantics in a conservative way. In doing so, U-Datalog programs are augmented with a set of active (Event-Condition-Action) rules. The obtained language is called *Active-U-Datalog*. 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, i.e., when the computation generates non-ground updates, we only discard the updates without aborting the transaction (see Section 4.3).

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.

Example 3. Consider again the U-Datalog database introduced in Example 1. In Active-U-Datalog, we may maintain the rules of that database and introduce some active rules. For example, the active rule that, if $q(X)$ is true (condition) and the fact $r(X)$ is removed (event), then inserts the fact $v(X, X)$ (action), can be represented as follows:

$$-r(X), q(X) \rightarrow +v(X, X).$$

$\diamond$

In the following sections, the syntax and the semantics of Active-U-Datalog will be described in more details.

3 Syntax of Active-U-Datalog

We consider a fixed (Π, Σ, V)-language. The set Σ and V are respectively the set of constant and variable symbols. The set of predicate symbols Π is partitioned into three sets: the extensional predicate symbols Π^e, the intensional predicate symbols Π^i and the update predicate symbols Π^u, defined as $\Pi^u = \{+p, -p \mid p \in \Pi^e\}$. We denote with (Π, Σ, V)-atom an atom whose predicate belongs to Π and whose terms are in $\Sigma \cup V$. Update atoms are extensional atoms prefixed by $+$, to denote insertion, and by $-$, to denote deletion.

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 *or* database $P = IDB \cup EDB \cup AR$ *consists of an extensional database EDB, of an intensional database IDB and of a set of active rules AR. The EDB, also called* state *of the database, is a set of ground extensional atoms. The IDB is a set of* deductive *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_j is a $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom, $j = 1, ..., m$), $U_1, \ldots, U_n$ is the update part (U_j is a (Π^u, Σ, V)-atom, $j = 1, ..., n$), 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_j is a (Π^u, Σ, V)-atom, $j = 1, ..., n$), $B_1, \ldots, B_m$ is the condition part (B_j is a positive or negative[1] $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom, $j = 1, ..., m$) and $U_1, \ldots, U_k$ is the action part (U_j is a (Π^u, Σ, V)-atom, $j = 1, ..., k$) which cannot be empty. We require for active rules two safety conditions: each variable occurring in a rule head should also occur in the body of the same rule and each variable occurring in a negated literal in the rule body must also occur in some positive literal in the rule body. □

The intuitive meaning of a rule belonging to the IDB of an Active-U-Datalog program is: "if $B_1, \ldots, B_m$ are true then H is true and, as side effect, the updates $U_1, \ldots, U_n$ are requested". While intensional rules give deductive power to our framework, active rules allow the system to autonomously react to the current (possibly inconsistent) state and to take appropriate actions to assure desired properties on the final state. The intuitive meaning of the active rules we use in Definition 1 is: "if the events $E_1, \ldots, E_n$ occur and $B_1, \ldots, B_m$ are true, then execute actions $U_1, \ldots, U_k$".

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

It should be clear that a transaction provides different functions: the query function, in that it returns a set of bindings, and the update function with a transactional behavior [24]. As we will see in Section 4.3, the transactional behavior ensures that all the updates are executed or, in case of ungroundness, none of them is performed. We always assume that our rules are safe with respect to a transaction.

Example 4. Let us consider the administrative database of a school. We want to store information on the students and the exams they passed. We also want that, when a student leaves the school, all related information is automatically removed from the database.

The following Active-U-Datalog database models this situation:

```
EDB: student(mary).      student(frank).
     exam(engl).         exam(phys).
     passed(mary,phys).
IDB: pass(S,E) ← student(S), exam(E), +passed(S,E).
     leave(S) ← student(S), −student(S).
AR:  −student(S), passed(S,E) → −passed(S,E).
```

[1] A negative atom is denoted by $\neg p(\tilde{t})$ and negation is understood as negation as failure.

The first deductive rule allows us to insert a fact `passed(S,E)` for each student S who passes exam E. The second rule allows us to remove students when they leave the school.

The intended meaning for the active rule is the following: *whenever there is an attempt to remove a student from the database, all the* `passed` *predicates concerning him or her must be removed, too.*

The formal semantics of this intended meaning is given in the next section. ◇

4 Semantics of Active-U-Datalog

The semantics of an Active-U-Datalog program is given 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 satisfy the query and the requested updates. This step corresponds to the marking phase in U-Datalog. However, if the result of this step is a set of inconsistent updates, we do not abort as U-Datalog does; instead, we solve the conflicting updates in the next step. In the second step, we compute the semantics of the active part of the program, according to the model and to the updates collected in the first step. The result of this step is the set of updates requested either from the deductive and/or from 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 three steps (deductive part semantics, active part semantics, updates incorporation) are repeated for every simple transaction in a complex transaction. Hence, the state of the database evolves after each simple transaction.

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. Actually, our transactions always commit because of the conflicting resolution policy of the active part, that solves any inconsistency. Although our semantics never produces aborts by itself, we still consider this as an observable property to account for implementation-related aborts, e.g., hardware failures or space exhaustion on some media, and to stay close to U-Datalog semantics.

4.1 Deductive Part Semantics

The semantics for the deductive part is given by defining a bottom-up operator T analogous to the immediate consequence operator of standard logic programming [28]. This operator is defined over an extended Herbrand Base $\mathcal{CB}$ containing *constrained* atoms of the form $H \leftarrow \overline{U}$ where H is a $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom and $\overline{U}$ is a set of updates. The meaning of a constrained atom $H \leftarrow \overline{U}$ is that H is true, and its evaluation requests the updates in $\overline{U}$.[2] Since the language

[2] As a shorthand, a constrained atom $H \leftarrow$ is simply denoted by H itself.

does not contain function symbols, the set $\mathcal{CB}$ is finite. In summary, the immediate consequence operator of standard logic programming is extended to deal with update atoms, which are "gathered" in the bodies of rules during query evaluation.

Definition 2 (Immediate consequence operator). *Let P be an Active-U-Datalog program. We define the* immediate consequence operator $T_P : \wp(\mathcal{CB}) \rightarrow \wp(\mathcal{CB})$ *as follows:*

$$T_P(I) = \{A \leftarrow \overline{U} \mid \begin{array}{l} H \leftarrow U_1, \ldots, U_n, B_1, \ldots, B_m \text{ is a renamed apart clause of } P \\ B'_r \leftarrow \overline{U}_r \in I, \quad 1 \leq r \leq m \\ \theta = \ mgu\ ((B_1, \ldots, B_m), (B'_1, \ldots, B'_m)) \\ A = H\theta \\ \overline{U} = (U_1, \ldots, U_n, \overline{U}_1, \ldots, \overline{U}_m)\theta^3 \end{array} \}$$

where the function mgu returns an idempotent mgu. □

Notice that the rules involved in this definition are only rules from the deductive part because the head H is a $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom.

