TLP **5** (6): 713–745, 2005. © 2005 Cambridge University Press
doi:10.1017/S1471068404002273 Printed in the United Kingdom

*Optimization of bound disjunctive queries with constraints**

GIANLUIGI GRECO, SERGIO GRECO, IRINA TRUBITSYNA
and ESTER ZUMPANO

DEIS, Università della Calabria, 87030 Rende, Italy
(*e-mail:* {ggreco,greco,irina,zumpano}@si.deis.unical.it)

submitted 5 June 2003; revised 30 January 2003; accepted 3 June 2004

Abstract

This paper presents a technique for the optimization of bound queries over disjunctive deductive databases with constraints. The proposed approach is an extension of the well-known Magic-Set technique and is well-suited for being integrated in current bottom-up (stable) model inference engines. More specifically, it is based on the exploitation of binding propagation techniques which reduce the size of the data relevant to answer the query and, consequently, reduces both the complexity of computing a single model and the number of models to be considered. The motivation of this work stems from the observation that traditional binding propagation optimization techniques for bottom-up model generator systems, simulating the goal driven evaluation of top-down engines, are only suitable for positive (disjunctive) queries, while hard problems are expressed using unstratified negation. The main contribution of the paper consists in the extension of a previous technique, defined for positive disjunctive queries, to queries containing both disjunctive heads and constraints (a simple and expressive form of unstratified negation). As the usual way of expressing declaratively hard problems 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, the technique proposed here is highly relevant for the optimization of queries expressing hard problems. The value of the technique has been proved by several experiments.

KEYWORDS: Disjunctive Datalog, Constraints, Theory, Magic set technique

1 Introduction

Disjunctive Datalog programs, i.e. programs allowing clauses to have both disjunction in their heads and negation in their bodies (short. Datalog$^{\vee,\neg}$ programs), have been successfully introduced with the aim of modelling incomplete data (Lobo *et al.* 1992). Over the last few years, there has been a great deal of interest in studying

* A preliminary version of this paper was presented at the LPAR'02 Conference (Greco *et al.* 2002). The second author is also supported by ICAR-CNR.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

declarative semantics for Datalog$^{\vee,\neg}$ programs. In fact, minimal model semantics (Grant and Minker 1986) is widely accepted for programs in the absence of body negation (Datalog$^{\vee}$ programs), and is naturally extended for programs with (possibly unstratified) negation. For Datalog$^{\vee,\neg}$ programs there are several extensions of the minimal model semantics such as disjunctive stable models (Przymusinski 1991; Gelfond and Lifschitz 1991) minimal founded semantics (Furfaro *et al.* 2004), and different proposals of well-founded semantics (Ross 1990; Brass and Dix 1999; Przymusinski 1995; Eiter *et al.* 1997; Baral *et al.* 1992; Wang 2000).

For each of the above semantics, several algorithms and techniques for model generation have also been proposed. Model generation is often carried out through bottom-up evaluation of clauses (Minker and Rajasekar 1990; Brass and Dix 1994; Fernández and Minker 1995; Leone *et al.* 2002; Simons *et al.* 2002).

For instance, bottom-up methods have been employed to compute perfect and stable models (Fernández and Minker 1995; Fernández *et al.* 1995) using ordered model trees, and to process logic programs under the minimal model semantics using a tableau calculus (Niemelä 1996).

This paper focuses on stable model semantics, according to which a disjunctive program may have several alternative models, called *answer sets*, each one corresponding to a possible view of the world – see Niemelä (2003) for a recent overview of *answer set programming*. Disjunctive logic programs under stable model semantics are very expressive, since they capture the complexity class Σ_2^P (Eiter *et al.* 1997) and they have been used for developing several practical applications. Furthermore, in the last few years several efficient inference engines implementing stable model semantics have been developed; the DLV system (Leone *et al.* 2002) and the GnT system (Janhunen *et al.* 2000) should be recalled which implement disjunctive stable models, while for the non-disjunctive case many other engines are currently available (e.g see *Smodels* (Niemelä and Simons 1997), DeReS (Cholewiński *et al.* 1996), and ASSAT (Lin and Zhao 2002)).

Even though model generator techniques for stable model semantics are quite useful for knowledge representation and reasoning tasks, it is well-known that they are inefficient when used for refutations, i.e. query answering. In fact, they often explore a search space much larger than that required, and tend to generate answers to all the possible queries rather than to the precise query.

However, it is often the case that only a strict subset of the stable models needs to be considered and that there is no need to compute these models in their entirety to answer the query. This intuition is exploited by top-down techniques which only consider atoms necessary to answer the current query and outperform model generators used for refutation. In fact, top-down approaches systematically utilize the query to propagate the binding into the body of the rules to avoid computing all the models of the program – a brief overview of top-down (disjunctive) reasoning is presented in the next section.

To optimize query evaluation, while still preserving the ability to compute models (well-suited for complex reasoning tasks), several works proposed the simulation of top-down strategies, by means of suitable transformations introducing new predicates and rewriting some clauses.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

Among them, the Magic-Set technique is the best-known technique. This is due to its efficiency and its generality, even though other focused methods, such as the supplementary Magic-Set and other special techniques for linear and chain queries have also been proposed (Beeri and Ramakrisnhan 1991; Greco *et al.* 1995; Ullman 1989b; Ramakrishnan *et al.* 1993).

It should be recalled that the formal equivalence between top-down evaluation with memoing and bottom-up evaluation with Magic-Set optimization is well-known (Ullman 1989b, 1989a). However, this Magic-Set optimization technique is only suitable for positive Datalog queries, i.e. queries without disjunction and negation. To the best of our knowledge the first extension of the Magic-Set technique for the evaluation of disjunctive Datalog$^{\vee}$ queries, in which negation is not allowed, was introduced in Greco (1999) and extended in (Greco 2003).

This paper further investigates optimization techniques simulating the top-down evaluation of queries, by extending the proposal presented in Greco (2003) from Datalog$^{\vee}$ programs to Datalog$^{\vee,\neg}$ programs. Actually, a syntactic restriction of Datalog$^{\vee,\neg}$ programs is considered in which unstratified negation can only be expressed in the form of constraint rules. Notice that this is not truly a restriction. In fact, constraints represent a natural way to extend database semantics, by explicitly defining properties which are supposed to be satisfied by all instances over a given database schema. Moreover, the usual way of expressing declaratively hard problems, such as NP problems and problems in the second level of the polynomial hierarchy (Σ_p^2 and Π_p^2 problems), 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. Therefore, the technique proposed here is highly relevant for the optimization of queries expressing hard problems.

Even though constraints can be easily managed in top-down approaches, it should be pointed out they represent a serious issue when simulating top-down reasoning by means of the Magic-Set technique. In fact, all the rewriting techniques presented so far in the literature cannot be straightforwardly extended to constraint rules.

1.1 Contributions

The main contributions of the paper are as follows.

- A query rewriting algorithm is defined which allows the simulation of top-down computation in bottom-up inference engines by propagating the bindings from the query-goal into the body of rules. The rewriting technique avoids the computation of complete models and useless models which are not necessary to answer the query. Essentially the technique is an adaptation of the Magic-Set technique (Bancilhon *et al.* 1986; Beeri and Ramakrisnhan 1991) to disjunctive Datalog programs with constraints.
- It is observed that the application of the Magic-Set technique to queries with constraints produces a query that can be evaluated more efficiently, but, unfortunately, the technique may produce a query that, generally, is not equivalent to the original one. The conditions that a program must satisfy in order to preserve this equivalence are investigated.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

- The proposed algorithm is an extension of Greco (2003) for Datalog$^{\vee}$ programs only. However, an in-depth analysis is also provided of the algorithm in Greco (2003), formally showing how it can work independently of the particular strategy adopted for simulating the binding occurring in top-down evaluation. In this way, this approach is orthogonal to the Magic-Set technique, since it can exploit any other rewriting strategy proposed in the literature.
- To verify the validity of this approach the technique has been tested with different disjunctive programs using the DLV system (Leone *et al.* 2002). The experiments comparing the execution time required to answer the source and the optimized query, have achieved very encouraging results proving the value of the approach.

Even though there are a number of proposals for efficient query answering in disjunctive deductive databases under well-founded semantics, there are few effective techniques for top-down reasoning under stable model semantics. Thus, the paper also contributes to provide an effective, implemented technique for query answering under stable model semantics.

Finally, it should be stressed that even though the results in Ullman (1989b, 1989a) suggest that this extension of the Magic-Set technique is equivalent to the form of binding propagation occurring in the top-down evaluation, the technique may suffer from some potential inefficiency due to the computation of additional predicates and rules needed in the rewriting (intrinsic in the Magic-Set technique). However, note that some redundancy can also occur in the top-down evaluation due to the tabling of intermediate results.

