Active Behaviour in Deductive Databases

30 March 1996

Fariba Sadri, Francesca Toni
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
{fs,ft}@doc.ic.ac.uk

Abstract

This paper reports work in progress on active and deductive databases.

Active rules are used for a variety of purposes, prominent amongst which are integrity enforcement, view maintenance and actions output to the environment.

In this paper we argue that a package of deductive databases, temporal reasoning, integrity enforcement and abduction captures and possibly extends the first two main uses of active rules. As a by-product, we obtain a declarative, logical semantics for active rules.

We propose embedding this package within an execution model, on the lines of recent proposals for formalisation of reactive agents, to capture the third main use of active rules.

1. Introduction

Active databases [B93, FWP95, HW93a, HW93b, MT95] are conventional databases (relational, deductive or object-oriented) enriched with active rules of the form[1]

on *events*, if *conditions* then *actions*

The procedural interpretation of such a rule is given as

when the *events* occur, the rule is triggered, its *conditions* are evaluated and, if these hold, the *actions* are executed.

This basic execution model is sometimes augmented with prioritisation procedures to determine in which order active rules whose *events* have occurred are to be triggered, and by execution modalities (deferred, detached, or immediate) that determine the temporal ordering of the actions with respect to the input event occurrences.

Active rules serve many, diverse purposes, the most important of which are integrity enforcement, view maintenance and actions output to the environment.

---

[1] Different active databases adopt different syntactic formulations of such rules, that are variants of the formulation given here.

Example 1.1

Given a personnel database defined in terms of the relations

emp(Employee, Department, Salary)
dep(Department)
works_overtime(Employee, Month, StartDate, EndDate)

the active rule

on delete(dep(D)), if emp(E,D,S) then delete(emp(E,D,S)) A1.1

enforces the integrity constraint that no employee can work in a non-existing department.

The active rule

on add(emp(E,D,S)) if D=toy then
add(works_overtime(E, december, 1, 24)) A1.2

maintains the view predicate *works_overtime(E, M, SD, Ed).*

The active rule

on clock-tick if date=31 March and tax-form(E,nil) then
print "E must fill in his/her tax-form" A1.3

generates an action external to the database, in reaction to the passage of time in the environment.

The kinds of *events*, *conditions* and *actions* that are allowed in active rules depend upon the formalism and upon the intended uses of these rules in the formalisms. Events can be categorised into database events (add, delete or replace tuples), and external events (taking place in the external environment and therefore outside the control of the database management system). Conditions can be queries over the database as well as comparisons between different states of the database (transition conditions). Actions can be database events such as adding, deleting or replacing tuples, aborting a transaction, or external events.

The relationship between active and deductive databases has been investigated by several researchers (e.g. W94, Z94, FWP95). Some [Z94] have developed a unifying framework for both under one semantics, and others [FWP95] have argued that they are essentially diverse concepts that cannot be unified profitably within a single semantics.

Our proposal is similar to that of Zaniolo [Z94] in its objective of unifying active and deductive databases within a uniform semantics, and to that of Fernandes et al [FWP95] in its use of the event calculus of Kowalski and Sergot [KS86] for representing events and time.

We argue that active rules can best be understood within a context that includes deductive, temporal databases together with static and transition constraints, ramifications in the temporal sense [for example LR93, KS94], and integrity enforcement through retractible literals [KS88] and abduction [e.g. T95, F96, DD92]. We present a concrete formalisation of events and time using the event calculus, and argue that the resulting package extends some of the functionalities of active rules and provides them with a declarative logical semantics. We then embed this package within an execution model of a kind developed for reactive agents by Kowalski [K95] and Shoham [S93], to capture the external reactivity feature of active rules.

In section 2 we provide a brief introduction to deductive databases and integrity constraints. In section 3 we introduce the form of the event calculus we employ in this paper and briefly discuss transition constraints and ramifications. In section 4 we compare two different ways of representing active rules in deductive databases. In section 5 we deal with external reactivity. In section 6 we compare our proposal with related work, and in section 7 we conclude.

