Comparing Deductive and Active Databases

Alvaro A.A. Fernandes

ABSTRACT This chapter compares the use of *rules* in deductive and in active databases. Syntactic similarities notwithstanding, deductive rules and active rules express quite different knowledge models and induce quite distinct execution models. In the last few years, there has been a strong research interest in unifying the two uses of rules by means of a language that either reconciles or, at least, encompasses the most distinctive aspects of their expected combined functionality. After a discussion of the most salient differences between deductive and active rules, an overview is given of different approaches to characterize a rule language providing some level of support for the knowledge and execution models that underlie deductive and active rules. The chapter includes a brief survey, of rule languages that attempt to bridge the divide.

1 Introduction

1.1 Main Classes of Rule-Based Database Systems

The use of rule sets to express application semantics has spawned two classes of database systems: deductive databases and active databases. This chapter has the main objective of discussing the differences between these two classes and surveying recent research efforts applied to the problem of unifying them.

Rules in database systems have been deployed as specifications of different kinds of functionality. On the one hand, rules have been used to specify schemas, derived data, views and integrity constraints, and on the other, rules have been used to specify events of interest along with the automatic, but conditional, response of the database system to the occurrence of such events. The first deployment of rules as a specification language gave rise to deductive databases, the second to active databases.

The starting point for this comparison of deductive and active databases is the observation that, syntactic similarities notwithstanding, the different knowledge and execution models of deductive and active rules are difficult to reconcile, let alone unify. Many attempts find themselves forced to adopt a controversial, because reductionist, view of one or the other kind of rule.

1.2 Some Pragmatic Issues to Confront

The facts that any unification attempt must confront include:

- *Deductive and active database technologies target different aspects of application functionality.*

  While deductive databases major at providing database designers with a unified language to define schemas, integrity constraints, views and (certain classes of) derived data, active databases major at allowing application programmers to delegate to the database management system parts of the application's functionality that would otherwise be scattered across many modules and many programs.

  Although this generalization is no doubt too broad, it is supported by the fact that deductive database research has suffered from a dearth of clearly defined applications which could be argued to benefit from its results. In contrast, much active database research is targeted at, and sometimes driven by, specific application areas, and would, in any case, find it less difficult to list beneficiaries of its results.

- *Deductive and active databases have different, formally distinct, theoretical foundations.*

  Although both deductive and active databases could be said to have origins in artificial intelligence research, while the former have emerged from research into automated theorem proving via logic programming, the latter are closer to research into expert systems based on production rules.

  Thus, deductive databases spring from a tradition in which there is little concern with modeling state transitions because only the static aspects of applications are of concern, whereas active databases spring from a tradition in which one of the main concerns of the system is to model state transitions, insofar as the focus is markedly on the dynamic aspects of applications

- *Research efforts into deductive and active databases have long evolved independently and with different overall objectives.*

  Deductive database research has been mainly driven by metalogical development, i.e., by the exploitation of the fact that deductive databases form a class of very constrained formal systems based on classical first-order predicate logic. Concentration has been on studying useful techniques for query optimization and evaluation, integrity-constraint checking and view maintenance, while at the same time exploring issues of expressiveness and computational complexity of particular classes of logic-based query languages. Deductive database research has, therefore, followed the pattern of relational database re-

> search in first tackling the development of formal foundations before engaging in the implementation of prototype systems.
>
> Active database research has followed the different pattern of experimenting with prototype systems before pondering the formal properties that such systems may be formally shown to have. Thus, an implementation-driven approach to issues such as the design of very expressive event specification languages, subtle and refined coupling modes, and very powerful execution models, has dominated research activity with much less attention being given to formal theories with which the results of this activity could be studied in abstraction from their concrete realizations.
>
> Thus, much of the attractiveness of deductive database systems lies in their formal underpinnings, which rely crucially on languages with carefully controlled expressiveness and on an assumption of a static model of reality, whereas active databases are often attractive mainly because of their largely unconstrained expressiveness and of their potential effectiveness as a tool for modeling an important class of phenomena implicit in a dynamic model of reality.

In the context of this chapter, the main questions posed by these observations include whether, to what extent and, broadly, how can a class of database systems be characterized that reconciles, or more ambitiously, unifies deductive and active databases.

1.3 Possible Directions for Unification Efforts

Two possible approaches suggest themselves: either one starts from the framework underlying deductive rules and works towards encompassing active database technology under that framework or else one starts from the framework underlying active rules and works towards encompassing deductive database technology. It comes as no surprise that the literature contains records of attempts following each of these approaches.