Theorem 1 (Continuity). *Given an Active-U-Datalog program P, the immediate consequence operator T_P (Definition 2) is a continuous function.*

Proof. By definition, the operator T_P is monotonic on the lattice $(\wp(\mathcal{CB}), \subseteq)$ and since $\wp(\mathcal{CB})$ is finite, the monotonicity of a function is a sufficient condition for its continuity. □

The continuity of T allows the fixpoint semantics for the deductive part to be defined as the least upper bound of the chain of the iterated applications of T_P, starting from the empty set.

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

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

where, as usual, $T_P^{\omega}(\emptyset)$ represents $\bigcup_n T_P^n(\emptyset)$. □

Since the domain is finite, the least fixpoint is reached in a finite number of steps [12].

Example 5. Let us consider the following Active-U-Datalog program P.

```
EDB: v(a,a).    v(a,b).     r(b).
IDB: p(X) ← q(X),+r(X).
     q(X) ← v(X,X),-r(X).
     q(X) ← r(X).
AR:  -r(X),r(Y) → +v(X,Y)
     -r(X),v(X,Y) → -v(X,Y)
     -r(X),q(X) → +v(X,X)
```

[3] We use $(U_1, \ldots, U_n, \overline{U}_1, \ldots, \overline{U}_m)\theta$ as a shorthand for $\{U_1\theta, \ldots, U_n\theta\} \cup \overline{U}_1\theta \cup \ldots \cup \overline{U}_m\theta$.

The least fixpoint is computed as follows:

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

$T_P^4(\emptyset) = T_P^3(\emptyset)$, so we reached the least fixpoint $\mathcal{F}(P)$. ◇

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 [9], where inconsistent updates are never introduced in the model, we include them and resolve the conflicts during the active rules evaluation (Section 4.2).

The semantics of a simple transaction T with respect to an Active-U-Datalog program P is defined by using the fixpoint operator defined above. As usual in database systems, we give a default set-oriented semantics, that is, the query-answering process computes a set of answers. Before formally introducing the semantics, we give two auxiliary definitions:

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

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

□

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 denote with $\mathsf{Set}(T, P)$ the set of pairs $\langle bindings, updates \rangle$ computed as answers to the transaction T.

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:*

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

□

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 (Section 4.2).

Example 6. Let $p(X), q(X)$ be a query to the program P of Example 5. Then,

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

It is worth noticing that the first tuple contains an inconsistent set of updates. ◇

4.2 Active Part Semantics

The active part semantics is given following the line of the PARK semantics proposed in [23]. This semantics is well suited to a deferred-update approach, like the one we used in modeling deductive part semantics, and adds considerable flexibility in that it uses a parametric policy to resolve conflicts.

This semantics builds an auxiliary model containing, in particular, the update atoms needed to trigger active rules and to obtain the new state of the system. To this end, we define a bottom-up operator whose domain is

$$\mathcal{B}^{\pm} = \mathcal{B} \cup \{+p(\tilde{t}), -p(\tilde{t}) \mid p(\tilde{t}) \in \mathcal{B}, p \in \Pi^e\}$$

where $\mathcal{B}$ is the standard Herbrand Base.[4] 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 pair of opposite updates, i.e. $+a$ and $-a$. Notice that this is exactly the consistency definition we have already given for sets of updates.

In the following, we denote with B, B' a $(\Pi^i \cup \Pi^e, \Sigma, V)$-atom, with U a (Π^u, Σ, V)-atom and with G a (Π, Σ, V)-atom. We sometimes add subscripts to these symbols.

To establish when an active rule can trigger, that is when its event occurs and its condition holds, we introduce the *valid* function on atoms and i-interpretations.

Definition 7 (Validity). *[23] The validity of a ground literal a in an i-interpretation I is defined as follows:*

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

□

A positive $(\Pi^e \cup \Pi^i, \Sigma, V)$-atom is valid if it belongs to I or if it is inserted by an update in I. A negative $(\Pi^e \cup \Pi^i, \Sigma, V)$-atom is valid if it is deleted by

[4] We recall that the standard Herbrand Base is the set of ground positive atoms made from predicate symbols in $\Pi^i \cup \Pi^e$ and constant symbols in Σ.

an update in I, or if its positive atom is not valid. A (Π^u, Σ, V)-atom is valid if it belongs to I. Notice that both $p(\tilde{t})$ and $\neg p(\tilde{t})$ can be valid according to this definition. The intuition behind the above definition is that since a positive or negative atom belongs to the condition part of the active rule, its validity must be checked with respect to the derived atoms and also to the inserted and deleted atoms. Otherwise, to represent the occurrence of an event, we require that just the update modeling such an event belongs to the i-interpretation.

We want to use active rules to repair consistency-breaking updates. In other words, active rules should take into account the *current* (possibly inconsistent) state of the database, and in particular the set of requested updates (updates that will not be necessarily performed). We also want active rules to take advantage of the available intensional knowledge when checking conditions on the current state of the database. However, condition checking must be unobtrusive, and in particular, must not trigger further updates that would change the state of the database as soon as it is observed. To fulfill this requirement, we remove the update part from the rules of the intensional database by using the purification operation defined below.

Definition 8 (Purification). *Given the intensional database IDB of an Active-U-Datalog program, we define its* purified *version* $\widehat{IDB}$ *as the set of rules*

$$B_1, \ldots, B_m \rightarrow H.$$

such that there exists in IDB a rule

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

□

Example 7. The purified form of the IDB of the program in Example 5 is the following:

```
IDB̂: q(X) → p(X).
     v(X,X) → q(X).
     r(X) → q(X).
```

◇

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.

Naturally, other semantics are possible. For example, rules containing update atoms could be removed or they could be maintained, considering only those intensional atoms whose derivation generates sets of updates contained in the current i-interpretation. With respect to these alternatives, purification does not loose any intensional knowledge specified by the user.

In the sequel, we generically use the term "rules" to refer to both active and purified rules. Notice that Definitions 9, 10, and 11, given a set of rules, only take into account active rules, while subsequent ones work on both kinds of rules.

Now, suppose that given an i-interpretation, more than one rule is fireable. It could happen that the actions to be executed are in conflict: some rules want to add a certain atom and others want to remove it from the i-interpretation. In order to obtain a new consistent i-interpretation we prevent one set of rules from firing: if we choose to insert the atom, only rules adding it are triggered, otherwise only rules removing it are.

Formally, we first define the notion of *conflict*, which consists of an atom and the sets of rules inserting and removing it.

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. $\exists\ r = l_1, \ldots, l_n \rightarrow u_1, \ldots, u_k,\ \ r' = l'_1, \ldots, l'_m \rightarrow u'_1, \ldots, u'_s,\ \ r, r' \in P$, *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*. To solve conflicts, a parametric conflict resolution policy is introduced.

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 $\mathsf{sel}(\mathrm{EDB}, P, I, (a, \mathsf{ins}, \mathsf{del}))$ is to choose whether 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.

