A Unified Semantics for Active and Deductive Databases

Carlo Zaniolo
Computer Science Department
University of California
Los Angeles, California U.S.A.
zaniolo@cs.ucla.edu

Abstract
These two rule-oriented paradigms of databases have been the focus of extensive research and are now coming of age in the commercial DBMS world. However, the systems developed so far support well only one of the two paradigms—thus limiting the effectiveness of such systems in many applications that require complete integration of both kinds of rules. In this paper, we discuss the technical problems that make such an integration difficult, and trace their roots to a lack of a unified underlying semantics. Then, we review recent advances in the semantics of non-monotonic logic and show that they can be used to unify the foundations of active databases and deductive databases. Finally, we outline the design a new rule language for databases that integrates a deductive system with a trigger-based DBMS.

1 Introduction
Rules provide the main paradigm for expressing computation in active databases and deductive databases. The unification of the two paradigms represents a research problem of obvious theoretical interest and many important applications could benefit from it; yet, there has been little formal work on the marrying these powerful paradigms. While cultural and historical biases share the blame for this chasm, the root of the problem is actually technical and can be traced to certain semantic inadequacies in the conceptual foundations of both approaches.

Several active database languages and systems have been proposed so far: a very incomplete list include [4, 8, 20, 21, 28]. However, there is is no unifying semantic theory for active databases: most of the work done so far has concentrated on explaining operational semantics of particular systems. On the contrary, not one but three equivalent formal semantics exist for Horn clauses that form the core of deductive database languages [29, 18]. Unfortunately, this elegant semantics is brittle and can not be generalized easily to deal with non-monotonic constructs, such as negation and updates. Similar non-monotonic reasoning problems have emerged in the areas of knowledge representation and of logic programming and remain the focus of intense research. While deductive databases encountered early successes in this problem area (e.g., with the introduction of the concept of stratified negation), recent progress has been very
slow. No totally satisfactory semantics currently exist for programs which use non-monotonic constructs such as negation and aggregates in recursion—and the problem of updates in recursion is understood even less. Also, the problem of finding a logic for reasoning about actions and situations represents one of the classical challenge of AI [17].

Given this rather ominous background, the solution presented in this paper is surprisingly simple and general. We introduce the notion of XY-stratified programs that allow non-monotonic constructs in recursive rules. Then, we show that a formal semantics for updates and triggers in databases can be given using XY-stratified programs. The blueprints for the design of a unified rule language for active databases and deductive databases follow from such a solution.

2 Logic-Based Semantics

This area has benefited significantly from research in deductive databases. The adoption of the fixpoint-based bottom-up approach to define the declarative and constructive semantics of logic programs led almost immediately to the concept of stratified negation and stratified set aggregates [24]. This concept removes several of the limitations and problems of Prolog's negation-by-failure, and it is conductive to efficient implementation, as demonstrated by systems such as Glue-Nail, LDI and CORAL [23, 5, 26]. However, experience gained with real-life applications [33] revealed that stratification is too restrictive and there remain many important applications where negation and set aggregates are needed: such applications, range from processing a Bill of Materials to finding the shortest path in a graph [33].

Therefore, during the last five years, a substantial research effort has been devoted to solving the non-stratification issue. This endeavor has produced significant progress on the theoretical front, with the introduction of concepts such as locally stratified programs, well-founded models [12], and the stable models[9], but it has not yet begotten a solution that is both general and practical. Indeed a practical solution must satisfy three difficult requirements, inasmuch as it must

- have a formal logic-based semantics,
- have a simple and intuitive constructive semantics,
- be amenable to efficient implementation.

Thus, in addition to requiring formal semantics and efficient implementation, any practical proposal must also stress the importance of having a simple concrete semantics: i.e., one that can be easily comprehended by the application programmer, without a need to understand abstract formalisms. For instance, a notion such as stratification can be mastered by a programmer, who can make full use of it without having to understand its formal perfect-model semantics. Furthermore, it is simple for a compiler to verify that stratification is satisfied, and then support stratified programs by an efficient bottom-up computation. However, an understanding of the logical formalism is required to understand notions such as well-founded models or stable models. Furthermore, no simple syntactic check exists for deciding whether a program has a well-founded