There has been a great deal more research activity in the deductive-to-active direction than in the opposite one. This seems to stem from the following implicit assumptions. Firstly, that there is functionality that active databases deliver which deductive databases cannot deliver unless they are extended, and that such extensions require reconsidering the very logical basis which underlies current deductive databases. Secondly, that active databases could be made to support any functionality delivered by deductive databases if suitably constrained. These assumptions lead to the view that there are more, and more challenging, research issues in the deductive-to-active direction than in the opposite one.

Whereas the first assumption is trivially true, insofar as classical deductive databases cannot effect state transitions as can active ones, it is more difficult to agree without qualifications on the truth of the second. Even if one assumes an active database whose knowledge model is constrained to effect no state transition and to not monitor events (other than user requests reflecting the activation of deductive database services, e.g., evaluate a query, check integrity, etc.), it remains the case that the execution model of (pure) active databases would have to be extended with a theorem-proving procedure if they were to deliver the functionality of deductive databases as currently construed.

1.4 Main Requirements for Unification Efforts

Adopting a somewhat simplifying stance to what is in fact a complex web of issues, the following observations (to which labels are attached for further reference) seem to lie at the start of any enterprise aimed at integrating or unifying deductive and active databases:

(R1) a logic-based view of a dynamic reality: Supporting active functionality in deductive databases involves supporting, at least, a logic-based view of a dynamic reality, e.g., one that is modeled as a sequence of states. A logic-based account of event specification and detection, of the interleaving of event detection, condition verification and action execution, and of action specification and execution must emerge.

(R2) a declarative semantics for rule sets: Supporting deductive functionality in active databases requires, at least, the execution model to support automated theorem-proving in the adopted system of logic. The knowledge model must allow behavior invocations that request the initiation of the tasks typically supported by a deductive database and must allow action execution to have its ability to effect state transitions curtailed.

The remainder of this chapter is structured as follows. Section 2 discusses the syntactic similarities and the semantic differences between the deductive- and the active-rule paradigms. Section 3 is aimed at delineating the specific issues that an attempt to unify deductive and active rules must aim at. Section 4 is a brief survey of proposals for a unifying rule language that provides some level of support for the knowledge and execution models that underlie deductive and active rules. Finally, Section 5 draws some conclusions.

2 Deductive Rules Compared with Active Rules

A rule in a deductive database system can be viewed as a closed formula of first-order predicate logic or as a state-preserving production rule. The former is a logic-based view and the latter a production view of what a deductive rule specifies.

2.1 Semantic Dissimilarities

In the simpler cases, the two views are equivalent, in the sense that the information that can be extracted through rule processing from a database state in either view is the same and, since no state transitions are allowed, so is the unique database state resulting from rule processing. This equivalence means that the semantics of a deductive rule can be formally described from three different, but equivalent, viewpoints [vEK76]: the first two stem from the logic-based view and are referred to as the model-theoretic and the proof-theoretic semantics, respectively, while the third stems from the production view and is typically referred to as the fixpoint semantics.

The model-theoretic approach is purely declarative. It does not rely on any algorithm to define formally the meaning of a deductive rule set nor does it suggest how such an algorithm might be constructed, e.g., to be used in extracting information from the database using the deductive rule set. The proof-theoretic approach is also declarative insofar as it does not rely on an algorithm to define the meaning of a rule set either, however it does suggest one or more proof procedures, i.e., proof-constructing algorithms, one of which would stand at the core of an automated theorem-prover by means of which information can be extracted from the database using the deductive rule set. Crucially, what is common to these two approaches to the assignment of meaning to deductive rule sets in a formal way is the fact that both view the union of a deductive rule set with the database state underlying it as the set of axioms of a logical theory [Men87, Rei84].

The production view is not at all dependent on viewing rule sets and database states as axioms of a logical theory. The semantic account that underlies the production view of deductive rule sets relies on, and precisely defines, an algorithm by means of which information can be extracted from the database. This account is referred to as a fixpoint semantics because the algorithm computes the fixpoint of a pattern-matching operator, sometimes referred to as *elementary production* [CGT90].

The production view faces comparatively simple problems to extend the rule language towards allowing rules to effect state transitions, i.e., to insert and delete facts as a result of computations. In contrast, the logic-based view faces, from the outset, a fundamental dilemma: either to provide an account of state transitions within the framework of first-order logic, or else to reconstruct somewhat from scratch the foundations of deductive rule processing. This dilemma stems from the fact that first-order logic views the reality from which semantic models are drawn as essentially static, and hence the logic-based view may need to switch to using as root a logical system which intrinsically models a dynamic reality (e.g., an intensional, perhaps modal, logic of time or of state).