Gottlob et al. [23] present a number of common policies, and discuss their advantages and disadvantages. We point out here just some 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 [25], Postgres [37] and Starburst [42], assumes that each rule has a (static or dynamic) priority associated with it; sel returns insert or delete as needed to execute the update requested by the highest-priority rule (in our framework, this can be obtained by looking up priorities in P). Other policies, like voting

schemas or user queries, are also reasonable, but the final choice is better left to the particular application.

Based on the result of the sel policy, we prevent the rule instances in one of the two sets of a conflict from firing, by blocking them according to the following definition.

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*

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

We define then

$$\mathsf{blocked}(EDB, P, I, \mathsf{sel}) = \left(\bigcup_{x \in X} x\right) \bigcup \left(\bigcup_{y \in Y} y\right).$$

□

We block an entire rule instance, rather than a single update, so that the set of updates requested by the same rule instance exhibits an atomic behavior: either all the updates in the set are executed, or no update at all. This avoids the risk of bringing a database in an inconsistent state due to partially-executed actions.

Example 8. Let us consider the Active-U-Datalog program in Example 5. Let R be the set of rules consisting of $\widehat{\mathrm{IDB}}$ (Example 7) and AR.

```
EDB:    v(a,a).  v(a,b).  r(b).
R:   r1 q(X) → p(X).
     r2 v(X,X) → q(X).
     r3 r(X) → q(X).
     r4 -r(X),r(Y) → +v(X,Y).
     r5 -r(X),v(X,Y) → -v(X,Y).
     r6 -r(X),q(X) → +v(X,X).
```

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

$$\mathsf{conflicts}(R, I) = \{\ (v(a,a), \{(r_6, \{X \leftarrow a\})\}, \{(r_5, \{X \leftarrow a, Y \leftarrow a\})\}),$$
$$(v(a,b), \{(r_4, \{X \leftarrow a, Y \leftarrow b\})\}, \{(r_5, \{X \leftarrow a, Y \leftarrow b\})\}) \quad \}.$$

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

$$\mathsf{sel}(\mathrm{EDB}, R, I, (v(a,a), \{(r_6, \{X \leftarrow a\})\}, \{(r_5, \{X \leftarrow a, Y \leftarrow a\})\})) = \mathsf{insert}$$
$$\mathsf{sel}(\mathrm{EDB}, R, I, (v(a,b), \{(r_4, \{X \leftarrow a, Y \leftarrow b\})\}, \{(r_5, \{X \leftarrow a, Y \leftarrow b\})\})) = \mathsf{delete}$$

Then, we have that

$$\mathsf{blocked}(\mathrm{EDB}, R, I, \mathsf{sel}) = \{(r_5, \{X \leftarrow a, Y \leftarrow a\}), (r_4, \{X \leftarrow a, Y \leftarrow b\})\}.$$

◇

Using the concepts defined above, the immediate consequence operator on i-interpretations can be defined as follows:

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* $\{u_1\theta, \ldots, u_k\theta\} \subseteq U$.

□

The operator $\Gamma_{P,B}$ is monotonic on the lattice $(\wp(\mathcal{B}^{\pm}), \subseteq)$, hence continuous because $\wp(\mathcal{B}^{\pm})$ is finite. The proof is analogous to that of Theorem 1.

The main difference of the above operator with respect to the traditional immediate consequence operator of logic programming is that it may happen that some of the rules are not fired even if their body is valid.

Example 9. Maintaining the definitions of Example 8, we compute the value of $\Gamma_{R,\emptyset}(I)$, i.e. when no rule is blocked.

$$\Gamma_{R,\emptyset}(I) = \{\ v(a,a),\ v(a,b),\ r(b),\ -r(a),\ q(a),\ p(a),\ q(b), +v(a,b),\ -v(a,a),\ -v(a,b),\ +v(a,a)\ \}$$

The resulting set is inconsistent. However, if we compute $\Gamma_{R,B}(I)$ where B is the set of blocked rules in Example 8, i.e. $B = \mathsf{blocked}(\mathrm{EDB}, R, I, \mathsf{sel})$, then we obtain the following consistent i-interpretation:

$$\Gamma_{R,B}(I) = \{\ v(a,a), v(a,b), r(b), -r(a), q(a), p(a), q(b), -v(a,b), +v(a,a)\ \}$$

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

In general, the application of the function $\Gamma_{P,B}$ to a consistent i-interpretation does not return a consistent i-interpretation, as shown in the previous example. Therefore, we cannot compute the semantics of P as the least fixpoint of $\Gamma_{P,B}$. We must instead appropriately select rules, that is we must build a set of blocked rules B such that the least fixpoint of $\Gamma_{P,B}$ is consistent. Thus, instead of dealing with i-interpretations, the notion of bi-structures is introduced, as in [23], in order to take into account blocked rules.

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

$$\langle B, I\rangle \prec \langle B', I'\rangle \quad \stackrel{\text{def}}{\Leftrightarrow} \quad \begin{cases} B \subset B' & \text{or} \\ B = B' \text{ and } I \subset I' \end{cases}$$

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

Bi-structures ordered by $\preceq$ form a complete partial order because there are only finitely many pairs $\langle B, I\rangle$. On this domain we can define an operator having a fixpoint, that is used as model of the active part.

Definition 14 (Δ operator). *Given a set of rules P, a bi-structure $\langle B, I\rangle$ and a conflict resolution policy* sel*, we define*

$$\Delta_{P,\mathsf{sel}}(\langle B, I\rangle) = \begin{cases} \langle B, \Gamma_{P,B}(I)\rangle & \text{if } \Gamma_{P,B}(I) \text{ is consistent;} \\ \langle B \cup \mathsf{blocked}(I^{\perp}, P, I, \mathsf{sel}), I^{\perp}\rangle & \text{otherwise.} \end{cases}$$

where $I^{\perp}$ is the set of the extensional atoms of I, i.e. $I^{\perp} = \{p(\tilde{t}) \in I \mid p \in \Pi^e\}$. □

The definition of Δ we give here differs from the original in [23] because the set of rules P contains not only rules with updates in the right hand side (properly active rules) but also purified rules that allow us to derive intensional knowledge rather than new updates. Notice that this extension does not affect the consistency of i-interpretations, since the purified rules can only add (Π^i, Σ, V)-atoms to the i-interpretation.

The intuitive idea of the Δ operator is that, if no conflict arises, Δ does not change the blocked rules set B, and only the i-interpretation of the bi-structure is changed by adding the immediate consequences of the non blocked rules. On the other hand, as soon as a conflict arises, the conflict is solved via the resolution policy sel and all blocked rule instances are collected. Then the computation of Δ is started again from the i-interpretation $I^{\perp}$ with the augmented set of blocked rules. The i-interpretation $I^{\perp}$ represents the set of the extensional atoms of the database, and we have to resort to it to be sure that the starting point of the new computation does not contain atoms whose validity depends on actions of rule instances that are now blocked. As remarked in [23] for the PARK semantics, this semantics may be viewed as a smooth cycle integrating inflationary fixpoint computation [27] and conflict resolution policies.