However, the main aim here is to formally prove the possibility of efficient query evaluation in bottom-up systems, breaking down the complexity of a brute force approach and still using the internal model generator. Thus, implementations of top-down systems supporting disjunction and stable model semantics have not been sought in order to make an experimental comparison w.r.t. this approach. This aspect calls for further study and research.

1.2 Related work on the evaluation of disjunctive programs

The (efficient) evaluation of disjunctive logic programs has a long history (Grant and Minker 1986; Yahya and Henschen 1985; Liu and Sunderraman 1990; Yuan and Chiang 1998). Recently there has been a growing interest in answering queries on disjunctive deductive databases. The two main approaches proposed in the literature for the evaluation of queries and programs are now discussed.

1.2.1 Bottom-up techniques

The definition of efficient bottom-up evaluation algorithms for assigning semantics to disjunctive deductive databases has been the subject of several proposals. In the following some of these approaches proposed in the literature are briefly described.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

Minker and Rajasekar (1990) introduce the concept of *state*, consisting of a set of positive disjunctions, as the domain of a fixpoint operator that gives semantics to disjunctive logic programs. The fixpoint computation operates bottom-up and produces, as the resulting fixpoint, the *model state*, i.e. a state whose minimal models satisfy the disjunctive deductive database.

Brass and Dix (1994) propose a general approach for defining the semantics of disjunctive logic programs. The framework consists of: a semantical part, where the declarative meaning of a program is defined in an abstract way as the weakest semantics satisfying certain properties, and a procedural part, namely a bottom-up query evaluation method based on operators working on conditional facts. More specifically, the approach is based on the generation of a residual program, i.e. a program obtained by transforming the original one, which makes the use of disjunctive information explicit.

Fernández and Minker (1995) introduce a new fixpoint characterization of the minimal models of disjunctive logic programs. The proposed operator, applied iteratively, is shown to characterize the perfect model semantics of stratified disjunctive logic programs. Based on these results the authors present a bottom-up evaluation algorithm for stratified disjunctive deductive databases that uses the model-tree data structure to both represent the information contained in the database and compute answers to queries.

Leone *et al.* (2002) propose the DLV system, which exploits an algorithm for computing stable models for disjunctive logic programs. This approach searches for stable models by using efficient fixpoint algorithms computing the semantics of programs. In particular, it obtains effective performances by using an (intelligent) ground instantiation of programs, i.e. a program in which unsatisfiable ground rules are deleted, and heuristics are used for implementing a backtracking search strategy for pruning the search space.

Simons *et al.* (2002) define a novel answer set programming language that generalizes normal logic programs. The language allows weighted constraint rules which increase the expressivity of the language to express different types of constraint (e.g. cardinality conditions) optimization capabilities. The declarative semantics extends the one for normal programs while the complexity of computing stable models for this novel language is comparable to that of normal programs (without considering optimizations).

1.2.2 Top-down techniques

Top-down query evaluation is based on refutation procedures. The first of such refutation techniques, called *SLD-resolution*, was introduced in Kowalski (1984), and is suitable for Datalog programs only, i.e. programs without negation and disjunction. Clark (1978) extended *SLD-resolution* to *SLDNF-resolution*, introducing the negation as a failure rule for inferring negative information.

In order to prevent the possibility of infinite loops and a large amount of redundant sub-derivations (intrinsic in *SLD-resolution*), the *SLG-resolution* was introduced (Chen and Warren 1993, 1996). It is a tabling mechanism for the

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

evaluation of the well founded semantics of logic programs (without disjunction), and it is the evaluation strategy underlying XSB, the best known state-of-the-art top-down tabling system (Rao *et al.* 1997). Shen *et al.* (2002) present an optimization of *SLG-resolution*, called *SLT-resolution*.

The problem of top-down computation for disjunctive well-founded semantics (DWFS) was investigated in Wang (2001). Specifically, the author proposes a top-down procedure for disjunctive well-founded semantics called *D-SLS-resolution*, which can eventually be optimized by employing some techniques such as the tabling method of Chen and Warren (1996). A top-down method for testing DWFS membership, based on the characterization of the DWFS in terms of Gelfond–Lifschitz transformations, is presented in Johnson (2001).

A bottom-up procedure computing queries in a top-down fashion has also been proposed for minimal model semantics (Yahya 2000, 2002). The approach, suitable for positive queries, is based on the *duality* principle for interpreting logical connectives. In more detail, the duality transformation is obtained by reversing the direction of the implication arrows in the clauses representing both the program and the negation of the query goal. The application of a generic bottom-up procedure to the transformed clause set results in a top-down query answering.

Finally, Johnson (1999) shows that disjunctive stable models can be characterized in terms of cyclic covers, and in particular that such covers provide a powerful technique for characterizing the properties of query processing, compilation and view updating. A (correct and terminating) top-down method for query processing under the disjunctive stable model semantics is also presented. Supported covers have also been used to facilitate the top-down query processing under the possible model semantics (Johnson 2002).

1.3 Plan of the paper

The paper is organized as follows. Section 2 presents preliminary definitions and results on disjunctive Datalog, minimal and stable model semantics and Magic-Set rewriting; Section 3 presents the Magic-Set rewriting for positive disjunctive queries; Section 4 extends the binding propagation technique to disjunctive queries with constraints; Section 5 presents experimental results showing the validity of the proposed technique; finally, Section 6 presents the conclusions.

2 Preliminaries

In this section standard concepts on Datalog, Magic-set rewriting, query equivalence, disjunction and constraints are reviewed.

2.1 Disjunctive Datalog

The existence of alphabets of constants, variables and predicate symbols are aaumed. A *term* is a constant or a variable. An *atom* is of the form $p(t_1, \ldots, t_k)$ where p is a

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

k-ary predicate symbol and $t_1, \ldots, t_k$ are terms. A *literal* is an atom A or its negation $\neg A$. A *Datalog*$^{\vee,\neg}$ (or simply *disjunctive Datalog*) rule r is a clause of the form:

$$a_1 \vee \cdots \vee a_m \leftarrow b_1, \cdots, b_k, \neg c_1, \cdots, \neg c_n$$

where $n, k, m \geqslant 0$, $n + k + m > 0$ and $a_1, \cdots, a_m, b_1, \cdots, b_k, c_1, \cdots, c_n$ are function-free atoms. The disjunction $a_1 \vee \cdots \vee a_m$ is called the *head* of r and is denoted by $Head(r)$ while the conjunction $b_1, \cdots, b_k, \neg c_1, \cdots, \neg c_n$ is called the *body* and is denoted by $Body(r)$. If $m = 1$, then r is *normal* (i.e. $\vee$-free) or Datalog$^{\neg}$; if $n = 0$, then r is *positive* (i.e. $\neg$-free) or Datalog$^{\vee}$; if both $m = 1$ and $n = 0$, then r is *normal and positive* or Datalog; if $k = n = 0$ r is a fact, whereas if $m = 0$ r is a *constraint or denial rule*, i.e. a rule which is satisfied only if $Body(r)$ is false. A *Datalog*$^{\vee,\neg}$ *program* $\mathscr{P}$ is a finite set of Datalog$^{\vee,\neg}$ rules; it is *normal* (resp. *positive*) if all its rules are normal (resp. positive). Given a program $\mathscr{P}$ and a predicate symbol g occurring in $\mathscr{P}$, the definition of g, denoted by $def(g, \mathscr{P})$, consists of all rules in $\mathscr{P}$ having g in their heads.

The *Herbrand Universe* $U_{\mathscr{P}}$ of a program $\mathscr{P}$ is the set of all constants appearing in $\mathscr{P}$, and its *Herbrand Base* $B_{\mathscr{P}}$ is the set of all ground atoms constructed from the predicates appearing in $\mathscr{P}$ and the constants from $U_{\mathscr{P}}$. A *ground* term (resp. an atom, a literal, a rule or a program) is a term (resp. an atom, a literal, a rule or a program) where no variables occur in it. A rule r' is a *ground instance* of a rule r, if r' is obtained from r by replacing every variable in r with some constant in $U_{\mathscr{P}}$. We denote by $ground(\mathscr{P})$ the set of all ground instances of the rules in $\mathscr{P}$.

Given a set of ground atoms $\mathscr{I}$, a program $\mathscr{P}$, a predicate symbol g and an atom $g(t)$, $\mathscr{I}[g]$ denotes the set of g-atoms in $\mathscr{I}$, whereas $\mathscr{I}[\mathscr{P}]$ denotes the set of atoms in $\mathscr{I}$ whose predicate symbol appears in the head of some rule of $\mathscr{P}$. Given a set of interpretations S, then $S[g] = \{M[g] | M \in S\}$ and $S[\mathscr{P}] = \{M[\mathscr{P}] | M \in S\}$.