The crucial difference between deductive and active rules is that the latter only supports a production view of deductive rule processing if no state transitions are allowed, i.e., if actions are constrained not to transform, but merely report on, the state of the database.[1] This leads to the conclusion that, in their pure form, deductive rules and active rules have irreconcilable purposes, insofar as, by definition, the very purpose of active rules makes it impossible for deductive rules to retain their semantic foundations.

2.2 Syntactic Similarities

There is a deceptive syntactic similarity between deductive and active rules, further fostered by the fact the logic-based and the production views agree if rules are prevented from expressing state transitions. If one only considers active rules with no event specification part, both deductive and active rules can be read as '**if ... then ...**' expressions. However, for deductive rules the intended reading interprets the rule in a declarative, truth-denoting way, whereas for active rules the intended reading is altogether different, insofar as it interprets the rule in an imperative, transition-effecting way.

While the whole of a deductive rule expresses a condition, only the condition part of an active rule does so, insofar as the whole of an active rule, in general, denotes instead a state transition. Furthermore, the antecedent of an active rule is indeed truth-valued but it is only interpreted as a guard on the execution of the consequent, which may cause a state transition (e.g., via a simple destructive assignment, or via the invocation of a procedure capable, in principle, of expressing any computation whatsoever). In contrast, the entire deductive rule is truth-valued, being equivalent to asserting that either the antecedent is false or the consequent is true or both. The bridge that is typically built in extensions of deductive databases towards encompassing active functionality is one that changes this purely declarative, truth-functional interpretation and views the antecedent as a guard. The problem, of course, is that this change of interpretation may require a far-reaching reconstruction of the semantic foundations of deductive rule sets.

2.3 A Dimensional Analysis of Deductive Rule Sets

Clearly, the knowledge models that can be expressed by an active rule set and by a deductive rule set have, in principle, only a small intersection. Only the deceptive syntactic similarity between deductive and active rules somewhat obscures this realization. The application to Datalog of the

[1]In terms of the dimensions proposed in the Introduction to this book this is equivalent to constraining the Options dimension of the Action component of the Knowledge model to be the singleton {Inform}.

framework adopted in this book to describe active rule systems highlights how an attempt to view a pure deductive rule system as an active one reveals a severely limited functionality and throws light, therefore, on the areas where extensions should be sought in any unification effort.

For the purpose of illustrating some of the observations above, Table 1.1 describes deductive rules along the dimensions proposed in the Introduction to this book.

Knowledge Model:

Event	Source	[not applicable]
	Granularity	[not applicable]
	Type	[not applicable]
	Role	None
Condition	Role	Mandatory
	Context	$DB_T = DB_C$
Action	Options	[not applicable]
	Context	$DB_T = DB_C = DB_A$

Execution Model:

Condition-Mode	Immediate
Action-Mode	Immediate
Transition granularity	[not applicable]
Net-effect policy	[not applicable]
Cycle policy	[not applicable]
Priorities	None
Scheduling	Saturation
Error handling	[not applicable]

Management Model:

Description	Query Language
Operations	$\emptyset$
Adaptability	Compile Time
Data Model	Deductive

TABLE 1.1. Dimensions for (Pure) Deductive Rules

Notice that deductive rules do not monitor events. Conditions are mandatory, insofar as a fact, i.e., a consequent-only rule is understood to have a vacuously true antecedent. The contexts for processing both conditions and actions are, and must remain, the same as they were at the start of the transaction.

Notice also that actions are not specifiable, in particular notice that in pure deductive rule systems the consequent of a rule is *not* the specification of an action. Rule processing culminates in an answer set to the query that initiated that processing cycle, although one might choose to view the processing of each individual rule as yielding a set of bindings, and the processing of the entire set as yielding the final set of bindings as an answer to a query which is then returned to the caller. The condition and action modes are said to be immediate in the sense that deductive rule processing does not interact with the caller transaction in any more sophisticated manner. This also explains why the error handling dimension is typically not applicable to deductive rule processing.

Deductive rule processing can be carried out without any conflict resolution in a purely declarative language such as Datalog. However, if stratification is used to allow negation in bodies [CGT90], then the strata defined by the occurrence of negative literals impose an ordering on rules that the evaluation algorithms take into account. Scheduling is typically by saturation, but there are evaluation schemes that process rules with an all-parallel scheduling.

Notice, finally, that no operations on the rule set are allowed, and the mandatory creation and deletion operations are only allowed at compile time. This means that the requirement that no state transition occurs is extended to the rule set itself and is consistent with the view that a database state is the union of the rule set and the set of extensionally asserted facts, and not just the latter.