Remark 1. The Δ operator is not monotonic (thus not continuous) since it depends on the function blocked which, in turns, depends on an arbitrary function sel. In fact, consider again Example 8. Suppose we add the following active rule to the program:

$$r_7 \quad \texttt{-r(X),q(Y),r(Y)} \rightarrow \texttt{+v(X,Y)}$$

Consider the i-interpretation $I' = I \cup \{q(b)\}$. The set $\mathsf{conflicts}(R, I')$ is the following:

$$\begin{aligned} \mathsf{conflicts}(R, I') = \{\ & (v(a,a), \{(r_6, \{X \leftarrow a\})\}, \{(r_5, \{X \leftarrow a, Y \leftarrow a\})\}), \\ & (v(a,b), \{(r_4, \{X \leftarrow a, Y \leftarrow b\}), (r_7, \{X \leftarrow a, Y \leftarrow b\})\}, \\ & \{(r_5, \{X \leftarrow a, Y \leftarrow b\})\})\ \}. \end{aligned}$$

The occurrence of the atom $q(b)$ in I' triggers rule r_7.

Let us suppose also that sel is a rule priority conflict resolution policy, and that rules are prioritized according to their position in the program. Therefore,

sel chooses to insert (delete) the atom, object of the conflict, if in the set of insertion (deletion) rules we find the rule with the highest priority.

$$\mathsf{sel}(\text{EDB}, R, I', (v(a,a), \{(r_6, \{X \leftarrow a\})\}, \{(r_5, \{X \leftarrow a, Y \leftarrow a\})\})) = \mathsf{insert}$$
$$\mathsf{sel}(\text{EDB}, R, I', (v(a,b), \{(r_4, \{X \leftarrow a, Y \leftarrow b\}), (r_7, \{X \leftarrow a, Y \leftarrow b\})\}, \{(r_5, \{X \leftarrow a, Y \leftarrow b\})\})) = \mathsf{insert}$$

It is worth noticing that in the second conflict now the policy chooses to insert the atom $v(a,b)$ because the rule r_7 has the highest priority.
Then, we have that

$$\mathsf{blocked}(\text{EDB}, R, I', \mathsf{sel}) = \{(r_5, \{X \leftarrow a, Y \leftarrow a\}), (r_5, \{X \leftarrow a, Y \leftarrow b\})\}.$$

Comparing the set $\mathsf{blocked}(\text{EDB}, R, I, \mathsf{sel})$ and $\mathsf{blocked}(\text{EDB}, R, I', \mathsf{sel})$, we observe that there is no relation of containment between them. Therefore blocked is not monotonic. As consequence,

$$\begin{array}{ccc} \langle \mathsf{blocked}(I^{\perp}, R, I, \mathsf{sel}), I^{\perp} \rangle & \not\preceq & \langle \mathsf{blocked}(I'^{\perp}, R, I', \mathsf{sel}), I'^{\perp} \rangle \\ \| & & \| \\ \Delta_{P,\mathsf{sel}}(\langle \emptyset, I \rangle) & \not\preceq & \Delta_{P,\mathsf{sel}}(\langle \emptyset, I' \rangle) \end{array}$$

where $I^{\perp} = I'^{\perp} = \text{EDB}$. ◇

Since Δ is not continuous, we cannot prove that Δ has a fixpoint by using the fixpoint theorem. However, the Δ operator is *growing*, therefore, for the finiteness of the domain, a fixpoint is reached by iterating Δ a finite number of steps.

Lemma 1. *Given a set of rules P, a conflict resolution policy* sel*, a bi-structure $\mathcal{A} = \langle B, I \rangle$, the following statements hold:*

1. $\mathcal{A} \preceq \Delta_{P,\mathsf{sel}}(\mathcal{A})$,
2. *there exists k such that $\Delta^k_{P,\mathsf{sel}}(\mathcal{A})$ is a fixpoint of $\Delta_{P,\mathsf{sel}}$.*

Proof. We proceed along the lines of Theorem 4.1 in [23].

1. Let $\Delta_{P,\mathsf{sel}}(\mathcal{A}) = \langle B', I' \rangle$. If I' is consistent, then $B' = B$ and $I' = \Gamma_{P,B}(I) \supseteq I$ by definition of Γ; hence $\langle B, I \rangle \preceq \langle B', I' \rangle$. If instead I' is not consistent, then $B' = B \cup \mathsf{blocked}(\text{EDB}, P, I, \mathsf{sel}) \supseteq B$ and so we have again $\langle B, I \rangle \preceq \langle B', I' \rangle$.
2. By statement 1, for all natural numbers n, we have

$$\Delta^n_{P,\mathsf{sel}}(\mathcal{A}) \preceq \Delta_{P,\mathsf{sel}}(\Delta^n_{P,\mathsf{sel}}(\mathcal{A})).$$

Hence, $\{\Delta^i_{P,\mathsf{sel}}(\mathcal{A})\}_{i \in \mathbb{N}}$ is a chain in the cpo of the bi-structures. Since such a cpo is finite, every chain consists of a finite number of elements. Therefore

$$\exists k.\ \forall n \geq k.\ \Delta^n_{P,\mathsf{sel}}(\mathcal{A}) = \Delta^{n+1}_{P,\mathsf{sel}}(\mathcal{A}).$$

We can conclude that $\Delta^k_{P,\mathsf{sel}}(\mathcal{A})$ is a fixpoint of $\Delta_{P,\mathsf{sel}}$. □

The next theorem shows that we can find a set of blocked rules B such that the least fixpoint of $\Gamma_{P,B}$ is a consistent i-interpretation.

Theorem 2. *Given a set of rules P, a conflict resolution policy* sel*, a bi-structure $\mathcal{A} = \langle B, I\rangle$, with I a set of ground extensional atoms, there exists k such that $\Delta^k_{P,\mathsf{sel}}(\mathcal{A})$ is a fixpoint of $\Delta_{P,\mathsf{sel}}$ and if $\Delta^k_{P,\mathsf{sel}}(\mathcal{A}) = \langle B', I'\rangle$, then $I' = \mathsf{lfp}_I(\Gamma_{P,B'})$[5] and I' is consistent.*

Proof. First we notice that if $\Delta_{P,\mathsf{sel}}(\langle B_1, I_1\rangle) = \langle B_2, I_2\rangle$ then ${I_1}^{\perp} = {I_2}^{\perp}$, that is the set of ground extensional atoms is not modified by Δ. This follows from the definition of $\Delta_{P,\mathsf{sel}}$ and from the fact that $\Gamma_{P,B}$ can add only intensional and update atoms to an i-interpretation. As a consequence, for all natural numbers n, if $\Delta^n_{P,\mathsf{sel}}(\langle B_1, I_1\rangle) = \langle B_{n+1}, I_{n+1}\rangle$ then ${I_1}^{\perp} = {I_{n+1}}^{\perp}$.
Since the i-interpretation of $\mathcal{A}$ only consists of extensional atoms, then $I^{\perp} = I$.