An interpretation of $\mathscr{P}$ is any subset of $B_{\mathscr{P}}$. The value of a ground atom L w.r.t. an interpretation I, $value_I(L)$, is 1 (*true*) if $L \in I$ and 0 (*false*) otherwise. The value of a ground negated literal *not* L is $1 - value_I(L)$. The truth value of a conjunction of ground literals $C = L_1, \ldots, L_n$ is the minimum over the values of the L_i, i.e. $value_I(C) = min(\{value_I(L_i) \mid 1 \leqslant i \leqslant n\})$, while the value of a disjunction $D = L_1 \vee \ldots \vee L_n$ is its maximum, i.e. $value_I(D) = max(\{value_I(L_i) \mid 1 \leqslant i \leqslant n\})$; if $n = 0$, then $value_I(C) = true$ and $value_I(D) = false$. A ground rule r is *satisfied* by I if $value_I(Head(r)) \geqslant value_I(Body(r))$. Thus, a rule r with empty body is satisfied by I if $value_I(Head(r)) = true$. An interpretation M for $\mathscr{P}$ is a model of $\mathscr{P}$ if M satisfies all rules in $ground(\mathscr{P})$.

The *model-theoretic semantics* for a positive program $\mathscr{P}$ assigns the set of its *minimal models* $\mathscr{MM}(\mathscr{P})$. A model M for $\mathscr{P}$ is minimal, if no proper subset of M is a model for $\mathscr{P}$ (Minker 1982). Accordingly, the program $\mathscr{P} = \{a \vee b \leftarrow\}$ has the two minimal models $\{a\}$ and $\{b\}$, i.e. $\mathscr{MM}(\mathscr{P}) = \{\ \{a\},\ \{b\}\ \}$. The more general *disjunctive stable model semantics* generalizes stable model semantics, previously defined for normal programs (Gelfond and Lifschitz 1988) and also applies to programs with (unstratified) negation (Gelfond and Lifschitz 1991; Przymusinski 1991).

Let $\mathscr{P}$ be a logic program $\mathscr{P}$ and let I be an interpretation for $\mathscr{P}$, $\mathscr{P}^I$ denotes the ground positive program derived from $ground(\mathscr{P})$ by (1) removing all rules that contain a negative literal $\neg a$ in the body and $a \in I$, and (2) removing all negative

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

literals from the remaining rules. An interpretation M is a (disjunctive) stable model for $\mathscr{P}$ if and only if $M \in \mathscr{MM}(\mathscr{P}^M)$.

For general $\mathscr{P}$, the stable model semantics assigns to $\mathscr{P}$ the set $\mathscr{SM}(\mathscr{P})$ of its *stable models*. It is well known that stable models are minimal models (i.e. $\mathscr{SM}(\mathscr{P}) \subseteq \mathscr{MM}(\mathscr{P})$) and that for negation-free programs minimal and stable model semantics coincide (i.e. $\mathscr{SM}(\mathscr{P}) = \mathscr{MM}(\mathscr{P})$) and that Datalog programs have a unique minimal model.

Predicate symbols can be either extensional (i.e. defined by the ground facts of a database – *EDB predicate symbols*), also called *base predicates*, or intensional (i.e. defined by the rules of the program – *IDB predicate symbols*), also called *derived predicates*. Thus a database D consists of a set of ground facts having in the head a base predicate symbol (i.e. ground normal rules with empty body defining base predicates), whereas a program $\mathscr{P}$ consists of a set of (disjunctive) rules having in the heads derived predicate symbols.

A disjunctive Datalog *query* over a database defines a mapping from the database to a finite (possibly empty) set of finite (possibly empty) relations for the goal. A query is a pair $\langle G, \mathscr{P} \rangle$ where G is an atom, called a *goal*, and $\mathscr{P}$ is a program. The application of a query $\mathscr{Q}$ to a database D is denoted by $\mathscr{Q}(D)$ and the union of the program $\mathscr{P}$ and the facts in D is denoted by $\mathscr{P}_D$. Clearly, all models for $\mathscr{P}_D$ contain the database D.

The result of a query $\mathscr{Q} = \langle G, \mathscr{P} \rangle$ on an input database D is defined in terms of the stable models of $\mathscr{P}_D$, by taking either the union (*possible inference*) or the intersection (*certain inference*) of all models. Thus, given a program $\mathscr{P}$ and a database D, a ground atom G is true, under possible (brave) semantics, if there exists a stable model M for $\mathscr{P}_D$ such that $G \in M$. Analogously, G is true, under certain (cautious) semantics, if G is true in every stable model for $\mathscr{P}_D$.

Given an atom G and an interpretation M, $A(G, M)$ denotes the set of substitutions for the variables in G such that G is true in M. The answer to a query $\mathscr{Q} = \langle G, \mathscr{P} \rangle$ over a database D under *possible* (resp. *certain*) semantics, denoted $Ans_p(\mathscr{Q}, D)$ (resp., $Ans_c(\mathscr{Q}, D)$) is the relation $\cup_M A(G, M)$ (resp., $\cap_M A(G, M)$) such that $M \in \mathscr{SM}(\mathscr{P}, D)$. Two queries $\mathscr{Q}_1 = \langle G_1, \mathscr{P}_1 \rangle$ and $\mathscr{Q}_2 = \langle G_2, \mathscr{P}_2 \rangle$ are said to be *equivalent* under semantics s ($\mathscr{Q}_1 \equiv_s \mathscr{Q}_2$) if for every database D (on a fixed schema) $Ans_s(\mathscr{Q}_1, D) = Ans_s(\mathscr{Q}_2, D)$. Moreover, we say that two programs $\mathscr{P}_1$ and $\mathscr{P}_2$ are equivalent under a given semantics s: $\mathscr{P}_1 \equiv_s \mathscr{P}_2$ if for every atom g $\langle g, \mathscr{P}_1 \rangle \equiv_s \langle g, \mathscr{P}_2 \rangle$. Finally, if $\mathscr{Q}_1 \equiv_p \mathscr{Q}_2$ and $\mathscr{Q}_1 \equiv_c \mathscr{Q}_2$ (the two queries or programs are equivalent under both brave and cautious semantics) we simply write $\mathscr{Q}_1 \equiv \mathscr{Q}_2$.

2.2 Magic-set rewriting

In the literature different approaches have been proposed for the efficient bottom-up evaluation of queries, e.g. the Magic-Set (Bancilhon *et al.* 1986), the supplementary Magic-Set (Beeri and Ramakrisnhan 1991) and other specialized rewriting techniques (Greco *et al.* 1995; Ullman 1989b; Ramakrishnan *et al.* 1993). 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

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

section we recall the *Magic-Set* rewriting is recalled, which is a general and well-known technique for the optimization of Datalog queries. Although the Magic-Set technique can be applied to general Datalog queries, for the sake of simplicity, here the technique for linear programs is presented, i.e. programs whose rules contain, at most, one body predicate mutually recursive with the head predicate.

The Magic-Set rewriting consists of three separate steps:

1. 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.
2. A *Generation step* in which the adorned program is used to generate the *magic rules* which simulate the top-down evaluation scheme.
3. 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*, $\mathscr{P}^{\beta}$ is a program whose predicate symbols have associated a string α, 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 adornment 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 α of the rule head, the adorned version of r is derived as follows:

1. identify the distinctive arguments of the rules as follows: an argument is distinctive if it is bound in the adornment α, is a constant or appears in a base predicate of the rule-body which includes an adornment argument;
2. 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 $\mathscr{Q} = \langle q(T), \mathscr{P} \rangle$ and letting α be the adornment associated with $q(T)$, the set of adorned rules for $\mathscr{Q}$ is generated by 1) first computing the adorned version of the rules defining q and 2) then generating, for each new adorned predicate p^{α} introduced in the previous step, the adorned version of the rules defining p w.r.t. α; 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:

1. eliminate all the derived predicates in the rule body which are not mutually recursive with the rule head;
2. replace the derived predicate symbol p^{α} with $magic_p^{\alpha}$ and eliminate the variables which are free w.r.t. α;
3. Replace the head predicate symbol q^{α} with $magic_q^{\alpha}$ and eliminate the variables which are free w.r.t. α;
4. interchange the transformed head and derived predicate in the body.

Finally, the modification step of an adorned rule is performed as follows: for each adorned rule whose head is $p^{\alpha}(X)$, where X is a list of variables, extend the

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

rule body with $magic_p^{\alpha}(X')$ where X' is the list of variables in X which are bound w.r.t. α.

The final program will contain only the rules which are needed to answer the query.