2. Deductive databases, Integrity Constraints and Retractible Literals

Deductive Databases

A *deductive database* is a set of sentences (called *clauses*) of the form

$$H \leftarrow L_1, \ldots, L_n \qquad \text{D2.1}$$

where H, is an atom, the L_i are literals (atoms or their negations) and $n \geq 0$. Clauses where n=0 and H contains no variables are called *facts*. Non-fact clauses are called *general rules*. Predicates occurring in a deductive database can be partitioned into two exclusive sets of *extensional* and *intensional* (or *view*) predicates. The extensional predicates are defined by facts only. The intensional predicates are defined by general rules only. Databases consisting entirely of function-free definitions of extensional predicates correspond to *relational databases*. Deductive databases without function symbols and negation correspond to *Datalog* programs.

Views in relational database terminology are catered for by intensional predicates in deductive databases. For example, the clause

works_overtime(E, december, 1, 24) ← emp(E, toy, S)

defines the view predicate *works_overtime*.

Notice that the addition of a new *emp* fact to a database containing this clause implicitly adds a corresponding new *works_overtime* fact. Thus deductive databases automatically incorporate the effect of certain active rules, such as A1.2, that are used for view maintenance.

Databases change over time as a result of *updates*, in the form of addition (add) and deletion (del) of information[2]. In deductive databases both facts and general rules can be updated. Updates may be requested on both extensional and intensional predicates. Updates to the intensional predicates would be performed implicitly by updating the extensional predicates involved in the definition of the intensional ones. A *transaction* is a set of updates, grouped by the user.

Integrity Constraints

[2] Replacement of information can be achieved by combining additions and deletions.

Integrity constraints are sentences of first order logic that are properties that the database is required to satisfy as it changes by means of transactions. We assume, without loss of generality [L87] that integrity constraints are formalised as denials of conjunction of literals, i.e. have the form

$\leftarrow L_1, ..., L_m \qquad m \geq 1$

where L_i are literals.

For example, the constraint

$\leftarrow$ emp(E, D, S), not dep(D) D2.2

expresses that no one can be employed in a non-existent department.

It can be argued that active rules that provide integrity maintenance (e.g. A1.1) do two things. They specify an integrity constraint and prescribe the modifications that have to be made to the database in case of integrity violation. This combination of tasks has been addressed in the context of deductive databases, among others, by [KS88] using the notion of retractible literals.

Retractible Literals and abduction

Retractible literals indicate how integrity should be restored when an integrity constraint is violated. At least one literal in each integrity constraint is specified as retractible. Informally, if a transaction leads to a violation of an integrity constraint, then one of the retractibles in that constraint should be "withdrawn" to restore integrity. In most cases it is natural to identify a strict subset of the literals in an integrity constraint as retractible. If this is not possible, all literals in the constraint can be declared to be retractible. In general, the atom of a retractible literal may be either intensional or extensional.

For example, in constraint D2.2 either literal can be designated as retractible. The choice of *emp(E,D,S)* as retractible imposes that any update *add(emp(E,D,S))* cannot take place if the atom *dep(D)* is not in the database, and that any update *del(dep(D))* will induce the update del(emp(E,D,S)) (compare with A1.1). The choice of *not dep(D)* as retractible would force the update *add(dep(D))* from the update *add(emp(E,D,S))*, when *dep(D)* is not in the database.

To capture the functionality of active databases, we can restrict attention to the case where there is a single retractible literal in each integrity constraint whose predicate is extensional. At the end of section 3 and in section 4 we briefly discuss how the removal of this restriction extends the functionality of active databases.

The process of inducing updates while processing transactions and enforcing integrity constraints can be given an *abductive interpretation* [KM90b, KM90a, F96, DD92, KKT93].