By Lemma 1, there exists k such that $\Delta^k_{P,\mathsf{sel}}(\mathcal{A})$ is a fixpoint of $\Delta_{P,\mathsf{sel}}$. By definition of Δ and by the above remark, there exists $i \leq k$ such that $\Delta^i_{P,\mathsf{sel}}(\mathcal{A}) = \langle B', I\rangle$. Then for all j such that $i \leq j \leq k$, we have $\Delta^j_{P,\mathsf{sel}}(\mathcal{A}) = \langle B', \Gamma^{j-i}_{P,B'}(I)\rangle$, because B' does not increase. Since $\Delta^k_{P,\mathsf{sel}}(\mathcal{A}) = \langle B', I'\rangle = \langle B', \Gamma^{k-i}_{P,B'}(I)\rangle$ is a fixpoint, then $\langle B', \Gamma^{k-i}_{P,B'}(I)\rangle = \langle B', \Gamma^{k-i+1}_{P,B'}(I)\rangle$. Therefore $I' = \mathsf{lfp}_I(\Gamma_{P,B'})$ and by definition of Δ, the set I' is consistent (otherwise the set of blocked rules would be augmented). □

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 or Abort).

Definition 15 (Observables). *An* observable *is a triple* $\langle$Ans, *EDB*, 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. □

The semantics of the previous steps (Sections 4.1 and 4.2) does not include the execution of the collected updates, neither considers the transactional behavior. We now define a function which, given an i-interpretation and the current state of the system, returns the next state 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

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

□

[5] $\mathsf{lfp}_I(f)$ denotes the least fixpoint of f which is greater than or equal to I.

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

Definition 17 (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*

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

where the function $\mathcal{S}_{IDB,AR,\mathsf{sel}}(T) : \mathsf{Oss} \to \mathsf{Oss}$ is defined as follows:
If T is a simple transaction, then

$$\mathcal{S}_{IDB,AR,\mathsf{sel}}(T)(\langle \alpha, \varepsilon, \rho\rangle) = \begin{cases} \langle \emptyset, \varepsilon, \mathsf{Abort}\rangle & \text{if } \rho = \mathsf{Abort} \\ \langle \mathsf{Ans}, \mathsf{incorp}(I, \varepsilon), \mathsf{Commit}\rangle & \text{if } \rho \neq \mathsf{Abort} \text{ and } \overline{U} \text{ is ground} \\ \langle \emptyset, \varepsilon, \mathsf{Commit}\rangle & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} \mathsf{Ans} &= \{b_j \mid \langle b_j, \overline{u}_j\rangle \in \mathsf{Set}(T, P)\} \\ \overline{U} &= \bigcup \{\overline{u}_j \mid \langle b_j, \overline{u}_j\rangle \in \mathsf{Set}(T, P)\} \\ \langle B, I\rangle &= \Delta^{\omega}_{\Xi,\mathsf{sel}}(\langle \emptyset, \varepsilon\rangle) \\ \Xi &= (\widehat{IDB} \cup AR) \cup \{ \rightarrow +a \mid +a \in \overline{U}\} \cup \{ \rightarrow -a \mid -a \in \overline{U}\}. \end{aligned}$$

If T is a complex transaction $T_1; \ldots; T_k$ $(k \geq 2)$, then

$$\mathcal{S}_{IDB,AR,\mathsf{sel}}(T_1; \ldots; T_k)(Oss) \;=\; \mathcal{S}_{IDB,AR,\mathsf{sel}}(T_2; \ldots; T_k)(\mathcal{S}_{IDB,AR,\mathsf{sel}}(T_1)(Oss))$$

where $T_1, \ldots, T_k$ are simple transactions. □

To compute the semantics of a simple transaction, first we build the set of answers in the marking phase (Definition 6). This step returns a set of bindings for the variables of the transaction (Ans) and a set of updates ($\overline{U}$) which are requested but which will not necessarily be executed. Then we gather rules in order to apply the Δ operator (Definition 14).

Such a set of rules (Ξ) contains the purified rules from the intensional database ($\widehat{\mathrm{IDB}}$), the rules in AR and the updates requested from the deductive part ($\overline{U}$), represented as rules with neither event nor condition. Therefore we keep in this set also the intensional knowledge of the program to verify the condition part of the active rules; the updates in $\overline{U}$ become the initial events to which the active rules in AR have to react.

To obtain the set of updates to be executed, we apply the Δ operator starting from an empty set of blocked rule instances and from the extensional database as initial i-interpretation ($\langle \emptyset, \mathrm{EDB}\rangle$). Theorem 2 assures that a fixpoint of $\Delta_{\Xi,\mathsf{sel}}$ is reached in a finite number of steps by computing the approximations

$\Delta^{i}_{\Xi,\mathsf{sel}}(\langle\emptyset, \mathrm{EDB}\rangle)$ and that the i-interpretation, I, in the resulting bi-structure is consistent. Finally, the new state of the database is computed by incorporating the updates belonging to I in the current state of the database (Definition 16).

The semantics of a complex transaction is simply given by the sequential composition of the semantics of its components. The state of the system is updated after each simple transaction. Besides, this semantics discards the answers to all but the last simple transaction, to stay close to the approach in [9].

It is worth remarking that our semantics, unlike the semantics developed in [9], never generates aborts because of conflicts, thus augmenting the successful computations. This is possible because of the presence of the active component of our language, and especially of the conflict resolution policy. However, we propagate extra-semantics Aborts, e.g., those generated by media failures.

Moreover, if non-ground updates are requested, we do not abort, in keeping with [5,9], but we commit discarding all changes to the state of the system, to point out that we are not able to decide which updates have to be executed in order to guarantee safety.

As already noted, the answer set Set and the Δ fixpoint are computed in a finite number of steps, hence $Sem_{P,\mathsf{sel}}$ is computed in a finite number of steps. Actually, it is computable in polynomial time in the size of the state of the system when the conflict resolution policy sel is implementable in polynomial time. This result follows from the fact that the PARK semantics for active rules as well as the U-Datalog semantics guarantee polynomial complexity [7,23], when the condition above is satisfied.

Example 10. Let us consider again the database introduced in Example 4. We want to extend it by adding information about the school library, namely available books and loans. We assume that it is the library policy to claim back books on loan as soon as the borrower passes the exam the book is associated with, and that further loans are to be denied as long as books claimed are not given back to the library. However, a student can escape this policy, and obtain an extension of a loan, by presenting a written request to this end from his tutor.

The following database P, building on that presented in Example 4, satisfies our requirements:

```
EDB: student(frank).              student(mary).
     exam(engl).                  exam(phys).
     book(othello,engl).          book(principia,phys).
     book(quanta,phys).
     onloan(quanta,frank).        onloan(principia,frank).


IDB: pass(S,E) ←student(S), exam(E), +passed(S,E).
     leave(S) ←student(S), −student(S).
     denyloan(B,S) ←request(X,S), book(X,E).
     denyloan(B,S) ←onloan(B,X), student(S).
     return(B,S) ←onloan(B,S), −onloan(B,S).
     extend(B) ←onloan(B,S), −request(B,S).
```

```
AR:  −student(S), passed(S,E) →−passed(S,E).
     −student(S), onloan(B,S) →+request(B,S).
     −onloan(B,S), request(B,S) →−request(B,S).
     +passed(S,E), onloan(B,S), book(B,E) →+request(B,S).
```

When Frank passes physics, he obtains from his tutor an extension on the loan of the *Quanta*. We record this information in the database by executing a transaction `pass(frank,phys), extend(quanta)`.

