Rewriting and Efficient Computation of Bound Disjunctive Datalog Queries

Sergio Greco  
DEIS Università della Calabria, Italy

Ester Zumpano  
DEIS Università della Calabria, Italy

INTRODUCTION

A strong interest in enhancing Datalog programs by the capability of disjunction emerged in several areas, such as databases, artificial intelligence, logic programming, and so forth (Lobo, Minker & Rajasekar, 1992). Disjunctive rules have been profitably used in several contexts including knowledge representation, databases querying, and representation of incomplete information (Eiter, Gottlob & Mannila, 1997a; Gelfond & Lifschitz, 1991). Most of the works on disjunction are concerned with the definition of intuitive and expressive semantics, which are commonly based on the paradigm of minimal models (Abiteboul, Hull & Vianu, 1995; Lobo et al., 1992; Ullman, 1989).

Disjunction in the head of rules increases the expressive power of the language but makes the computation of queries very difficult, as it allows the presence of multiple models whose number is generally exponential with respect to the size of the input (Abiteboul et al., 1995; Eiter et al., 1997a).

Therefore, a great interest has been devoted to the definition of efficient algorithms for both computing the semantics of programs and answering queries.

Most of the research has concentrated on the definition of efficient fixpoint algorithms computing the semantics of programs, and the more effective proposals are based on the evaluation of the (intelligent) ground instantiation of programs and on the use of heuristics (Leone, Rullo & Scarcello, 1997). However, different from standard datalog, for disjunctive queries, there are few effective methodologies which systematically utilize the query to propagate bindings into the body of the rules to avoid computing all the models of the program.

The following example taken from Greco (2003), shows a program in which only a strict subset of the minimal models needs to be considered to answer the query.

Example 1. Consider the disjunctive program P1 consisting of the following rule:

\[ p(X) \lor q(X) \rightarrow a(X, Y) \]

and a database \( D \) consisting of the set of facts \{a(1,2), a(2,3),..., a(k,k+1)\}. Consider now a query asking if there is some model for \( P \cup D \) containing the atom \( p(1) \). A “brute force” approach, based on an exhaustive search of the minimal models of \( P \cup D \), would consider \( 2^k \) minimal models.

However, to answer the query, we could only consider the ground rule: \( p(1) \lor q(1) \rightarrow a(1, 2) \) and, therefore, only two minimal (partial) (for partial model we intend a subset of the model containing all the atoms necessary to answer the query) models: \( M_1 = \{ p(1) \} \cup D \) and \( M_2 = \{ q(1) \} \).

From the above example, it is evident that using the query goal to reduce the size and the number of models to be considered for answering the query is necessary.

Constraints play an active role in the deductive database process as they model the interaction among data and define properties that have to be satisfied by models.

Thus, this article explores the efficient computation of bound queries over disjunctive Datalog programs enriched with constraints.

The technique here proposed is very relevant for the optimization of queries expressing hard problems. Indeed, the usual way of expressing declaratively hard problems, such as NP problems and problems in the second level of the polynomial hierarchy, is based on the guess-and-check technique, where the guess part is expressed by means of disjunctive rules, and the check part is expressed by means of constraints. However, as shown by the following example, the presence of constraints reduces the number of feasible models but could increase both the number of ground rules and the number of (partial) models to be considered for answering the query.

Example 2. Consider the query goal \( p(1) \) over the disjunctive program \( P_2 \) consisting of the rule in the above example plus the following constraint (a rule with empty head which is true if the body is false)

\[ \leftarrow p(X), a(X, Y), q(Y), X \leq 1. \]
To answer the query, we have to consider other than the ground rule:

\[ r1: \ p(1) \lor q(1) \leftarrow a(1,2). \]
also the ground rule:
\[ r2: \ p(2) \lor q(2) \leftarrow a(2,3). \]
as the atom \( p(1) \) in the head of the rule \( r1 \) is influenced by the atom \( q(2) \) through the ground constraint \( c1 \)

- \( p(1), a(1,2), q(2), 1 \leq 1. \)

Consequently, there are three stable models to be considered:

\[ N1 = \{ p(1), p(2) \} \cup D, \]
\[ N2 = \{ q(1), p(2) \} \cup D, \] and
\[ N3 = \{ q(1), q(2) \} \cup D. \]

The interpretation \( \{ p(1), q(2) \} \cup D \) is not a model as it does not satisfy the constraint.