Example 1
Consider the query $\mathcal{Q}_3 = \langle \mathtt{p}(1,\mathtt{C}), \mathcal{P}_3 \rangle$ *where* $\mathcal{P}_3$ *is defined as follows:*

$$\begin{array}{lcl} \mathtt{p}(\mathtt{X},\mathtt{C}) & \leftarrow & \mathtt{q}(\mathtt{X},2,\mathtt{C}). \\ \mathtt{q}(\mathtt{X},\mathtt{Y},\mathtt{C}) & \leftarrow & \mathtt{a}(\mathtt{X},\mathtt{Y},\mathtt{C}). \\ \mathtt{q}(\mathtt{X},\mathtt{Y},\mathtt{C}) & \leftarrow & \mathtt{b}(\mathtt{X},\mathtt{Y},\mathtt{Z},\mathtt{W}),\ \mathtt{q}(\mathtt{Z},\mathtt{W},\mathtt{D}),\ \mathtt{c}(\mathtt{D},\mathtt{C}). \end{array}$$

The adorned version of $\mathcal{P}_3$ w.r.t. the query goal $\mathtt{p}(1,\mathtt{C})$ is:

$$\begin{array}{lcl} \mathtt{p}^{\mathtt{bf}}(\mathtt{X},\mathtt{C}) & \leftarrow & \mathtt{q}^{\mathtt{bbf}}(\mathtt{X},2,\mathtt{C}). \\ \mathtt{q}^{\mathtt{bbf}}(\mathtt{X},\mathtt{Y},\mathtt{C}) & \leftarrow & \mathtt{a}(\mathtt{X},\mathtt{Y},\mathtt{C}). \\ \mathtt{q}^{\mathtt{bbf}}(\mathtt{X},\mathtt{Y},\mathtt{C}) & \leftarrow & \mathtt{b}(\mathtt{X},\mathtt{Y},\mathtt{Z},\mathtt{W}),\ \mathtt{q}^{\mathtt{bbf}}(\mathtt{Z},\mathtt{W},\mathtt{D}),\ \mathtt{c}(\mathtt{D},\mathtt{C}). \end{array}$$

The rewritten query is $\mathcal{Q}'_3 = \langle \mathtt{p}^{\mathtt{bf}}(1,\mathtt{C}), \mathcal{P}'_3 \rangle$ where $\mathcal{P}'_3$ is as follows:

$$\begin{array}{l} \mathtt{magic_p}^{\mathtt{bf}}(1). \\ \mathtt{magic_q}^{\mathtt{bbf}}(\mathtt{X},2) \leftarrow \ \mathtt{magic_p}^{\mathtt{bf}}(\mathtt{X}). \\ \mathtt{magic_q}^{\mathtt{bbf}}(\mathtt{Z},\mathtt{W}) \leftarrow \ \mathtt{magic_q}^{\mathtt{bf}}(\mathtt{X},\mathtt{Y}),\mathtt{b}(\mathtt{X},\mathtt{Y},\mathtt{Z},\mathtt{W}). \\ \\ \mathtt{p}^{\mathtt{bf}}(\mathtt{X},\mathtt{C}) \leftarrow \mathtt{magic_p}^{\mathtt{bf}}(\mathtt{X}),\ \mathtt{q}^{\mathtt{bbf}}(\mathtt{X},2,\mathtt{C}). \\ \mathtt{q}^{\mathtt{bbf}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{magic_q}^{\mathtt{bbf}}(\mathtt{X},\mathtt{Y}),\ \mathtt{a}(\mathtt{X},\mathtt{Y},\mathtt{C}). \\ \mathtt{q}^{\mathtt{bbf}}(\mathtt{X},\mathtt{Y},\mathtt{C}) \leftarrow \mathtt{magic_q}^{\mathtt{bbf}}(\mathtt{X},\mathtt{Y}),\mathtt{b}(\mathtt{X},\mathtt{Y},\mathtt{Z},\mathtt{W}),\mathtt{q}^{\mathtt{bbf}}(\mathtt{Z},\mathtt{W},\mathtt{D}),\mathtt{c}(\mathtt{D},\mathtt{C}). \end{array}$$

Note that the first set of rules consists of the magic rules generated in the second step, while the second set of rules consists of the modified rules. □

Observe that, although the technique presented here applies only to negation free linear programs, the Magic-Set rewriting is general and can also be applied to non-linear programs with some form of negation (e.g. stratified negation) where bindings are also propagated through derived predicates (Beeri and Ramakrisnhan 1991).

Let $\mathcal{Q} = \langle G, \mathcal{P} \rangle$ be a query, then $Magic(\mathcal{Q})$ denotes the query derived from $\mathcal{Q}$ by applying the Magic-Set technique. The query $Magic(\mathcal{Q})$ will also be denoted as $\langle G^{\alpha}, magic(G, \mathcal{P}) \rangle$ where G^{α} denotes the adorned version of G, and $magic(G, \mathcal{P})$ denotes the rewriting of $\mathcal{P}$ w.r.t. the goal G. The rewritten program consists of two distinct sets of rules: a set of new rules (generated in Step (2)), called *magic rules*, and the set of *modified rules*, (generated in Step (3)) which is derived from the set of rules in the source program. The adorned rules generated in Step (1) are denoted by $Adorn(G, \mathcal{P})$.

3 Binding propagation for positive queries

In this section the technique proposed in Greco (1999, 2003) for applying the Magic-Set technique to positive disjunctive Datalog programs is reviewed and extended.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

The technique proposed in Greco (1999, 2003) produces a rewriting of the source query which, under the bottom-up evaluation, simulates the propagation of bindings occurring in the query-goal, performed in the top-down evaluation.

It should be pointed out that this formalization stems from the approach in (Greco 2003). However, it is further extended by providing a new result on query equivalence (not stated in Greco (2003)), which is crucial for allowing the technique to work in the case of disjunctive programs with constraint rules. This result is important and states that the rewriting technique is independent of the particular strategy adopted for simulating the propagation of bindings carried out in top-down evaluation. Therefore, even though conceptually introduced as an extension of the Magic-Set technique, this approach is orthogonal to the Magic-Set technique, since it can use any other rewriting strategy proposed in the literature.

For the sake of simplicity, the following running example is considered.

Example 2
Consider the query $\langle \texttt{ancestor(john, Y)}, \texttt{ANC} \rangle$ where the program ANC consists of the following rules:

```
father(X, Y) ∨ brother(X, Y)   ←  related(X, Y).
ancestor(X, Y)                 ←  father(X, Y).
ancestor(X, Y)                 ←  father(X, Z), ancestor(Z, Y).
```

The predicate ancestor defines the transitive closure of father, while father is defined by a disjunctive rule. □

Given a positive disjunctive Datalog program $\mathscr{P}$, the first step is to construct a suitable normal program.

Definition 1
Let $\mathscr{P}$ be a (positive) disjunctive Datalog program. The *extended standard* version of $\mathscr{P}$, denoted $esv(\mathscr{P})$, is the Datalog program derived from $\mathscr{P}$ by replacing each disjunctive rule $a_1 \vee \ldots \vee a_m \leftarrow B$ with

1. m rules of the form $a_i \leftarrow B$ for $1 \leqslant i \leqslant m$, and
2. $m \times (m-1)$ rules of the form $a_i \leftarrow a_j, B$ for $1 \leqslant i, j \leqslant m$ and $i \neq j$.

Given a query $\mathscr{Q} = \langle G, \mathscr{P} \rangle$, we denote with $esv(\mathscr{Q})$ the query $\langle G, esv(\mathscr{P}) \rangle$. □

Example 3
The program esv(ANC), where ANC is the program presented in Example 2, consists of the following rules:

```
father(X, Y)     ←  related(X, Y).
brother(X, Y)    ←  related(X, Y).
father(X, Y)     ←  brother(X, Y), related(X, Y).
brother(X, Y)    ←  father(X, Y), related(X, Y).
ancestor(X, Y)   ←  father(X, Y).
ancestor(X, Y)   ←  father(X, Z), ancestor(Z, Y).
```

□

Observe that the rules introduced by applying Item 2 of Definition 1 are subsumed by those introduced by applying Item 1; indeed, the semantics of $esv(\mathscr{P})$ is not

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

affected by the insertion of these rules. However, these rules are necessary in order to allow the propagation of the bindings, as will be clear in the following.

It should be pointed out that programs $\mathscr{P}$ and $esv(\mathscr{P})$ are not equivalent; in fact, $\mathscr{P}$ is a disjunctive Datalog program that is able to express all the queries in Σ_2^P, while $esv(\mathscr{P})$ is a positive Datalog program that is able to express a subset of the queries computable in polynomial time.

The second step is to derive a program that must be equivalent to the original one. Let us first present some notation.