The first rule in IDB causes the insertion of `passed(frank,phys)` in the database; this, in turn, causes the firing of the last active rule, whose conditions are met by `onloan(principia,frank)` and `book(principia,phys)`, and by `onloan(quanta,frank)` and `book(quanta,phys)`, so the insertion of `request(principia,frank)` and `request(quanta,frank)` in the database is requested.

However, the last rule in the IDB, due to the presence of `extend(quanta)` in the transaction, asks for the removal of `request(quanta,frank)` from the database, so a conflict arises. We suppose that the database is using an inertial conflict resolution policy, so the conflict is solved by preventing the insertion of `request(quanta,frank)` (but not of `request(principia,frank)`).

If Frank asks another loan, the third rule in the IDB will deny the loan due to the presence of `request(principia,frank)` in the database. Frank will have to return the *Principia* first, as expressed by the query `return(principia,frank)`. In response to this query, the loan record `onloan(principia,frank)` will be removed, and this in turn will fire the third active rule, thus removing `request(principia,frank)` from the database. Frank's requests for other books on loan will then be accepted.

Let us now verify this behavior in terms of our semantics. The first step consists of computing the fixpoint semantics of the deductive part, as shown in Definition 3.

$$\begin{aligned}\mathcal{F}(P) = \{\ & \mathit{student}(\mathcal{S})^6,\quad \mathit{exam}(\mathcal{E}),\quad \mathit{book}(\mathit{othello},\mathit{engl}),\quad \mathit{book}(\mathit{quanta},\mathit{phys}),\\ & \mathit{book}(\mathit{principia},\mathit{phys}),\quad \mathit{onloan}(\mathit{quanta},\mathit{frank}),\\ & \mathit{onloan}(\mathit{principia},\mathit{frank}),\quad \mathit{pass}(\mathcal{S},\mathcal{E}) \leftarrow\!+\mathit{passed}(\mathcal{S},\mathcal{E}),\\ & \mathit{leave}(\mathcal{S}) \leftarrow\!-\mathit{student}(\mathcal{S}),\quad \mathit{denyloan}(\mathit{quanta},\mathcal{S}),\\ & \mathit{denyloan}(\mathit{principia},\mathcal{S}),\\ & \mathit{return}(\mathit{quanta},\mathit{frank}) \leftarrow\!-\mathit{onloan}(\mathit{quanta},\mathit{frank}),\\ & \mathit{return}(\mathit{principia},\mathit{frank}) \leftarrow\!-\mathit{onloan}(\mathit{principia},\mathit{frank}),\\ & \mathit{extend}(\mathit{quanta}) \leftarrow\!-\mathit{request}(\mathit{quanta},\mathit{frank}),\\ & \mathit{extend}(\mathit{principia}) \leftarrow\!-\mathit{request}(\mathit{principia},\mathit{frank})\quad \}\end{aligned}$$

The answer to the transaction $T = \mathtt{pass}(\mathtt{frank},\mathtt{phys}), \mathtt{extend}(\mathtt{quanta})$, by Definition 6, is thus

$$\mathsf{Set}(T,P) = \{\langle \emptyset, \{+\mathit{passed}(\mathit{frank},\mathit{phys}), -\mathit{request}(\mathit{quanta},\mathit{frank})\}\rangle\}.$$

[6] As a shorthand, in the following we write $\mathcal{S}$ to stand for all elements in {mary, frank}, and $\mathcal{E}$ for all elements in {engl, phys}.

Next, we have to compute the active part semantics. The purified database, according to Definition 8, is

```
IDB̂: student(S), exam(E) →pass(S,E).
     student(S) →leave(S).
     request(X,S), book(X,E) →denyloan(B,S).
     onloan(B,X), student(S) →denyloan(B,S).
     onloan(B,S) →return(B,S).
     onloan(B,S) →extend(B).
```

Thus, we have

$$\Xi = \widehat{\mathrm{IDB}} \cup \mathrm{AR} \cup \{\rightarrow + passed(frank, phys), \rightarrow - request(quanta, frank)\}.$$

The computation proceeds by computing the fixpoint of $\Delta_{\Xi,\mathsf{sel}}(\langle\emptyset, \mathrm{EDB}\rangle)$ as follows:

$$\begin{aligned}
&\Delta^0_{\Xi,\mathsf{sel}}(\langle\emptyset, \mathrm{EDB}\rangle) = \langle\emptyset, I_0 = \mathrm{EDB}\rangle \\
&\Delta^1_{\Xi,\mathsf{sel}}(\langle\emptyset, \mathrm{EDB}\rangle) = \langle\emptyset, I_1 = I_0 \cup \{pass(\mathcal{S}, \mathcal{E}), leave(\mathcal{S}), \\
&\qquad\qquad denyloan(quanta, \mathcal{S}), denyloan(principia, \mathcal{S}), \\
&\qquad\qquad return(quanta, frank), return(principia, frank), \\
&\qquad\qquad extend(quanta), extend(principia), \\
&\qquad\qquad +passed(frank, phys), -request(quanta, frank)\}\rangle.
\end{aligned}$$

Now, in the computation of $\Gamma_{\Xi,\emptyset}(I_1)$ a conflict arises:

$$\begin{aligned}
\Gamma_{\Xi,\emptyset}(I_1) = I_2 = \{\ & \textit{student(}\mathcal{S}\textit{), exam(}\mathcal{E}\textit{), book(othello,engl), book(quanta,phys),} \\
& \textit{book(principia,phys), onloan(quanta,frank),} \\
& \textit{onloan(principia,frank), pass(}\mathcal{S}\textit{,}\mathcal{E}\textit{), leave(}\mathcal{S}\textit{),} \\
& \textit{denyloan(quanta,}\mathcal{S}\textit{), denyloan(principia,}\mathcal{S}\textit{),} \\
& \textit{return(quanta,frank), return(principia,frank),} \\
& \textit{extend(quanta), extend(principia), +passed(frank,phys),} \\
& -\mathbf{request(quanta,frank)}\textit{, +request(principia,frank),} \\
& +\mathbf{request(quanta,frank)} \quad \}
\end{aligned}$$

The conflict is

$$\begin{aligned}
c = (request(quanta, frank), \{(r_1, \{S{\leftarrow}frank, E{\leftarrow}phys, B{\leftarrow}quanta\})\}, \\
\{(r_2, \emptyset)\} \quad)
\end{aligned}$$

where r_1 is the rule

```
+passed(S,E), onloan(B,S), book(B,E) →+request(B,S).
```

from AR, while r_2 is the rule

```
→−request(quanta,frank).
```

inserted in Ξ as result of the deductive part semantics. Since we are using an inertial conflict resolution policy sel, and $request(quanta, frank)$ is not in EDB, we have that

$$\mathsf{sel}(\mathrm{EDB}, \Xi, I_2, c) = \mathsf{delete}$$

and thus

$$B' = \mathsf{blocked}(\mathrm{EDB}, \Xi, I_2, \mathsf{sel}) = \{(r_1, \{S{\leftarrow}frank, E{\leftarrow}phys, B{\leftarrow}quanta\})\}.$$