Abduction has been used for a variety of applications, including AI planning [e.g. E88, DMB92], explanation [e.g. S89] and view updates [e.g. KM90b]. Its semantics has been investigated in [e.g. KM90a, F96] and abductive proof procedures have been proposed in [e.g. F96, DD92].

An abductive framework consists of a deductive database D, a set of integrity constraints I, a goal G, and a set of abducible literals A whose predicates are extensional. An abductive solution to G is a set Δ of ground (i.e. variable-free) literals, each of which is an instance of a literal in A, such that $D \cup \Delta \models G$ and $D \cup \Delta$ satisfies I. The designation of retractible literals in the integrity constraints further constrains the literals that appear in Δ.

This framework can be adopted for the purposes of this paper by treating transactions T as goals to be solved abductively in a deductive database D with integrity constraints I. An abductive "solution" to T is then a set Δ that implies the updates in T and includes any additional updates induced by T because of I. D updated by Δ is guaranteed to satisfy I. If no such Δ exists the transaction is unacceptable and should abort.

Example 2.1

Consider the following relational database scheme

lecturer(Name)
teaches(Person, Course)
classroom(Course, Room)
term(Course, TermOffered)
compulsory(Course)
optional(Course)

where the last two are view predicates, and the active rules

on add(teaches(P,C)), if term(C,1) then add(compulsory(C)) A2.1
on add(teaches(P,C)), if term(C,2) then add(optional(C)) A2.2
on add(teaches(P,C)), add(lecturer(P)) A2.3
on add(compulsory(C)), add(classroom(C, 311)) A2.4

In our framework active rules A2.1 and A2.2 are expressed as deductive rules

compulsory(C) ← teaches(P,C), term(C,1) D2.3
optional(C) ← teaches(P,C), term(C,2) D2.4

The active rules A2.3 and A2.4 are expressed as integrity constraints with retractibles. (The retractible literals are underlined. We use the convention of underlining the retractible literals throughout the rest of the paper.)

← teaches(P,C), <u>not lecturer(P)</u> D2.5
← compulsory(C), <u>not classroom(C,311)</u> D2.6

Consider a transaction consisting of additions of facts *teaches(tom,databases)* and *term(databases, 1)*, i.e.

T= { teaches(tom,databases), term(databases,1)}.

Then

Δ = { teaches(tom,databases), term(databases,1),
lecturer(tom), classroom(databases,311)}

The fact *compulsory(databases)* will be added implicitly through D2.3.

Notice that in this example the constraints D2.5 and D2.6 can be rewritten as general rules

lecturer(P) ← teaches(P,C)
classroom(C,311) ← compulsory(C)

All induced updates are then simply implied and there is no need for abduction. This simplification is possible for most cases of conventional active rules, except for those that deal with external reactivity.

So far we have ignored a precise treatment of deletions as well as events, temporal considerations and transition (dynamic) constraints referring to more than one database state. We remedy these shortcomings through the use of the event calculus.

3. Deductive databases as theories in the Event Calculus

The event calculus was proposed by Kowalski and Sergot [KS86] in 1986. Since then a number of simplified forms have been proposed and used for different applications [e.g. FWP95, S89, KS94]. For our purposes we will employ the simplified form which consists of the following core axioms:

holds(P,T)← happens(E,T1), initiates(E,T1,P), T1≤ T, not broken(T1,P,T) E1
broken(T1,P,T) ← happens(E*,T*), terminates(E*,T*,P), T1≤ T*≤ T E2
initiates(add(P),T, P) E3
terminates(del(P),T, P) E4

E1 and E2 state that a property (or sentence) P holds at all times after it is initiated by an event until a subsequent terminating event, if any. Database updates can be specified by assertions of the form

happens(add(P), Time)
happens(del(P), Time).

E3 states that the addition of a sentence P initiates its holding, and E4 states that the deletion of a sentence terminates its holding.