3 Bringing Deductive and Active Rules Together

3.1 Issues and Targets

As Section 2 suggested, the differences between deductive rule systems and active rule systems give rise to, at least, the following broad questions (to which labels are attached for further reference):

(Q1) ***How to support events in deductive rule processing?*** Deductive rule systems neither detect nor react to events. From this it follows that they are, in the terminology introduced in the Introduction to this book, *call-driven systems.*

(Q2) ***How to support state transition in deductive rule processing?***

(Q3) ***How to support interaction with the transaction manager in deductive rule processing?*** Deductive rule systems neither effect state transitions nor interact with the transaction management system in any significant way.

(Q4) ***How to support a logic-based view of the database in active rule processing?*** Active rule systems do not support a logic-based view (i.e., model- and proof-theoretic) of rule processing.

Questions **Q1**, **Q2**, and **Q3** stem from requirement **R1** from Section 1.4, viz., that a logic-based view of a dynamic reality must emerge from a deductive database that is extended to support active database functionality. Question **Q4** stems from requirement **R2** from Section 1.4, viz., that a declarative semantics for rule sets must be provided by an active database that is extended to support deductive database functionality.

These questions, therefore, implicitly set targets for any endeavor to extend support for active behavior in deductive databases and for deductive behavior in active databases. For the former, more complex case, Table 1.2 shows, in terms of the dimensions proposed in the Introduction to this book, what have been typical targets in attempts to support active behavior in deductive rule processing.

Knowledge Model:

Event	Source	Structure Operation
	Granularity	Set
	Type	Primitive
	Role	Optional
Condition	Role	Mandatory
	Context	$\mathrm{DB}_T = \mathrm{DB}_C$
Action	Options	Structure Operation
	Context	$= \{\mathrm{Bind}_C, \mathrm{DB}_C = \mathrm{DB}_A\}$

Execution Model:

Condition-Mode	$\subset$ { Immediate, Deferred }
Action-Mode	$\subset$ { Immediate, Deferred }
Transition granularity	Set
Net-effect policy	No
Cycle policy	$\subset$ { Iterative, Recursive }
Priorities	Relative
Scheduling	Saturation
Error handling	[not applicable]

Management Model:

Description	Query Language
Operations	$\emptyset$
Adaptability	Compile Time
Data Model	Deductive

TABLE 1.2. Typical Targets for an Active Extension of Deductive Databases

Thus, attempts so far have, in general, confined themselves to supporting primitive, structure-operation events without an explicit specification and a limited ability to support actions denoting structure operations only, with bindings from the condition evaluation stage being propagated to the action execution stage.

3.2 Basic Research Stances

Apart from specific proposals, of which a small subset is described later in this chapter, there has been some interest from both the deductive database and the active database research areas in investigating whether a broad, general framework that reconciles the two kinds of rules in a single database system can be devised in an intuitive and clean way. Apart from their intrinsic interest, attempts to formulate such broad views are useful in characterizing basic research stances.

Two somewhat symmetrical positions can be discerned. The first position, reported in [Wid94], stems from the active-rule research area and proposes that active rules subsume (the important aspects of) deductive rules. It presents as evidence a framework from which a characterization of both deductive and active rule systems can be derived in a straightforward way. Under this approach, active rules and deductive rules are construed as lying at opposite ends of a spectrum in which one can move from purely declarative to totally imperative rule-based expression of behavior in discrete incremental steps, each of which relinquishes some declarativeness and encompasses more imperativeness.

The second position [Zan94] stems from the deductive-rule research area and proposes that deductive rules subsume (important aspects of) active rules. It argues that the fixpoint interpretation of deductive rule sets can be extended to compute strata which emulate the notion of database states. In addition to this crucial first step, [Zan94] defines four predicates (to be understood as built-ins): these are referred to as `quevt`, `add`, `del`, and `evt`. Intuitively, `quevt` characterizes an intensional relation which models an event queue; `add` and `del` characterize intensional relations which model the set of inserted and deleted facts, respectively, and, finally, `evt` characterizes an intensional relation that models the log of raised events. These extensions open the way for the development of an account of active rules in which the semantics of event detection and state transition is declaratively specified, which allows [Zan94] to remain firmly within the semantic framework of deductive rules. Both [Wid94] and [Zan94] are further discussed later in this chapter.

It seems that the respective stances taken by [Wid94] and [Zan94] can only be sustained if one judiciously arbitrates which features of deductive rules (in the case of [Wid94]) or of active rules (in the case of [Zan94]) are important and need supporting in the extended framework. Thus, it is not clear that certain of the implementation technologies associated with deductive-rule processing (especially optimization) can be so easily assimilated in the active-rule setting as implicitly assumed in [Wid94]. Similarly, the framework proposed in [Zan94] falls short of achieving the expressiveness of active rules, in particular with respect to interactions with the transaction manager. In terms of the questions put forward in Section 3.1, it is arguable that under the stance taken in [Wid94] certain implications of **(Q4)** would be difficult to support, and the stance taken in [Zan94] does not address **(Q3)** and takes a limiting view of both **(Q1)** and **(Q2)**.