The computation of $\Delta^{\omega}_{\Xi,\mathsf{sel}}$ continues without further conflicts, and the fixpoint is:

$$\begin{aligned}\Delta^{4}_{\Xi,\mathsf{sel}}(\langle\emptyset, \mathrm{EDB}\rangle) = \langle B', I_4 = \mathrm{EDB} \cup \{&pass(\mathcal{S}, \mathcal{E}), leave(\mathcal{S}),\\ &denyloan(quanta, \mathcal{S}), denyloan(principia, \mathcal{S}),\\ &return(quanta, frank), return(principia, frank),\\ &extend(quanta), extend(principia),\\ &+passed(frank, phys), -request(quanta, frank),\\ &+request(principia, frank)\}\rangle\end{aligned}$$

The new database after having incorporated the updates is EDB′.

$$\begin{aligned}\mathrm{EDB}' = \mathsf{incorp}(I_4, \mathrm{EDB}) = \{\ &\textit{student(mary), student(frank), exam(engl),}\\ &\textit{exam(phys), passed(frank,phys),}\\ &\textit{book(othello,engl), book(quanta,phys),}\\ &\textit{book(principia,phys), onloan(quanta,frank),}\\ &\textit{onloan(principia,frank), request(principia,frank)}\\ &\}\end{aligned}$$

We call $P' = \mathrm{IDB} \cup \mathrm{EDB}' \cup \mathrm{AR}$.

If now Frank asks to borrow another book, say *Othello*, the system checks if the loan is permitted before giving its ok, i.e. $T' = \texttt{denyloan}(\texttt{othello}, \texttt{frank})$. The deductive part semantics returns the following set:

$$\begin{aligned}\mathcal{F}(P') = \mathrm{EDB}' \cup \{\ &\textit{pass(}\mathcal{S}\textit{,}\mathcal{E}\textit{)}{\leftarrow}{+}\textit{passed(}\mathcal{S}\textit{,}\mathcal{E}\textit{)}, \quad \textit{leave(}\mathcal{S}\textit{)}{\leftarrow}{-}\textit{student(}\mathcal{S}\textit{)},\\ &\textit{denyloan(quanta,}\mathcal{S}\textit{)}, \quad \textit{denyloan(principia,}\mathcal{S}\textit{)},\\ &\textit{denyloan(othello,frank)},\\ &\textit{return(quanta,frank)}{\leftarrow}{-}\textit{onloan(quanta,frank)},\\ &\textit{return(principia,frank)}{\leftarrow}{-}\textit{onloan(principia,frank)}\\ &\textit{extend(quanta)}{\leftarrow}{-}\textit{request(quanta,frank)},\\ &\textit{extend(principia)}{\leftarrow}{-}\textit{request(principia,frank)} \quad \}\end{aligned}$$

This time no updates are collected from the deductive part semantics, so the active part semantics immediately converges with an empty set of updates, thus leaving the database in the same state as before, i.e., EDB′. Since the query was answered positively, Frank will not be lent *Othello*. To obtain it,

he will have to return Sir Isaac's masterwork, by executing the transaction `return(principia, frank)`. In order to solve this transaction, the deductive part semantics computes

$$\mathsf{Set}(\texttt{return}(\texttt{principia},\texttt{frank}), P') = \{\langle \emptyset, \{-onloan(principia, frank)\}\rangle\}$$

and we have

$$\Xi'' = \widehat{\mathrm{IDB}} \cup \mathrm{AR} \cup \{\rightarrow - onloan(principia, frank)\}.$$

The fixpoint of $\Delta^{\omega}_{\Xi'',\mathsf{sel}}(\langle \emptyset, \mathrm{EDB}'\rangle)$ is reached in three steps, without any conflict, and consists of an empty set of blocked rules and of the following i-interpretation:

$$\begin{aligned} I'' = \mathrm{EDB}' \cup \{ \; & \mathit{pass}(\mathcal{S},\mathcal{E}),\ \mathit{leave}(\mathcal{S}),\ \mathit{denyloan}(\mathit{othello},\mathit{frank}), \\ & \mathit{denyloan}(\mathit{quanta},\mathcal{S}),\ \mathit{denyloan}(\mathit{principia},\mathcal{S}), \\ & \mathit{return}(\mathit{quanta},\mathit{frank}),\ \mathit{return}(\mathit{principia},\mathit{frank}),\ \mathit{extend}(\mathit{quanta}), \\ & \mathit{extend}(\mathit{principia}),\ \mathit{-onloan}(\mathit{principia},\mathit{frank}), \\ & \mathit{-request}(\mathit{principia},\mathit{frank}) \quad \} \end{aligned}$$

and thus the new extensional database EDB″, after updates incorporation, is

$$\begin{aligned} \mathrm{EDB}'' = \mathsf{incorp}(I'', \mathrm{EDB}') = \{ \; & \mathit{student}(\mathit{mary}),\ \mathit{student}(\mathit{frank}),\ \mathit{exam}(\mathit{engl}), \\ & \mathit{exam}(\mathit{phys}),\ \mathit{passed}(\mathit{frank},\mathit{phys}), \\ & \mathit{book}(\mathit{othello},\mathit{engl}),\ \mathit{book}(\mathit{principia},\mathit{phys}), \\ & \mathit{book}(\mathit{quanta},\mathit{phys}),\ \mathit{onloan}(\mathit{quanta},\mathit{frank}) \quad \} \end{aligned}$$

Now, Frank has no longer pending requests from the library, hence he can borrow other books.

◇

5 Conclusions and Future Work

In this paper we have extended U-Datalog, in a conservative way, with support for active rules. The resulting language, called Active-U-Datalog, provides a uniform logical representation of queries, transactions, and active behavior. The semantics extends the PARK semantics to deal not only with active rules but also with transactions. In particular, updates generated by deductive rules and updates generated by active rules are treated in a uniform way. Similarly to the PARK semantics, a flexible management of conflicts is also provided. Thus, our approach can also be seen as an extension of U-Datalog to deal with multiple policies for conflict resolution. As far as we know, no other approach of this kind has been proposed yet in the context of deductive databases.

This work can be extended in several ways. A first important question is related to the definition and analysis of properties concerning distributed query, transaction and active rule execution. Some preliminary results about the analysis of U-Datalog programs have been presented in [6]. Active rules offer an easy

mechanism to express system reactions depending on the generation of specific events; while only update requests were considered as events in this paper, the extension to other classes of events seems promising and relevant for practical applications. Also, a better handling of negation both in the deductive and in the active part is needed; work is currently in progress on this issue. A formal analysis of the expressive power of Active-U-Datalog with respect to U-Datalog is another important topic that should be investigated.