Definition 2
Given a (positive) disjunctive Datalog program $\mathscr{P}$, $ESV(\mathscr{P})$ denotes the program derived from $esv(\mathscr{P})$ by replacing each derived predicate symbol g with a new predicate symbol G. Given a query $\mathscr{Q} = \langle g(t), \mathscr{P} \rangle$, $ESV(\mathscr{Q})$ denotes the query $\langle G(t), ESV(\mathscr{P}) \rangle$ where G is the new symbol used to replace g. □

Definition 3
Let $\mathscr{P}$ be a (positive) disjunctive Datalog program. The *restricted* version of $\mathscr{P}$, denoted by $RV(\mathscr{P})$, is the disjunctive Datalog program defined as follows:

$$RV(\mathscr{P}) = \{a_1 \vee \ldots \vee a_m \leftarrow A_1, \ldots, A_m, B \mid a_1 \vee \ldots \vee a_m \leftarrow B \in \mathscr{P}\}$$

where each A_i, for $1 \leqslant i \leqslant m$, is the atom replacing a_i in the program $ESV(\mathscr{P})$. The rewritten version of $\mathscr{P}$ is $Rew(\mathscr{P}) = RV(\mathscr{P}) \cup ESV(\mathscr{P})$. Given a query $\mathscr{Q} = \langle g(t), \mathscr{P} \rangle$, $Rew(\mathscr{Q})$ denotes the query $\langle g(t), Rew(\mathscr{P}) \rangle$. □

Observe that in the above definition, the rewriting of the program $\mathscr{P}$ does not take into account the goal. This is not true for rewriting techniques propagating bindings.

Example 4
The program Rew(ANC), where ANC is the program defined in Example 2, consists of the following rules RV(ANC):

```
father(X,Y) ∨ brother(X,Y)  ←  FATHER(X,Y), BROTHER(X,Y), related(X,Y).
ancestor(X,Y)               ←  ANCESTOR(X,Y), father(X,Y).
ancestor(X,Y)               ←  ANCESTOR(X,Y), father(X,Z), ancestor(Z,Y).
```

plus the set of rules ESV(ANC) :

```
FATHER(X,Y)     ←  related(X,Y).
BROTHER(X,Y)    ←  related(X,Y).
FATHER(X,Y)     ←  BROTHER(X,Y), related(X,Y).
BROTHER(X,Y)    ←  FATHER(X,Y), related(X,Y).
ANCESTOR(X,Y)   ←  FATHER(X,Y).
ANCESTOR(X,Y)   ←  FATHER(X,Z), ANCESTOR(Z,Y).
```

The rewritten query is ⟨ancestor(john,Y), Rew(ANC)⟩ □

The programs $\mathscr{P}$ and $Rew(\mathscr{P})$ actually have the same semantics (w.r.t. the predicates in $\mathscr{P}$).

Proposition 1
Let $\mathscr{P}$ be a (positive) disjunctive Datalog program, and $Rew(\mathscr{P})$ be the rewritten *version of $\mathscr{P}$. Then, for every atom $g(t)$, $\langle g(t), \mathscr{P} \rangle \equiv \langle g(t), Rew(\mathscr{P}) \rangle$.*

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

Proof

To prove that for any $g(t)$, $\langle g(t), \mathscr{P}\rangle \equiv \langle g(t), Rew(\mathscr{P})\rangle$, under both possible and certain semantics, it will be shown that for any database D, an interpretation M of $Rew(\mathscr{P})_D$ is a stable model if, and only if, $M[\mathscr{P}_D]$ is a stable model for $\mathscr{P}_D$. Moreover, only minimal models can be considered, since for disjunctive positive Datalog programs, the set of stable models of a given program coincides with the set of minimal models.

It should be pointed out that the program $Rew(\mathscr{P})$ consists of two distinct components: i) the program $ESV(\mathscr{P})$ whose rules only depend on D, and ii) the program $RV(\mathscr{P})$ whose rules depend on predicates defined in $ESV(\mathscr{P})$ and D. Hence, the set of stable models of $Rew(\mathscr{P})_D$ can be computed in a level wise manner.

As for the models of $ESV(\mathscr{P})_D$, observe that $ESV(\mathscr{P})_D$ has a unique minimal model, since it is a positive program. Let M_{ESV} be such a model, and let $M_{esv} = \{a \mid A \in M_{ESV}\}$.

Then, $\mathscr{MM}(ESV(\mathscr{P})_D \cup RV(\mathscr{P})_D) = \mathscr{MM}(FM_{ESV} \cup RV(\mathscr{P})_D)$, where FM_{ESV} denotes the set of facts associated with each atom in the model M_{ESV}. Finally, in order to conclude the proof, it can be claimed that $M_{ESV} \cup M$ is a minimal model for $\mathscr{MM}(FM_{ESV} \cup RV(\mathscr{P})_D)$ if, and only if, M is a minimal model for $\mathscr{P}_D$.

In fact, consider a rule $r : a_1 \vee \ldots \vee a_m \leftarrow b, b_1, \ldots, b_n$ in $\mathscr{P}$ and the corresponding rule $r' : a_1 \vee \ldots \vee a_m \leftarrow A_1, \ldots, A_m, b, b_1, .., b_n$ in $RV(\mathscr{P})$, where b is a conjunction of derived predicates, while $b_1, \ldots, b_n$ are extensional predicates. It is easy to see, that for any model M for $\mathscr{P}_D$, we have $M \subseteq M_{esv}$. Hence, if $b, b_1, \ldots, b_n$ is true in M then also $B, b_1, \ldots, b_n$ is true in M_{ESV}. Then, due to the rules $A_i \leftarrow B, b_1, .., b_n$ in $ESV(\mathscr{P})$, $A_1, \ldots, A_m$ is true in M_{ESV}, too.

Conversely, if b is false in M, then the body of r' is trivially false (no matter what the evaluation of B), too. Hence, after the assertion of the facts in M_{ESV}, the semantics of $RV(\mathscr{P})_D$ is not affected from r' the predicates $A_1, \ldots, A_m$ are removed. □

The previous proposition states that the program $\mathscr{P}$ and $Rew(\mathscr{P})$, obtained by restricting the rules of $\mathscr{P}$, are equivalent. In fact, adding the atoms defined in $ESV(\mathscr{P})$ inside the body of the rules of $\mathscr{P}$, does not make any effective restriction, as for every ground atom $a(t)$ appearing in the head of a disjunctive ground rule r there is a ground atom $A(t)$ which is derived from $ESV(\mathscr{P})$.

Thus, in the following, instead of using the program $ESV(\mathscr{P})$ to restrict the rules in $\mathscr{P}$, a different program is considered which makes an effective and sound restriction; this program will be obtained by performing the binding propagation from the query goal into the rules of the program $esv(\mathscr{P})$. However, program $ESV(\mathscr{P})$ is used instead of program $esv(\mathscr{P})$, in order to distinguish between atoms of the source program and atoms making restrictions.

Throughout the paper, the Magic-Set technique is considered which is a well-known and general technique. For special classes of queries we could use specialized techniques as the choice of the rewriting technique is independent and orthogonal w.r.t. the proposed framework.

For details about the Magic-Set technique the reader should refer to Beeri and Ramakrishnan (1991) andf Ullman (1989b), while for specialized rewriting

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

techniques see Ullman (1989b), Ramakrishnan *et al.* (1993) and Greco *et al.* (1995).

Let us now formally provide a way of "collecting" all the adorned predicates generated by the rewriting of queries.

Definition 4
Given a (positive) disjunctive Datalog program $\mathscr{P}$, a predicate symbol p appearing in $\mathscr{P}$ and an adorned program $\mathscr{P}^{\beta}$ derived from $\mathscr{P}$, then

$$Coll(p, \mathscr{P}, \mathscr{P}^{\beta}) = \{p(X_1, .., X_k) \leftarrow p^{\alpha}(X_1, .., X_k) | \text{for every } p^{\alpha} \text{ in } \mathscr{P}^{\beta} \text{ derived from } p\}$$

denotes the set of rules (also called *collecting rules*) used to collect the atoms of the predicates having different adornments, but the same predicate symbols. Moreover,

$$Coll(\mathscr{P}, \mathscr{P}^{\beta}) = \bigcup_{p \text{ appearing in } \mathscr{P}} Coll(p, \mathscr{P}, \mathscr{P}^{\beta})$$

denotes the set of all collecting rules derived from $\mathscr{P}$ and $\mathscr{P}^{\beta}$. □

With respect to the rewriting of Definition 3, $ESV(\mathscr{P})$ is now replaced by the rules of the Magic-Set program $\mathscr{P}' = Magic(G(t), ESV(\mathscr{P}))$ (where G is the symbol corresponding to g in $ESV(\mathscr{P})$) plus the rules used to collect the atoms with different adornments of $\mathscr{P}'$, denoted by $Coll(ESV(\mathscr{P}), \mathscr{P}')$.