The stances taken by [Wid94] and [Zan94] can be characterized as favoring a tight coupling between the deductive and the active rule processing components. A third stance emerges from considering a looser coupling. This is the view taken by [FWP97]. It is founded on the perception that paradigm-merging enterprises are only justified if they deliver more than is provided by each paradigm in isolation and if the original paradigms retain their distinctive strengths in the resulting merger.

The basic idea is to use a history of events as the basis for the characterization of all deductive and active database functionality. Firstly, over the event history, occurrences of primary and composite events are declaratively characterized by deductive rules, thereby allowing event detection to be given a logical semantics. Secondly, over the same event history, a Kowalski-Sergot event calculus [Kow92] models the successive states of the database as time-stamped intensional relations, and over these, in turn, application-specific deductive rules can be written. This means that condition verification retains its classical logical semantics. Finally, the events caused by the execution of the action component in an active rule need only be appended to the event history for the logical framework to reflect it both in event detection and condition verification. This makes the semantics of action execution very simple in comparison to other approaches.

In contrast with [Wid94], in [FWP97] deductive functionality of the database is fully supported. In contrast with [Zan94], in [FWP97] a logical framework is provided from which a natural formalization of those aspects of active behavior which are necessary to address **(Q1)** and **(Q2)** more directly emerge in the form of more expressive event and action specification languages than are possible in the approach taken by [Zan94]. Although **(Q3)** is also not addressed by [FWP97], there is no strong impediment for extending the approach so that it does so. ROCK & ROLL, an implemented system whose ADB functionality is based upon the foundations laid in [FWP97] is further discussed later in this chapter.

4 A Brief Survey of Proposals for Languages with Deductive and Active Capabilities

4.1 The Widom Spectrum

The proposal in [Wid94] is not of a language, but rather of a spectrum within which the execution models of existing rule languages exist as points. The central argument is that there is a basic computational engine for rule processing that can be gradually extended, thereby giving rise to a discrete number of points in a spectrum, so as to characterize the execution model of a number of rule languages from purely-deductive Datalog to very-active Postgres. Only the production view of deductive rules is considered and there is no attempt to view deductive rules as axioms of a logical theory. Hence, **(Q4)** is not addressed.

As a result, unless one is prepared to accept a more restricted perspective of deductive database technology than currently embodied in existing prototypes, [Wid94] falls short of the unification ideal. Also, it is arguable that some of the extension steps from one point to another are not as small, intuitive and well-understood as one would wish, i.e., the distance between any two adjacent pair of languages in the spectrum is far from constant and can indeed reflect quite complex extension steps.

The important contribution of [Wid94] is to demonstrate that there exists a level of abstraction which allows the description of a subsumption relationship in which deductive rules are active rules. It seems, however, that the level of abstraction in which the description is viable is far too high to explain the actual practice of deductive databases (DDBs) and of active databases (ADBs) as embodied in concrete prototypes and systems.

4.2 DDB-Oriented Proposals

The proposals discussed in this subsection are grouped together by their origins and allegiance to the deductive approach to database programming. The proposals discussed below are DLP [MW88], $\mathcal{LDL}$ [NT89], U-Datalog [BCG^{+}94, BGM95], $\mathcal{T}_{\mathcal{R}}$ [BK93], and [Zan94].[2] Since this survey does not aim at completeness it should not be assumed that the above are the only proposals for active extensions to the deductive approach.

DLP follows a top-down, left-to-right approach to evaluation that leads naturally to the specification of updates in the antecedents of rules. DLP (for Dynamic Logic Programming) casts a Harel-style modal logic of programs into the logic programming framework. Thus, it provides a logic-based view of a dynamic reality via the classical semantics for modal logics based on Kripke structures. As a consequence, update operators ('+' for insertion and '−' for deletion) are interpreted as modal operators. Updates are performed immediately and update atoms can be preceded or followed in the evaluation order by a query-only atom. If a query atom following an update atom evaluates to false in the updated state then changes are rolled back. Updates can only be specified on the extensional part of the database. There are other restrictions on the expressiveness of DLP programs (e.g., negation is not handled, recursive rules with update operators are disallowed).