An implementation of Active-U-Datalog could be obtained by extending the implementation of U-Datalog, described in [8], according the the semantics given in this paper. Since Active-U-Datalog is conservative with respect to U-Datalog, such an extension does not seem difficult.

Acknowledgements. The authors wish to thank the anonymous referees for their valuable suggestions leading to significant improvements of this paper.

References

1. S. Abiteboul and V. Vianu. A Transaction Language Complete for Database Updates and Specification. In *Proc. of the Seventh ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 260–268, 1987.
2. S. Abiteboul and V. Vianu. Procedural and Declarative Database Update Languages. In *Proc. of the Eighth ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 240–250, 1988.
3. ANSI TC X3H2 and ISO/IEC JTC 1/SC 21/WG 3. Master Index for SQL and all its Parts, March 1966. Document X3H2-96-066 DBL:MCI-011.
4. ASK Computer Co. *INGRES/SQL Reference Manual*.
5. E. Bertino, G. Guerrini, and D. Montesi. Toward Deductive Object Databases. *Theory and Practice of Object Systems*, 1(1):19–39, 1995.
6. E. Bertino and B. Catania. Static Analysis of Intensional Databases in U-Datalog. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems* (PODS'96), pages 201–212, 1996.
7. E. Bertino and B. Catania. The Expressive Power of U-Datalog Programs. In Preparation, 1998.
8. E. Bertino, B. Catania, G. Guerrini, M. Martelli, and D. Montesi. A Bottom-Up Interpreter for a Database Language with Updates and Transactions. In *Proc. of the Joint Conference on Declarative Programming*, Vol. II, pages 206–220, Peniscola, Spain, 1994.
9. E. Bertino, M. Martelli, and D. Montesi. Transactions and Updates in Deductive Databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):784-797, 1997.
10. A.J. Bonner and M. Kifer. An Overview of Transaction Logic. *Theoretical Computer Science*, 133(2):205–265, 1994.
11. S. Castagna, G. Guerrini, D. Montesi, and G. Rodriguez. Design and Implementation for the Active Rule Language of Chimera. In *Proc. of the Int. Conf. and Workshop on Database and Expert Systems*, pages 45–54, 1995.
12. S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, 1990.

13. S. Ceri and J. Widom. Deriving Incremental Production Rules for Deductive Data. *Information Systems*, 19(6):467–490, 1994.
14. 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, Massachusetts, August 1990.
15. W. Chen. Declarative Specification and Evaluation of Database Updates. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *LNCS 566: Proc. of the Int. Conf. on Deductive and Object Oriented Databases*, pages 147–166, 1991.
16. W. Chen. Programming with Logical Queries, Bulk Updates, and Hypothetical Reasoning. *IEEE Transactions on Knowledge and Data Engineering*, 9(4):587–599, 1997.
17. K.L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 56–65, 1996.
18. L.M.L. Delcambre and J.N. Etheredge. The Relational Procedural Language: A Production Language for Relational Databases. In *Proc. of the Second Int. Conf. on Expert Database Systems*, pages 333-351, 1988.
19. Digital Equipment Corporation. *Rdb/VMS - SQL Reference Manual*, November 1991.
20. A.A.A. Fernandes, M.H. Williams, and N.W. Patton. A Logic-Based Integration of Active and Deductive Databases. *New Generation Computing*, 15(2): 205–244, 1997.
21. N. Gehani and H.V. Jagadish. Ode as an Active Database: Constraints and Triggers. In *Proc. of the Seventeenth Int. Conf. on Very Large Data Bases*, pages 327–336, 1991.
22. V. Gervasi and A. Raffaetà. Integrating Active Rules in U-Datalog. In Moreno Falaschi, Marisa Navarro, and Alberto Policriti, editors, *Proc. of the APPIA-GULP-PRODE '97 Joint Conf. on Declarative Programming*, pages 117–128, 1997.
23. G. Gottlob, G. Moerkotte, and V.S. Subrahmanian. The PARK Semantics for Active Rules. In *LNCS 1057: Proc. of the Fifth Int. Conf. on Extending Database Technology*, pages 35–55, 1996.
24. J. Gray and A. Reuter. *Transaction Processing Concepts and Tecniques*. Morgan-Kaufmann, 1993.
25. E. Hanson. Rule Condition Testing and Action Execution in Ariel. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 49–58, 1992.
26. Kirkwood. *Sybase Architecture and Administration*. Prentice-Hall, 1993.
27. P. Kolaitis and C. Papadimitriou. Why Not Negation by Fixpoint? *Journal of Computer and System Sciences*, 43(1):125–144, 1991.
28. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
29. B. Ludäscher, W. May, and G. Lausen. Nested Transactions in a Logical Languages for Active Rules. In *LNCS 1154: Proc. of the Int. Workshop on Logic in Databases*, pages 197–222, 1996.
30. B. Ludäscher, U. Hamann, and G. Lausen. A Logical Framework for Active Rules. In *Proc. of the Seventh Int. Conf. on Management of Data (COMAD)*, 1995.
31. S. Manchanda. Declarative Expression of Deductive Database Updates. In *Proc. of the SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 93–100, 1989.
32. S. Manchanda and D.S. Warren. A Logic-Based Language for Database Updates. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 363–394. Morgan-Kaufmann, 1988.

33. D.R. McCarthy and U. Dayal. The Architecture of an Active Database Management System. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 215–224, 1989.
34. D. Montesi and R. Torlone. A Transaction Transformation Approach to Active Rule Processing. In *Proc. of the Eleventh Int. Conf. on Data Engineering*, pages 109–116, 1995.
35. S. Naqvi and S. Tsur. *A Logical Language for Data and Knowledge Bases*. Computer Science Press, 1989.
36. Oracle Corp. *Oracle 7 Server Concepts Manual, 7.3*. Oracle Corp., 1996.
37. 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 Int. Conf. on Management of Data*, pages 281–290, 1990.
38. M. Stonebraker and G. Kemnitz. The POSTGRES Next-Generation Database Management System. *Communications of the ACM*, 34(10):78–92, 1991.
39. L. Tanca. (Re-)Action in Deductive Databases. In *Proc. of the Second Int. Workshop on Intelligent and Cooperative Information Systems*, pages 55–61, 1990.
40. J. Widom, R.J. Cochrane, and B.G. Lindsay. Implementing Set-oriented Production Rules as an Extension to Starburst. In *Proc. of the Seventeenth Int. Conf. on Very Large Data Bases*, pages 275–285, 1991.
41. C.A. Wichert and B. Freitag. Capturing Database Dynamics by Deferred Updates. In *Proc. of the Int. Conf. on Logic Programming*, 1997.
42. J. Widom and S. Finkelstein. Set-oriented Production Rules in Relational Databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 259–270, 1990.
43. C. Zaniolo. A Unified Semantics for Active and Deductive Databases. In *Proc. of the First Int. Workshop on Rules in Database Systems*, pages 271–287, 1993.