Example 5
Consider the program ANC of Example 4 and the query goal ancestor(john, Y). The corresponding extended standard query is $ESV(\langle \texttt{ancestor(john, Y)}, \texttt{ANC} \rangle) = \langle \texttt{ANCESTOR(john, Y)}, ESV(\texttt{ANC}) \rangle$. The first step consists in the generation of adornments for derived predicates. From the propagation of bindings only one binding for the predicate $\texttt{ANCESTOR}^{\texttt{b}}\texttt{(john, Y)}$ is derived.

Thus, the program $Magic(\texttt{ANCESTOR(john, Y)}, \texttt{ESV(ANC)})$ is as follows:

$\texttt{FATHER}^{\texttt{bf}}\texttt{(X, Y)} \leftarrow \texttt{magic_FATHER}^{\texttt{bf}}\texttt{(X), related(X, Y).}$
$\texttt{BROTHER}^{\texttt{bf}}\texttt{(X, Y)} \leftarrow \texttt{magic_BROTHER}^{\texttt{bf}}\texttt{(X), related(X, Y).}$
$\texttt{FATHER}^{\texttt{bf}}\texttt{(X, Y)} \leftarrow \texttt{magic_FATHER}^{\texttt{bf}}\texttt{(X), BROTHER}^{\texttt{bf}}\texttt{(X, Y), related(X, Y).}$
$\texttt{BROTHER}^{\texttt{bf}}\texttt{(X, Y)} \leftarrow \texttt{magic_BROTHER}^{\texttt{bf}}\texttt{(X), FATHER}^{\texttt{bf}}\texttt{(X, Y), related(X, Y).}$
$\texttt{ANCESTOR}^{\texttt{bf}}\texttt{(X, Y)} \leftarrow \texttt{magic_ANCESTOR}^{\texttt{bf}}\texttt{(X), FATHER}^{\texttt{bf}}\texttt{(X, Y).}$
$\texttt{ANCESTOR}^{\texttt{bf}}\texttt{(X, Y)} \leftarrow \texttt{magic_ANCESTOR}^{\texttt{bf}}\texttt{(X), FATHER}^{\texttt{bf}}\texttt{(X, Z), ANCESTOR}^{\texttt{bf}}\texttt{(Z, Y).}$

$\texttt{magic_ANCESTOR}^{\texttt{bf}}\texttt{(john).}$
$\texttt{magic_ANCESTOR}^{\texttt{bf}}\texttt{(Z)} \leftarrow \texttt{magic_ANCESTOR}^{\texttt{bf}}\texttt{(X), FATHER}^{\texttt{bf}}\texttt{(X, Z).}$
$\texttt{magic_FATHER}^{\texttt{bf}}\texttt{(X)} \leftarrow \texttt{magic_ANCESTOR}^{\texttt{bf}}\texttt{(X).}$
$\texttt{magic_BROTHER}^{\texttt{bf}}\texttt{(X)} \leftarrow \texttt{magic_FATHER}^{\texttt{bf}}\texttt{(X).}$
$\texttt{magic_FATHER}^{\texttt{bf}}\texttt{(X)} \leftarrow \texttt{magic_BROTHER}^{\texttt{bf}}\texttt{(X).}$

Here the predicate $\texttt{magic_ANCESTOR}^{\texttt{bf}}$ computes all ancestors which are relevant to establish whether a given person is an ancestor of john.
The set $Coll(ESV(\texttt{ANC}), Magic(\texttt{ANCESTOR(john, Y)}, ESV(\texttt{ANC})))$, consisting of the rules collecting atoms with the same predicate and different adornments, is

$\texttt{ANCESTOR(X, Y)} \leftarrow \texttt{ANCESTOR}^{\texttt{bf}}\texttt{(X, Y).}$
$\texttt{FATHER(X, Y)} \leftarrow \texttt{FATHER}^{\texttt{bf}}\texttt{(X, Y).}$
$\texttt{BROTHER(X, Y)} \leftarrow \texttt{BROTHER}^{\texttt{bf}}\texttt{(X, Y).}$

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

These rules collect into ANCESTOR (resp. FATHER, BROTHER) all the ANCESTOR (resp. FATHER, BROTHER) atoms with different adornments. Since there is only one adornment for each predicate, adornments and collecting rules could be eliminated. □

Definition 5
Let $\mathcal{Q} = \langle g(t), \mathcal{P} \rangle$ be a disjunctive Datalog query, then the disjunctive Magic-Set rewriting of $\mathcal{P}$ w.r.t. $g(t)$, denoted by $Disj_Magic(g(t), \mathcal{P})$, is the following program: $RV(\mathcal{P}) \cup Coll(ESV(\mathcal{P}), Magic(G(t), ESV(\mathcal{P}))) \cup Magic(G(t), ESV(\mathcal{P}))$. □

Example 6
The complete rewriting of the program in Example 2 consists of the rules in $Coll(ESV(\texttt{ANC}), Magic(\texttt{ANCESTOR(john, Y)}, ESV(\texttt{ANC}))) \cup Magic(\texttt{ANCESTOR(john, Y)}, ESV(\texttt{ANC}))$ (shown in Example 5) plus the rules in $RV(\texttt{ANC})$:

$$\texttt{father(X, Y)} \vee \texttt{brother(X, Y)} \leftarrow \texttt{FATHER(X, Y), BROTHER(X, Y), related(X, Y).}$$

$$\texttt{ancestor(X, Y)} \leftarrow \texttt{ANCESTOR(X, Y), father(X, Y).}$$
$$\texttt{ancestor(X, Y)} \leftarrow \texttt{ANCESTOR(X, Y), father(X, Z), ancestor(Z, Y).}$$

□

From the above observation and definition, combined with Proposition 1 the following result proved in Greco (2003) can be derived.

Fact 1
Let $\mathcal{Q} = \langle g(t), \mathcal{P} \rangle$ *be a Datalog*$^{\vee}$ *query, then,* $\mathcal{Q} \equiv \langle g(t), Disj_Magic(g(t), \mathcal{P}) \rangle$. □

4 Binding propagation in Datalog$^{\vee}$ programs with constraints

This section formally introduces a technique for propagating bindings into Datalog$^{\vee}$ queries with *strong* or *classical* constraints, a simple and powerful form of unstratified negation. A strong constraint is a rule with empty head of the form: $\leftarrow B(X)$ where $B(X)$ is a conjunction of literals and X is a vector of range restricted variables, which must be satisfied in each model[1].

Contrary to standard Datalog, where bindings are propagated from the head of rules into the body, the problem with programs containing constraints is that the bindings need to be propagated also through the constraints. For instance, if one is interested in knowing whether $p(1)$ is true in a program where there is the constraint $\leftarrow p(X), q(X)$, then the truth value of $q(1)$ also needs to be evaluated.

Thus, the truth value of each ground atom in a constraint depends on the truth value of the other atoms appearing in the same ground constraint, and, hence, in a more abstract perspective, constraints behave in a similar manner as disjunctive rules when propagating bindings into their heads.

In the following, a Datalog$^{\vee}$ program $\mathcal{P}$ with constraints (Datalog$^{\vee,\leftarrow}$) will be denoted by a pair $(\mathcal{P}_R, \mathcal{P}_C)$, where $\mathcal{P}_R$ is a nonempty set of (positive) disjunctive rules and $\mathcal{P}_C$ is a set of constraints. It is worth noting that $\mathcal{P}_R$ being a nonempty

[1] It should be recalled that, under stable models semantics, a constraint $\leftarrow B(X)$ could be rewritten into an equivalent rule with unstratified negation of the form: $p_i \leftarrow B(X), \neg p_i$, where p_i is a new predicate symbol not defined elsewhere in the program.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

set of positive disjunctive rules, the only form of negation contained in $\mathscr{P}$ is that related to the rewriting of constraints. Moreover, the use of constraints instead of general (possibly unstratified) negation is not a limitation since Datalog$^{\vee,\leftarrow}$ has the same expressive power as Datalog$^{\vee,\neg}$. Indeed, since in Eiter *et al.* (1997) it has been shown that Datalog$^{\vee,\neg_s}$ has the same expressivity as Datalog$^{\vee,\neg}$, whereas in Zumpano *et al.* (2004) it has been shown that stratified negation can be simulated by using disjunction and constraints, we have that Datalog$^{\vee,\leftarrow} \equiv$ Datalog$^{\vee,\neg_s}$.