$\mathcal{LDL}$ is in many ways similar to DLP but rule antecedents are more

[2] Unfortunately, space restrictions prevent a wider coverage. In particular, the reader might wish to consider the interesting work reported in [BJ94] and in [HD94].

clearly structured into a query part and a procedure part. The former is executed first and must ground all atoms in the procedure part. $\mathcal{LDL}$ can detect that the procedure part yields the same final state irrespective of the order of evaluation and hence is capable of optimizing its execution plan. Another important distinction between DLP and $\mathcal{LDL}$ is characterized by the tuple-at-a-time approach of the former and the set-at-a-time approach of the latter. The DLP approach can lead to non-determinism, while the $\mathcal{LDL}$ approach allows an all-parallel scheduling policy.

Both DLP and $\mathcal{LDL}$ are representative of the first wave of deductive languages extended with active functionality. They remain close to to the top-down evaluation approach of traditional logic programming proof-procedures. They face up to the ideal of having a model- and a proof-theoretic semantics for transition-effecting languages too. However, while they address **(Q2)**, the effecting of state transitions, they are also totally oblivious to **(Q1)**, the support for events, and fall short of addressing **(Q3)**, the integration with transaction management. This last question is picked up by the second wave of logic-programming based deductive rule languages extended with active functionality.

U-Datalog relinquishes the immediate-update approach first proposed by DLP (which implies the loss of declarativeness under most circumstances) and incorporates into the evaluation mechanisms an awareness of transaction management, going as far as allowing constructors into the rule language to form composite transactions. U-Datalog is an extension of Datalog supporting the declarative specification of updates. The execution model of U-Datalog consists of two phases: the marking phase and the update phase. In the marking phase, the updates specified as a result of the evaluation process are collected but not carried out; the update phase carries out the updates provided there are no conflicting pairs and all update specifications are ground. This is in contrast with DLP and $\mathcal{LDL}$ in which updates are also specified in the antecedent of rules but are carried out as an immediate side effect of the evaluation process. The semantic framework underlying U-Datalog is that of constraint logic programming, which is a generalization of the classical logic programming framework based on first-order term unification. In U-Datalog, updates are specified as constraints. This gives U-Datalog a more general semantic setting than DLP or $\mathcal{LDL}$. Also, a bottom-up evaluation strategy exists for U-Datalog, where none has been provided for DLP.

$\mathcal{T_R}$ (for Transaction Logic) is, in some sense, the confluence of DLP and U-Datalog, insofar as its logical foundations are, like those of DLP but not U-Datalog, in a Harel-style modal logic of programs, and it formalizes, like U-Datalog but not DLP, the relationship between a transaction model and rule evaluation.

Thus, U-Datalog and $\mathcal{T_R}$ address **(Q2)**, the effecting of state transitions, as do DLP and $\mathcal{LDL}$, and go beyond the latter by also addressing **(Q3)**, the integration with transaction management. However, none of these languages tackles **(Q1)**, the support for events. This question is picked up by the most recent proposal for a logic-programming based deductive rule language extended with active functionality.

The main contribution of [Zan94] is an approach to unifying aspects of deductive and active databases that encompasses event specification, thereby addressing **(Q1)**. The approach is founded on a model of state transition that identifies distinct states with different strata of a fixpoint computation. Not only does this enable [Zan94] to model state transitions non-destructively, it also yields a model of an event queue, thereby opening the way for the explicit specification of events in a deductive rule context. Unfortunately, [Zan94] does not include an account of how deductive rule evaluation integrates with a transaction manager as do U-Datalog and $\mathcal{T}_{\mathcal{R}}$.

It is interesting to note that by explicitly naming states, [Zan94] seems to stand in the same relation with the modal approaches of DLP and $\mathcal{T}_{\mathcal{R}}$ as a tensed logic (i.e., a logic which can quantify over time-denoting terms) to a temporal logic (in which, rather then explicit quantification, modal operators are used to talk about formulas that hold, or not, over more than one point in time).

4.3 ADB-Oriented Proposals

The proposals discussed in this subsection are grouped together by the fact that their users primarily see them as instances of the active approach to database programming. The proposals discussed are A-RDL [SK96], Chimera [CM93, CFPT96] and ROCK & ROLL [DPWF96]. Since this survey does not aim at completeness it should not be assumed that the above are the only proposals for the integration of deductive features with the active approach.

A-RDL is an extension of RDL1 [KdMS90]. RDL1 only supported deductive rules but it already had the ability to express state transitions (via update operators on rule consequents). A-RDL supports active rules that react to events modifying the state of the database. A-RDL active rules are production rules, i.e., they do not allow event specification. They allow the specification of coupling modes and can command the rule not to fire again. The condition and action specification languages are richer than any of the deductive-oriented proposals discussed above. However, A-RDL rigidly adheres to a production view of rule processing. Thus, it does not answer **(Q4)**, the support of a logic-based view of the database, insofar as a model- and proof-theoretic semantics for A-RDL rule sets has not yet been provided.