In order to formalise deductive database updates, P would have to be any clause of the form of D2.1. *holds*, in effect, specifies what clauses are present explicitly. Using the proposal of [KS94] we use the provability predicate *demo(T,P)* to allow derivation of what is also provable implicitly (as a ramification). *demo(T,P)* specifies what sentences P are present implicitly or explicitly in the database at time T. The following definition of *demo* is adequate for our purposes:

demo(T,P) ← holds(P,T) E5
demo(T,P) ← demo(T,P←Q), demo(T,Q) E6
demo(T,P∧Q) ← demo(T,P), demo(T,Q) E7
demo(T, not P) ← not demo(T,P) E8
demo(T,P∨Q) ← demo(T,P) E9
demo(T,P∨Q) ← demo(T,Q) E10

(E9 and E10 are needed to deal with integrity constraints of the form of E13 given later.)

To simulate an active rule of the form

on E@T if E_1@T_1, ..., E_m@T_m, P_1@T_1', ..., P_n@T_n', Rest then E*@T*

where *E, E** and the E_i are events, the P_i are properties (positive or negative literals), and *Rest* is a, possibly empty, conjunction of non-temporal conditions, such as *D=toy* or *Salary1<Salary2*, and temporal ordering conditions such as *T1<T*, etc., we employ an integrity constraint of the form

←happens(E, T), happens(E_1,T_1),. .., happens(E_m,T_m),
demo(T_1',P_1), ..., demo(T_n',P_n), <u>not happens(E*,T*)</u>, Rest E11

happens(Event, Time) will be the only abducible literal.

If E* is a database event (add/del) we can rewrite integrity constraints of this form with the designated retractible as a general rule

happens(E*,T*) ←happens(E, T), happens(E_1,T_1),. .., happens(E_m,T_m),
demo(T_1',P_1), ..., demo(T_n',P_n), Rest E12

Notice that our event calculus theory consisting of E1-E10 and any set of *happens* atoms is locally stratified, and thus benefits from clear logic programming semantics. Whether or not the addition of rules such as E12 preserves this property depends on the temporal relationship between the time parameter T* in the head and those in the body.

An integrity constraint of a more general form

←happens(E, T), happens(E_1,T_1),. .., happens(E_m,T_m),
demo(T_1',P_1), ..., demo(T_n',P_n), <u>(not) demo(T*,P*)</u>, Rest E13

(with or without the *not* in the retractible literal) goes beyond conventional active rules, as we understand them, by allowing non-determinism and view updates. We will give examples of these in the next section.

Having an explicit time parameter allows us to specify transition constraints, for example

Employee's salaries do not decrease.

← demo(T1, emp(E,D,S1)), demo(T2, emp(E,D,S2)), S2< S1, T1<T2

To restore an employee's salary to its original value if an update decreases it, we have to designate the second *demo* condition as retractible.

Example 3.1

We consider a modification of example 2.1 and use the event calculus formalisation.

The modifications to example 2.1 are the inclusion of a new extensional predicate

student(Name, Department)

and the replacement of active rule A2.4 by

on add(lecturer(P)), del(student(P,D))

reinforcing a constraint that no one is both lecturer and student.

The initial database can be set up with a set of *happens* events at the initial time 0, including:

happens(add(compulsory(C) ← teaches(P,C), term(C,1)), 0) D3.1
happens(add(optional(C) ← teaches(P,C), term(C,2)), 0) D3.2

and any other *happens* events adding facts concerning predicates *teaches, student, term, lecturer*, etc., for example

happens(add(student(tom, cs)), 0) D3.3

Integrity constraint D2.5 is formalised as

← happens(add(teaches(P,C)), T), <u>not happens(add(lecturer(P)), T)</u> D3.4

and the new active rule is formalised as

← happens(add(lecturer(P)), T), <u>not happens(del(student(P,D)), T)</u> D3.5

Both constraints can be rewritten as general rules

happens(add(lecturer(P)), T) ← happens(add(teaches(P,C)), T) D3.4'

happens(del(student(P,D)), T) ← happens(add(lecturer(P)), T) D3.5'

