DATALOG Queries *
with Stratified Negation and Choice:
from \( \mathcal{P} \) to \( \mathcal{D}^P \)

Sergio Greco,¹ Domenico Saccà¹ and Carlo Zaniolo²

¹ DEIS, Univ. della Calabria, 87030 Rende, Italy
{ greco, sazca }@si.deis.unical.it
² Computer Science Dept., Univ. of California, Los Angeles, CA 90024
zaniolo@cs.ucla.edu

Abstract. This paper introduces a unified solution to the problem of extending stratified DATALOG to express DB-complexity classes ranging from \( \mathcal{P} \) to \( \mathcal{D}^P \). The solution is based on (i) stratified negation as the core of a simple, declarative semantics for negation, (ii) the use of a “choice” construct to capture non-determinism of stable models (iii) the ability to bind a query execution to the complexity class that includes the problem at hand, and (iv) a general algorithm that ensures efficient execution for the different complexity classes. We thus obtain a class of DATALOG programs that preserves computational tractability, while achieving completeness for a wide range of complexity classes.

1 Introduction

The issue of designing declarative, logic-oriented database languages with sufficient expressive power is the key motivation of much of current research on databases and knowledge bases. The introduction of DATALOG represents a major breakthrough in this line of work, due to DATALOG’s ability of expressing recursive queries. DATALOG is a rule-based language that has simple and elegant semantics based on the notion of minimal model—or equivalently, on the notion of least fixpoint. This second semantics leads to an operational semantics that is amenable to very efficient implementation as demonstrated by recent work on deductive database systems.

Unfortunately, the basic DATALOG language (no negation or function symbols) is still severely limited in its expressive power and cannot express many of the queries of practical interest. The exact expressive power of DATALOG is not yet understood but it has been shown that DATALOG only captures a proper subset of the monotonic polynomial-time queries [3].

In order to support non-monotonic queries, negation is allowed in the bodies of the rules. Of a particular interest is stratified negation, which avoids the semantic and implementation problems connected with the use of non-monotonic

* The work of the first two authors has been supported by the CNR project ”Sistemi Informatici e Calcolo Parallelo” and by the MURST project ”Metodi Formali per Basi di Dati”.

¹ The work of the first two authors has been supported by the CNR project "Sistemi Informatici e Calcolo Parallelo" and by the MURST project "Metodi Formali per Basi di Dati".
constructs in recursive definitions [4, 6, 27]. We will write DATALOG\~{\~{\textasciitilde}} to denote DATALOG with stratified negation. Simple, intuitive semantics, leading to efficient implementation exists for DATALOG\~{\~{\textasciitilde}}. Unfortunately, DATALOG\~{\~{\textasciitilde}} cannot express all polynomial-time queries, and can only express a proper a proper subset of fixpoint queries [14].

The simplest step toward greater expressive power is to remove the condition that negation must be stratified. Unfortunately, the relaxation of the stratification assumption for non monotonic programs opens a pandora box of semantic and computational problems. Take for instance the concepts of stable models [8] and well-founded models [28]. The well-founded model semantics defines a unique "intended" model for each program, which can be computed in polynomial time. Several programs, however, do not have a well-founded total model: the meaning of portions of these programs, therefore, remain undefined. In terms of expressibility, well-founded semantics can express all fixpoint queries, but neither can express all polynomial time queries, nor it can express queries in DB-\text{NP}.

A dramatic leap in expressive power is provided by the concept of stable models, which has emerged as the compendium of many concepts and theories developed over the years by AI researches working on non-monotonic reasoning, default theories and autoepistemic logic. This gain in expressive power, however, is not without complications. One is the non-deterministic nature of such semantics that follows from the fact that a program can have several stable models. Nevertheless, deterministic query semantics can be ensured by requesting answers that hold true for some stable model (possibility semantics), or for all stable models, (certainty semantics); these are known in AI as membership and entailment semantics, respectively. The two semantics capture the classes \text{NP} and co\text{NP}, respectively. However, there remain the following two problems:

1. The usage of unrestricted negation in programs is often not simple nor intuitive, and, e.g., might lead to the writing of programs for which no total stable model occurs (a similar problem also exists for well-founded models)
2. exponential-time algorithms are needed to compute stable models (as expected in situations where programs express NP-hard problems). Unfortunately, we do not know how to ensure that these exponential algorithms compute efficiently on programs that solve polynomial-time problems.

With current research focusing on overcoming these limitations, several proposals have been put forward that give up declarative semantics and fall-back on procedural semantics, e.g., based on the inflationary fixpoint computation procedure [2, 1, 15]. This paper is largely inspired by the conviction that model-theoretic semantics offers important benefits, and, therefore, we should strive to preserve it. Thus, we propose a novel approach whereby stratified DATALOG programs are extended to include (1) an explicit non-deterministic construct called choice whose semantics is defined in terms of stable models but can be implemented very efficiently, and (2) the use of quantifiers in the query goals to achieve the desired levels of expressive power and computational efficiency in a more controlled fashion, that allows a user to match the expressive power of the semantic used with the intrinsic complexity of the problem at hand.