Chimera, like ROCK & ROLL, integrates active and deductive databases in an object-oriented context. Chimera supports a comparatively sophisticated transaction model. Deductive rules (called 'passive' in Chimera) must be, as in practical implementations of Datalog, both range restricted and stratifiable. Active rules are event-condition-action rules. Conditions are queries, equivalent to passive-rule bodies augmented with special facilities to query the event history. While there are no restrictions on specifiable actions (within the manipulation capabilities of Chimera and of the externally-defined procedures into which the system has an interface), only query and manipulation events are specifiable. Moreover, there is no possibility of specifying complex events although Chimera does support alternative consumption modes. The formal semantics of the rule-language components of Chimera has yet to be made publicly available. Gauging from publicly available documents, there is reason to suppose that Chimera does support all of **(Q1)** to **(Q4)** subject to reasonable restrictions.

ROCK & ROLL shares many similarities with Chimera. It extends with active rules the deductive object-oriented database system described in [BFP+95]. The design of the extension is such that deductive rule processing is not disturbed syntactically or semantically. Since ROLL (the deductive language of ROCK & ROLL) is reducible to Datalog with range restriction and stratification, this implies that the active version of ROCK & ROLL supports **(Q4)**, i.e., a logic-based view of the database. With regard to **(Q1)**, i.e., support for events, ROCK & ROLL is comparable to Chimera, allowing the specification of a rich set of primitive events and providing a rich set of operators to compose events. With respect to **(Q2)**, support for state transitions, ROCK & ROLL allows unrestricted use of ROCK (the imperative language of ROCK & ROLL) in the action specification part of an active rule. Finally, ROCK & ROLL supports immediate and deferred coupling modes but is mostly dependent for its transaction management facilities on those provided by the implementation platform, viz., EXODUS [CDG+90]. The formal semantics of the active extensions of ROCK & ROLL builds upon the foundations laid in [FWP97].

5 Conclusion

The first conclusion to be drawn is that the Widom spectrum mentioned in Section 4.1 and discussed in more detail in [Wid94], seems to deserve further development. Independently, this is already happening, to a certain degree and under different motivations, in the explorations of the foundations of rule-based languages reported in [PV95] and [Via96] and in [FT95]. These explorations could still yield the formal underpinnings for the Widom spectrum which a position paper such as [Wid94] is not expected to address. Notice, however, that the impressive depth of the research just cited suggests that the formal semantics of unconstrained ADBs tends to diverge from, rather than approximate, the logical semantics which underlies the very idea of a DDB. This makes it less likely that the Widom spectrum can be justified by formal arguments. Even in the most optimistic view, for a convergence to occur, the depth and focus of research into the purely logical aspects involved will have to match that of [PV95, Via96, FT95].

The second conclusion, for which the survey in Section 4 provides the evidence, is that following a DDB-oriented approach has not yet led to as comprehensive and satisfying results as those yielded by an ADB-oriented approach.

Seen as a group the DDB-oriented proposals mentioned in Section 4.2 address **(Q1)**, the support for events,**(Q2)**, the effecting of state transitions, and **(Q3)**, the integration with transaction management. However, no single proposal has answers for all three questions. Even if the expressiveness of the event and action specification languages in this group is limited in comparison with state-of-the-art active database systems, as are the possibilities of integrating rule processing and transaction management, the time seems ripe to attempt a convergence of proposals, such as those surveyed above, into a clean, powerful unifying rule-based database language following a DDB-oriented approach.

In contrast, two of the three ADB-oriented proposals mentioned in Section 4.3, viz., Chimera and ROCK & ROLL, yield a fully-fledged active database with deductive capabilities. Both these proposals provide answers for all of **(Q1)** to **(Q4)**, the differences between them being few and not substantial in what concerns the topic of this chapter.

A final conclusion is that while Chimera and ROCK & ROLL suggest that adopting an ADB-oriented approach to the unification of the two classes of rule processing tends naturally to converge, this convergence seems possible but must be actively sought in the case of a DDB-oriented approach. There seems to be enough evidence to suggest that the body of research dispersed throughout the many proposals in Section 4.2 is ripe for consolidation and unification at a foundational level.

Acknowledgement The author's understanding of the subjects discussed in this chapter is due in great part to his joint work with the members, past and present, of the Database Research Group at Heriot-Watt University, and in particular, Dr. Norman Paton, Andrew Dinn, and Prof. M. Howard Williams. Any error, however, is unquestionably the author's own.