The transaction T consists of:

happens(add(teaches(tom,databases)), 1)
happens(add(term(databases, 1)), 1)

As with example 2.1 the required addition of *compulsory(databases)* is achieved implicitly by D3.1, and the other induced updates, *happens(add(lecturer(tom)),1), happens(del(student(tom,D)), 1)*, are generated deductively or abductively, respectively, with the general rule or integrity constraint version of the active rule formulation.

Our approach to formalising active rules goes some way towards representing active rules that trigger addition or deletion of other active rules, e.g.

happens(add(higher_rate_tax_payer(X)← salary(X,S), S>25000), T) ←
happens(add(new_tax_legislation_passes), T)

This general rule has the effect of triggering a new definition for deriving *higher_rate_tax_payer* on the event of the passing of new tax legislation.

4. Integrity constraint versus general rule formulation of active rules

The general rule formulation of active rules goes a long way towards achieving active behaviour. The integrity constraint formulation, however, goes further. As well as allowing embedding in an execution model suitable for reactivity external to the database, it also allows the introduction of non-determinism and view updates induced by transactions. In the next section we address the former, and in this section the latter. Our investigation of these issues is in progress, and the discussion, therefore, is somewhat sketchy.

As an example of non-determinism consider a modification of constraint D3.4 to one that insists that anyone who teaches a course must be either a lecturer or a visiting professor (vp). This can be formalised as a constraint

← happens(add(teaches(P,C)), T), <u>not hap(add(lecturer(P)</u>∨<u>vp(P)), T)</u>

We also add two general rules to the event calculus theory

hap(add(X∨Y), T) ← happens(add(X), T)
hap(add(X∨Y), T) ← happens(add(Y), T)

A transaction requesting the addition of
happens(add(teaches(tom, databases)), 1) will produce two alternative abductive solutions

Δ1={happens(add(teaches(tom, databases)), 1), happens(add(lecturer(tom)), 1)}
Δ2={happens(add(teaches(tom, databases)), 1), happens(add(vp(tom)), 1)}

As an example of an induced view update consider the constraint

← happens(add(teaches(P, C)), T), P=tom, <u>not demo(T, compulsory(C))</u>

which states that all the courses Tom teaches are compulsory.

With this constraint, and some refinement on the choice of abducibles, an update that adds a fact about Tom teaching a course will also trigger the addition of a fact that the course is offered in term 1.

5. Active rules for reactivity external to the database

In the previous sections we have attempted to show how reactivity in deductive databases can be achieved by deductive rules, integrity constraints and abduction. There is, however, another kind of reactivity which requires performing actions in an environment external to the database, as exemplified by example A1.3 in section 1.

Our language and its semantics can be used to express such active rules. What we need, in addition, is to embed this approach in an execution model that caters for reactive behaviour. Such an execution model can be found in Shoham's [S93] and Kowalski's [K95] formulation of reactive agents.

The database management system can be viewed as an agent, interacting with the external environment. Agents have goals (for example update requests), and are governed by integrity constraints that they have to satisfy while enforcing the goals. Actions might have to be performed in the external environment to guarantee that the goals are satisfied and constraints are enforced.

Kowalski [K95] gives an informal, simplified, procedural, meta-logic programming style characterisation of the top-most level of a logic-based agent as follows:

to "cycle'" at time T,
observe one input at time T,
assimilate the input from time T to T+m,
reduce current goals to subgoals from time T+m to T+m+n,
perform any requisite atomic action from time T+m+n to T+m+n+1,
"cycle" at time T+m+n+1.