As has been shown (Zumpano *et al.* 2004), that every rule containing stratified negation can be rewritten in Datalog$^{\vee,\leftarrow}$ rules. For instance, consider the following stratified rule where predicate b does not depend on predicate p:

$$p(X) \leftarrow a(X), \neg b(X)$$

This rule can be rewritten as:

$$\begin{aligned}
&1: \ p(X) \leftarrow p'(X)\\
&2: \ p'(X) \vee b'(X) \leftarrow a(X)\\
&3: \leftarrow p'(X), b'(X)\\
&4: \leftarrow b'(X) \neg b(X)\\
&5: \leftarrow p'(X), b(X)
\end{aligned}$$

where the rule 2 together with the constraint 3 states that a is partitioned into p' and b', whereas constraints 4 and 5 state, respectively, that b' must be a subset of b and that the intersection between p' and b must be empty. Therefore, tuples being in $a - b$ must be in p'. The rule 1 is necessary only in the case the predicate p in the source program is defined by more than one rule. In the following we assume that a predicate symbol p cannot appear both positively and negatively in two different constraints. This restriction does not make any limitation on the expressive power of the language as stratified negation can be emulated by considering the above restricted form of constraints.

By following the same guidelines as in the previous section, the technique and the main results, by using a running example.

Example 7

Suppose to have the query $\langle \texttt{2col(1,2)}, \texttt{Coloring} \rangle$ checking whether a graph is 3-colorable and whether the colors *red* and *blue* can be assigned to nodes 1 and 2, respectively. The program `Coloring` consists of the following rules:

```
r : 2col(X,Y)  ←  color(X,red), color(Y,blue).
s : color(X,red) ∨ color(X,blue) ∨ color(X,yellow)  ←  node(X).
c :  ←  edge(X,Y), color(X,C), color(Y,C).
```

□

Definition 6

Given a set of constraints P_C, $esv(P_C)$ denotes the set of Datalog rules obtained by replacing each constraint in P_C having the form

$$\leftarrow a_1, \ldots, a_k, b_1, .., b_m, \neg c_1, \ldots, \neg c_n$$

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

where $a_1, \ldots, a_k$ are base atoms, $b_1, \ldots, b_m$ are derived atoms and $c_1, \ldots, c_n$ are negated literals (either base or derived), with the following set of rules:

$$b_i \leftarrow b_1, .., b_{i-1}, b_{i+1}, .., b_m, a_1, \ldots, a_k, \neg c_1, \ldots, \neg c_n \quad \forall i \in [1..m]$$

Given program $\mathscr{P} = (\mathscr{P}_R, \mathscr{P}_C)$, the *extended standard version* of $\mathscr{P}$, denoted by $esv(\mathscr{P}) = esv((\mathscr{P}_R, \mathscr{P}_C)) = esv(\mathscr{P}_R) \cup esv(\mathscr{P}_C)$. □

Example 8
The extended standard version of the program `Coloring` of Example 7, obtained by the rewriting of constraints and disjunctive rules and denoted by *esv*(`Coloring`), is as follows:

```
r1 : 2col(X,Y)   ←  color(X,red), color(Y,blue).

s1 : color(X,red)      ←  node(X).
s2 : color(X,blue)     ←  node(X).
s3 : color(X,yellow)   ←  node(X).

s4 : color(X,red)      ←  color(X,blue), node(X).
s5 : color(X,blue)     ←  color(X,red), node(X).
s6 : color(X,red)      ←  color(X,yellow), node(X).
s7 : color(X,yellow)   ←  color(X,red), node(X).
s8 : color(X,blue)     ←  color(X,yellow), node(X).
s9 : color(X,yellow)   ←  color(X,blue), node(X).

c1 : color(X,C)  ←  edge(X,Y), color(Y,C).
c2 : color(Y,C)  ←  edge(X,Y), color(X,C).
```

where the rule $\mathtt{r}_1$ is derived from `r`, rules $\mathtt{s}_1 - \mathtt{s}_9$ are derived from the rule `s` and rules $\mathtt{c}_1$ and $\mathtt{c}_2$ are derived from constraint `c`. □

As in the case of disjunctive programs without constraints, $ESV(\mathscr{P})$ denotes the program derived from $esv(\mathscr{P})$ by replacing each derived predicate symbol g with a new predicate symbol G.

Definition 7
Let $\mathscr{P} = (\mathscr{P}_R, \mathscr{P}_C)$ be a Datalog$^\vee$ program with constraints. Program $Rew(\mathscr{P})$ is defined as $RV(\mathscr{P}_R) \cup \mathscr{P}_C \cup ESV(\mathscr{P})$. Given a query $\mathscr{Q} = \langle g(t), \mathscr{P} \rangle$, $Rew(\mathscr{Q})$ denotes the query $\langle g(t), Rew(\mathscr{P}) \rangle$. □

Notice that, with respect to programs without constraints, $RV(\mathscr{P}_R)$ is replaced by $RV(\mathscr{P}_R) \cup \mathscr{P}_C$ and $ESV(\mathscr{P}_R)$ by $ESV(\mathscr{P}_R) \cup ESV(\mathscr{P}_C)$.

Example 9
The set of restricted rules in RV(`Coloring`), derived from the program `Coloring` of Example 7, is as follows:

```
2col(X,Y)  ←  2COL(X,Y), color(X,red), color(Y,blue).
color(X,red) ∨ color(X,blue) ∨ color(X,yellow)  ←  COLOR(X,red), COLOR(X,blue),
                                                   COLOR(X,yellow), node(X).
```

where predicates `2COL` and `COLOR` are defined in ESV(`Coloring`) which is derived from program *esv*(`Coloring`), shown in Example 8, by replacing `2col` with `2COL`

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

and color with COLOR. The complete rewriting of the program Coloring consists of the above rules plus the constraint:

$$\leftarrow \quad \texttt{edge(X,Y)},\ \texttt{color(X,C)},\ \texttt{color(Y,C)}.$$

and the rules in $ESV(\texttt{Coloring})$, presented in Example 8. □

The first interesting result is that (as for Datalog$^{\vee}$ programs without constraints) the above rewriting method does not affect the semantics of the query.

Proposition 2
Let $\mathscr{P} = (\mathscr{P}_R, \mathscr{P}_C)$ be a Datalog$^{\vee}$ program with constraints, and $Rew(\mathscr{P})$ be the rewritten version of $\mathscr{P}$. Then, for every atom $g(t)$, $\langle g(t), \mathscr{P}\rangle \equiv \langle g(t), Rew(\mathscr{P})\rangle$.

Proof. Recall that $Rew(\mathscr{P}) = Rew((\mathscr{P}_R, \mathscr{P}_C)) = RV(\mathscr{P}_R) \cup \mathscr{P}_C \cup ESV(\mathscr{P})$; hence $Rew(\mathscr{P}) = Rew(\mathscr{P}_R) \cup \mathscr{P}_C \cup ESV(\mathscr{P}_C)$ can also be written.

Let us now consider program $\mathscr{P}' = Rew(\mathscr{P}_R) \cup ESV(\mathscr{P}_C)$; using the same arguments as for Proposition 1, $\langle g(t), \mathscr{P}_R\rangle \equiv \langle g(t), \mathscr{P}'\rangle$ is derived. In fact, it has already been shown that adding the atoms defined in $ESV(\mathscr{P}_R)$ inside the body of the rules of $\mathscr{P}$, does not make any effective restriction, provided that $\mathscr{P}_R$ is a positive program; moreover, program $ESV(\mathscr{P}_C)$ is also a positive program that possibly enlarges the unique model M_{ESV} of $ESV(\mathscr{P})$, without affecting the models of $RV(\mathscr{P}_R)$. Finally, the result follows by observing that the set of constraints $\mathscr{P}_C$ affects the result of the query only in the case when there is some ground constraint that is not satisfied, and by the fact that $\mathscr{P}$ and $Rew(\mathscr{P})$ share the same set of constraints. □

Thus, a viable way for reducing the number of models to be computed is to consider a suitable rewriting of $ESV(\mathscr{P})$, which is able to make an effective and sound restriction by simulating the binding propagation occurring in top-down evaluation. This rewriting is carried out by means of the Magic-Set technique, which in fact limits attention to the models that are really needed for answering the query.