The above example shows the necessity of an effective methodology that uses the properties specified by the constraints and the query goal in order to compute correct answers avoiding the evaluation of useless models. However, in the presence of constraints, we have to consider ground rules defining atoms on which the query goal depends on (both through standard rules and constraints), but also to check that constraints are satisfied by the whole set of ground rules.

**BACKGROUND**

In the literature, different approaches have been proposed for the efficient bottom-up evaluation of queries, for example, the Magic set (Bancilhon et al., 1986), the Counting (Beeri & Ramakrishnan, 1991), and the supplementary magic set and other specialized rewriting techniques (Beeri & Ramakrishnan, 1991; Greco, Saccà & Zaniolo, 1995; Ramakrishnan et al., 1993; Ullman, 1989). The key idea of all these techniques consists in the rewriting of deductive rules with respect to the query goal to answer the query without actually computing irrelevant facts. In this section, we recall the magic-set rewriting technique for Datalog queries as its generality and efficiency makes of this approach a kind of standard in the field.

Although the magic-set can be applied to general Datalog queries, for the sake of simplicity, we present here the technique for linear programs, that is, programs whose rules contain at most one body predicate mutually recursive with the head predicate.

The magic-set method consists of three separate steps:

- **An Adornment step** in which the relationship between a bound argument in the rule head and the bindings in the rule body is made explicit.
- **A Generation step** in which the adorned program is used to generate the magic rules which simulate the top-down evaluation scheme.
- **A Modification step** in which the adorned rules are modified by the magic rules generated in Step 2; these rules will be called modified rules.

An adorned program \( P^\alpha \) is a program whose predicate symbols have associated a string \( a \) defined on the alphabet \( \{ b, f \} \) of length equal to the arity of the predicate. A character \( b \) (resp. \( f \)) in the \( i \)-th position of the adomrnt associated with a predicate \( p \) means that the \( i \)-th argument of \( p \) is bound (resp. free).

The adornment step consists in generating a new program whose predicates are adorned. Given a rule \( r \) and an adornment \( \alpha \) of the rule head, the adorned version of \( r \) is derived as follows:

- Identify the distinctive arguments of the rules: an argument is distinctive if it is bound in the adornment \( \alpha \), is a constant or appears in a base predicate of the rule-body which includes an adornment argument;
- Assume that the distinctive arguments are bound and use this information in the adornment of the derived predicates in the rule body.
- Adornments containing only \( f \) symbols can be omitted.

Given a query \( Q = \langle q(T), P \rangle \) and letting \( \alpha \) be the adornment associated with \( q(T) \), the set of adorned rules for \( Q \) is generated by (1) computing the adorned version of the rules defining \( q \) and (2) generating, for each new adorned predicate \( p \) introduced in the previous step, the adorned version of the rules defining \( p \) w.r.t. \( \alpha \); Step 2 is repeated until no new adorned predicate is generated.

The second step of the process consists in using the adorned program for the generation of the magic rules. For each of the adorned predicates in the body of the adorned rule:

- Eliminate all the derived predicates in the rule body which are not mutually recursive with the rule head;
- Replace the derived predicates symbol \( p^\alpha \) with \( \text{magic}_p^\alpha \) and eliminate the variables which are free w.r.t. \( \alpha \);
- Replace the head predicates symbol \( q^\alpha \) with \( \text{magic}_q^\alpha \) and eliminate the variables which are free w.r.t. \( \alpha \); and
Related Content

Process-Embedded Data Integrity
www.igi-global.com/article/process-embedded-data-integrity/3307?camid=4v1a

Online Data Mining
www.igi-global.com/chapter/online-data-mining/11184?camid=4v1a

Improving the Understandability of Dynamic Semantics: An Enhanced Metamodel for UML State Machines
Eladio Dominguez, Angel L. Rubio and María A. Zapata (2004). *Advanced Topics in Database Research, Volume 3* (pp. 70-89).
www.igi-global.com/chapter/improving-understandability-dynamic-semantics/4354?camid=4v1a

A Survey of Approaches to Web Service Discovery in Service-Oriented Architectures
www.igi-global.com/article/survey-approaches-web-service-discovery/49725?camid=4v1a