Kowalski [K95] provides logic programming formulation based on this specification that can be adapted to our purposes. The parameters m and n can be adjusted to provide the required balance between the time an agent spends processing inputs and goals and the time it spends performing actions in reaction to inputs and goals, thus obtaining varying degrees of "rationality" or "reactiveness".

Both the agent and the agent's environment are interpreted as processes which communicate with each other by means of input and output streams. Input and output are symmetric in the sense that the input stream of the agent is the output stream of the environment and vice versa.

As an example formulation of reactive rules consider active rule A1.3. This can be formulated as the following integrity constraint:

← happens(add(newDate(D)), T), demo(T, tax_form(E, nil)),
<u>not happens(print "E must fill in his/her tax form", T)</u>

It may be more clear to use a different predicate, *do*, say, instead of *happens* in the second literal of the constraint to distinguish external events. This will give

← happens(add(newDate(D)), T), demo(T, tax_form(E, nil)),
<u>not do(print "E must fill in his/her tax form", T)</u>

do will then be declared abducible, in addition to *happens*.

The execution model determines what actions to perform when by inspecting its subgoals to determine whether any subgoal is an event of the form *do(A, T)* which needs to be performed at the current time T. If so, it selects one such subgoal as an update to be added to the environment.

6. Related work

This paper has been inspired by the work by Fernandes, Williams and Paton [FWP95] and Zaniolo [Z94]. Fernandes et al also use the event calculus: events and actions are represented by atoms in the predicate *happens*[3] and conditions are checked via predicates *holdsAt* and *holdsNow*, similar to our predicate *holds*, but without catering for ramifications, dealt with by the *demo* predicate in our approach. In [FWP95] view updates are not considered as possible events and actions. Moreover, active rules are seen as new entities separate from deductive rules and integrity constraints, and their semantics is given in operational terms only, via an imperative-style procedure *evaluateECASet*, rather than declaratively, via

---

[3] [FWP95] uses the Deliyanni-Kowalski [DK79] syntax for events and actions, that allows updating attributes of relations atomically. The Deliyanni-Kowalski syntax can be adopted in our approach as well.

the semantics of integrity constraints. Finally, we have suggested an agent-based architecture for the detection of events and execution of actions, whose role is of importance in the case of reactive rules. This architecture is described declaratively and thus differs from the one suggested in [FWP95].

Zaniolo [Z94] proposes a purely declarative framework for incorporating active rules within deductive databases, based on a unified semantics for active and deductive rules. Thus our objectives are similar to his. To some extent, our approach is also similar to his in our use of a temporal framework. His is based on a formalisation somewhat similar to the situation calculus. There is a close relationship between the situation and event calculi as discussed for example in [KS94, VDD95]. However, Zaniolo does not cater for ramifications or external reactivity, nor does he consider view updates as possible events and actions.

7. Conclusions

We hope that we have contributed to the discussion of the relationship between active and deductive databases, by proposing an approach based on integrity constraints and abduction to achieve reactive behaviour in deductive databases.

There is much that is left to do. Amongst these is a detailed study of the issues raised in section 4, for example. Also, so far, we have not considered the various modes of execution, deferred, immediate and detached, and the benefits that each may contribute to the deductive database setting.

Work is currently in progress on reactive agents specification and implementation [e.g. D96]. It would be interesting to follow that work and exploit its results in the context of reactive deductive databases.

Acknowledgements

This research was supported by Fujitsu Laboratories, Japan. The authors are grateful to R.A. Kowalski and D. Montesi for helpful discussions.