6 References

[BCG+94] E. Bertino, B. Catania, G. Guerrini, M. Martelli, and D. Montesi. A Bottom-Up Interpreter for Database Languages with Updates and Transactions. In *Proceedings Joint Conference on Declarative Programming Gulp-Prode*, pages 207–220. 1994.

[BFP+95] M.L. Barja, A.A.A. Fernandes, N.W. Paton, M.H. Williams, A. Dinn, and A.I. Abdelmoty. Design and Implementation of ROCK & ROLL: A Deductive Object-Oriented Database System. *Information Systems*, 20(3):185–211, 1995.

[BGM95] E. Bertino, G. Guerrini, and D. Montesi. Towards Deductive Object Databases. *Theory and Practice of Object Systems*, 1(1):19–39, 1995.

[BJ94] P. Bayer and W. Jonker. A framework for supporting triggers in deductive databases. In N.W. Paton and M.H. Williams, editors, *Proc. 1st Int. Workshop on Rules In Database Systems*, pages 316–330. Springer-Verlag, 1994.

[BK93] A.J. Bonner and M. Kifer. Transaction Logic Programming. In *Proceedings of 10th ICLP*, pages 257–279. MIT Press, 1993.

[CDG+90] M.J. Carey, D.J. DeWitt, G. Graefe, D.M. Haight, J.E. Richardson, D.T. Schuh, E.J. Shekita, and S. L. Vanderberg. The EXODUS Extensible DBMS Project: An Overview. In S. Zdonik and D. Maier, editors, *Readings in Object-Oriented Database Systems*, pages 474–499. Morgan Kaufmann, 1990.

[CFPT96] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Active Rule Management in Chimera. In J. Widom and S. Ceri, editors, *Active Database Systems*, pages 151–176. Morgan Kaufmann, 1996.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer-Verlag, 1990.

[CM93] S. Ceri and R. Manthey. Chimera: A Model and Language for Active DOOD Systems. In *Proceedings of the 2nd East-West Database Workshop*, pages 3–16. Springer-Verlag, 1993.

[DPWF96] A. Dinn, N.W. Paton, M.H. Williams, and A.A.A. Fernandes. An Active Rule Language for ROCK & ROLL. In *Proc. 14th British National Conference on Databases.* Springer-Verlag, 1996.

[FT95] P. Fraternali and L. Tanca. A Structured Approach for the Definition of the Semantics of Active Databases. *ACM Transactions on Database Systems*, 20(4):414–471, 1995.

[FWP97] A.A.A. Fernandes, M.H. Williams, and N.W. Paton. A Logic-Based Integration of Active and Deductive Databases. *New Generation Computing*, 15(2):205–244, 1997.

[HD94] J.V. Harrison and S.W. Dietrich. Integrating active and deductive rules. In N.W. Paton and M.H. Williams, editors, *Proc. 1st Int. Workshop on Rules In Database Systems*, pages 288–305. Springer-Verlag, 1994.

[KdMS90] G. Kiernan, C. de Maindreville, and E. Simon. Making Deductive Databases a Practical Technology: A Step Forward. In *Proceedings 1990 ACM SIGMOD*, pages 237–246. ACM Press, 1990.

[Kow92] R. Kowalski. Database updates in the event calculus. *Journal of Logic Programming*, 12:121–146, 1992.

[Men87] E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth & Brooks/Cole, 3rd. edition, 1987.

[MW88] 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.

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

[PV95] P. Picouet and V. Vianu. Semantics and Expressiveness Issues in Active Databases, 1995. To appear in *Journal of Computer and System Sciences*. Shorter version in *Proceedings PODS 95*, pp. 126-138.

[Rei84] R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, pages 191–233. Springer-Verlag, 1984.

[SK96] E. Simon and J. Kiernan. The A-RDL System. In J. Widom and S. Ceri, editors, *Active Database Systems*, pages 111–149. Morgan Kaufmann, 1996.

[vEK76] M.H. van Emden and R.A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742, 1976.

[Via96] V. Vianu. Rule-Based Languages, 1996. To appear in *Annals of Mathematics and Artificial Intelligence*.

[Wid94] J. Widom. Deductive and Active Databases: Two Paradigms or Ends of a Spectrum? In N.W. Paton and M.H. Williams, editors, *Rules in Database Systems*, pages 306–315. Springer-Verlag, 1994.

[Zan94] C. Zaniolo. A unified semantics for active and deductive databases. In N.W. Paton and M.H. Williams, editors, *Rules in Database Systems*. Springer-Verlag, 1994.