References

[B93] P. Bayer,
State-of-the-art report on reactive processing in databases and artificial intelligence.
The Knowledge Engineering Review, 8:2 (1993), 145--171

[D96] J.A. Davila Quintero,
A logic-based agent.
Department of Computing, Imperial College (1996)

[DD92] M. Denecker, D. De Schreye,
SLDNFA: an abductive procedure for normal abductive programs.
Proceedings of International Conference and Symposium on Logic Programming (1992), 686--700

[DK79] A. Deliyanni, R.A. Kowalski,
Logic and semantic networks.
Communications of the ACM, 22(3) (1979), 184--192

[DMB 92] M. Denecker, L. Missiaen and M. Bruynooghe,
Temporal reasoning with abductive event calculus.
Proceedings of ECAI (1992)

[E88] K. Eshghi,
Abductive planning with event calculus.
Proceedings of ICLP, MIT Press, Kowalski R.A. and Bowen K.A. (eds.) (1988), 562--579

[FWP95] A.A.A. Fernandes, M.H. Williams, N.W. Paton,
A logic-based integration of active and deductive databases.
Department of Computing and Electrical Engineering, Herriot Watt University, Riccarton, Edinburgh (1995) /*also in RIDS 96? */

[F96] T.H. Fung,
An abductive proof procedure based on Clark completion.
PhD thesis, Imperial College (1996)

[HW93a] E.N. Hanson, J. Widom,
Rule processing in active database systems.
International Journal of Expert Systems, 6(1) (1993), 83--119

[HW93b] E.N. Hanson, J. Widom,
An overview of production rules in database systems.
The Knowledge Engineering Review, 8:2 (1993), 121--143

[KKT93] A.C. Kakas, R.A. Kowalski, F. Toni,
Abductive logic programming.
Journal of Logic and Computation 2(6) (1993), 719--770

[KM90a] A.C. Kakas, P. Mancarella,
Generalized stable models: a semantics for abduction.
Proceedings of 9th European Conference on Artificial Intelligence (1990)

[KM90b] A.C. Kakas, P. Mancarella,
Database updates through abduction.
Proceedings VLDB (1990)

[KS88] R.A. Kowalski, F. Sadri,
Knowledge representation without integrity constraints.
Imperial College Technical Report (1988)

[KS94] R. Kowalski, F. Sadri,
The situation calculus and event calculus compared.
Proceedings of ILPS, Bruynooghe M. (ed.), MIT Press, 539-553, 1994.

[KS86] R. A. Kowalski, M. Sergot,
A logic-based calculus of events.
New Generation Computing, 4 (1986), 67--95

[K95] R.A. Kowalski,
Using meta-logic to reconcile reactive with rational agents.
In Meta-logic and Logic Programmimg, K.Apt and F. Turini (eds.), MIT Press (1995), 227--242

[LR93] F. Lin, R. Reiter,
State constraints revisited.
Proceedings of the Second Symposium of Logical Formalization of Commonsense Reasoning (1993), 114--121

[L87] J. W. Lloyd,
Foundations of logic programming.
Springer Verlag Symbolic Computation Series (1987)

[MT95] D. Montesi, R. Torlone,
A rewriting technique for the analysis and the optimization of active databases.
Proceedings of International Conference on Database Theory (1995)

[S89] M.P. Shanahan,
Prediction is deduction but explanation is abduction.
Proceedings of IJCAI (1989), 1055--1060

[S93] Y. Shoham,
Agent-oriented programming.
Artificial Intelligence 60(1) (1993), 51-- 92

[T95] F. Toni
Abductive logic programming.
PhD thesis, Imperial College (1995)

[VDD95] Van Belleghem K., Denecker M., De Schreye D.,
Combining situation calculus and event calculus.
Proceedings of ICLP, MIT Press, Stirling L. (ed.) (1995), 83--97

[W94] J. Widom,
Deductive and active databases: two paradigms or ends of a spectrum.
In Rules in Database Systems: Proceedings of the 1st International Workshop, N.W. Paton and H. Williams (eds.), Springer-Verlag (1994), 306--315

[Z94] C. Zaniolo,
A unified semantics for active and deductive databases.
In Rules in Database Systems: Proceedings of the 1st International Workshop, N.W. Paton and H. Williams (eds.), Springer-Verlag (1994), 271--287