Definition 8
Let $\mathscr{Q} = \langle g(t), \mathscr{P}\rangle$ with $\mathscr{P} = (\mathscr{P}_R, \mathscr{P}_C)$, then the disjunctive Magic-Set rewriting of $\mathscr{P}$ w.r.t. $g(t)$, denoted by $Disj_Magic(g(t), \mathscr{P})$, is defined as follows:
$RV(\mathscr{P}_R) \cup \mathscr{P}_C \cup Coll(ESV(\mathscr{P}), Magic(G(t), ESV(\mathscr{P}))) \cup Magic(G(t), ESV(\mathscr{P}))$. □

Example 10
Consider again the query $\langle \texttt{2col(1,2)}, \texttt{Coloring}\rangle$ of Example 7.
The program $Magic(\texttt{2COL(1,2)}, ESV(\texttt{Coloring}))$, obtained by applying the Magic-Set technique to the query $ESV(\langle \texttt{2col(1,2)}, \texttt{Coloring}\rangle)$, is as follows:

$$\begin{aligned}
&\texttt{magic_2COL}^{\texttt{bb}}\texttt{(1,2)}.\\
&\texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,red)} \leftarrow \texttt{magic_2COL}^{\texttt{bb}}\texttt{(X,Y)}.\\
&\texttt{magic_COLOR}^{\texttt{bb}}\texttt{(Y,blue)} \leftarrow \texttt{magic_2COL}^{\texttt{bb}}\texttt{(X,Y)}.\\
&\texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,blue)} \leftarrow \texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,red)}.\\
&\texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,red)} \leftarrow \texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,blue)}.\\
&\texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,yellow)} \leftarrow \texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,red)}.\\
&\texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,red)} \leftarrow \texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,yellow)}.\\
&\texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,blue)} \leftarrow \texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,yellow)}.\\
&\texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,yellow)} \leftarrow \texttt{magic_COLOR}^{\texttt{bb}}\texttt{(X,blue)}.
\end{aligned}$$

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

$\texttt{magic_COLOR}^{bb}(\texttt{Y},\texttt{C}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{C}), \texttt{edge}(\texttt{X},\texttt{Y}).$
$\texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{C}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{Y},\texttt{C}), \texttt{edge}(\texttt{X},\texttt{Y}).$

$\texttt{COLOR}^{bb}(\texttt{X},\texttt{red}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{red}), \texttt{node}(\texttt{X}).$
$\texttt{COLOR}^{bb}(\texttt{X},\texttt{blue}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{blue}), \texttt{node}(\texttt{X}).$
$\texttt{COLOR}^{bb}(\texttt{X},\texttt{yellow}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{yellow}), \texttt{node}(\texttt{X}).$

$\texttt{COLOR}^{bb}(\texttt{X},\texttt{red}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{red}), \texttt{COLOR}^{bb}(\texttt{X},\texttt{blue}).$
$\texttt{COLOR}^{bb}(\texttt{X},\texttt{blue}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{blue}), \texttt{COLOR}^{bb}(\texttt{X},\texttt{red}).$
$\texttt{COLOR}^{bb}(\texttt{X},\texttt{red}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{red}), \texttt{COLOR}^{bb}(\texttt{X},\texttt{yellow}).$
$\texttt{COLOR}^{bb}(\texttt{X},\texttt{yellow}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{yellow}), \texttt{COLOR}^{bb}(\texttt{X},\texttt{red}).$
$\texttt{COLOR}^{bb}(\texttt{X},\texttt{blue}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{blue}), \texttt{COLOR}^{bb}(\texttt{X},\texttt{yellow}).$
$\texttt{COLOR}^{bb}(\texttt{X},\texttt{yellow}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{yellow}), \texttt{COLOR}^{bb}(\texttt{X},\texttt{blue}).$

$\texttt{COLOR}^{bb}(\texttt{X},\texttt{C}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{X},\texttt{C}), \texttt{edge}(\texttt{X},\texttt{Y}), \texttt{COLOR}^{bb}(\texttt{Y},\texttt{C}).$
$\texttt{COLOR}^{bb}(\texttt{Y},\texttt{C}) \leftarrow \texttt{magic_COLOR}^{bb}(\texttt{Y},\texttt{C}), \texttt{edge}(\texttt{X},\texttt{Y}), \texttt{COLOR}^{bb}(\texttt{X},\texttt{C}).$

$\texttt{2COL}^{bb}(\texttt{X},\texttt{Y}) \leftarrow \texttt{magic_2COL}^{bb}(\texttt{X},\texttt{Y}), \texttt{COLOR}^{bb}(\texttt{X},\texttt{red}), \texttt{COLOR}^{bb}(\texttt{Y},\texttt{blue}).$

while the rules in $Coll(ESV(\texttt{Coloring}), Magic(\texttt{2COL}(1,2), ESV(\texttt{Coloring})))$ are:

$\texttt{2COL}(\texttt{X},\texttt{Y}) \leftarrow \texttt{2COL}^{bb}(\texttt{X},\texttt{Y}).$
$\texttt{COLOR}(\texttt{X},\texttt{red}) \leftarrow \texttt{COLOR}^{bb}(\texttt{X},\texttt{red}).$
$\texttt{COLOR}(\texttt{X},\texttt{blue}) \leftarrow \texttt{COLOR}^{bb}(\texttt{X},\texttt{blue}).$
$\texttt{COLOR}(\texttt{X},\texttt{yellow}) \leftarrow \texttt{COLOR}^{bb}(\texttt{X},\texttt{yellow}).$ □

Before formally presenting the correctness of the rewriting technique, let us resume the process and make some comments. The rewriting process takes in input a query $Q = \langle g(t), \mathscr{P} \rangle = \langle g(t), (\mathscr{P}_R, \mathscr{P}_C) \rangle$ and first generates the equivalent query $Q' = \langle g(t), (RV(\mathscr{P}_R) \cup ESV(\mathscr{P}), \mathscr{P}_C) \rangle$, where $ESV(\mathscr{P})$ is a normal program. Next the query $Q'' = \langle g(t), (RV(\mathscr{P}_R) \cup \mathscr{P}', \mathscr{P}_C) \rangle$, where $\mathscr{P}'$ is the optimized program derived from $ESV(\mathscr{P})$ is produced. To answer the source query Q is sufficient to consider the minimal models of $\mathscr{P}_R$ which satisfies $\mathscr{P}_C$. As the program $ESV(\mathscr{P})$ (resp. $\mathscr{P}'$) may contain negated literals, to answer the rewritten query Q' (resp. Q'') the stable models of $RV(\mathscr{P}_R) \cup ESV(\mathscr{P})$ (resp. $RV(\mathscr{P}_R) \cup \mathscr{P}'$) satisfying $\mathscr{P}_C$ have to be computed. Moreover, under the assumption that a predicate symbol p cannot appear both positively and negatively in two different constraints, the program $ESV(\mathscr{P})$ is stratified and therefore has a unique stable model (namely the perfect or stratified model). Therefore, in order to answer the query Q' it is sufficient to compute the perfect model M of $ESV(\mathscr{P})$ and then to compute the minimal models of $RV(\mathscr{P}_R) \cup M$ satisfying $\mathscr{P}_C$. For unstratified $ESV(\mathscr{P})$ the complete set of stable models has to be considered. However, as already mentioned, problems in the second level of the polynomial hierarchy can be expressed by means of Datalog$^{\vee}$ programs with restricted constraints.

4.1 Query equivalence results

For the sake of presentation in Figure 1 the main steps provided by the whole algorithm presented in this section are explicitly pointed out. The algorithm takes as input a query $\langle g(t), \mathscr{P} \rangle$ and a database D, and outputs the set of stable models of

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

Input: A query $Q = \langle g(t), \mathscr{P} \rangle$ with $\mathscr{P} = (\mathscr{P}_R, \mathscr{P}_C)$, a database D;
Output: The stable models of $(Disj_Magic(g(t), \mathscr{P}))_D$.
var: ESV, RV, $Coll$: **set of rules**;
begin
  let $ESV := \emptyset$, $RV := \emptyset$, $Coll := \emptyset$;
  //*** *generation of the restricted version* ***
  for each rule $r \in \mathscr{P}_R$ of the form $a_1(t) \vee \ldots \vee a_n(t) \leftarrow B$ **do**
    insert $a_1 \vee \ldots \vee a_n \leftarrow A_1, \ldots, A_n, B$ **into** RV;
  //*** *generation of the extended standard version* ***
  for each rule $r \in \mathscr{P}_R$ of the form $a_1(t) \vee \ldots \vee a_n(t) \leftarrow B$ **do begin**
    insert $a_i \leftarrow B$ for $1 \leqslant i \leqslant n$ **into** ESV;
    insert $a_i \leftarrow a_j, B$ for $1 \leqslant i, j \leqslant n$ and $i \neq j$ **into** ESV;
  end
  for each constraint $c \in \mathscr{P}_C$ of the form $\leftarrow a_1, \ldots, a_k, b_1, .., b_m, \neg c_1, \ldots, \neg c_n$ **do**
    insert $b_i \leftarrow b_1, .., b_{i-1}, b_{i+1}, .., b_k, a_1, \ldots, a_k, \neg c_1, \ldots, \neg c_n$ for$1 \leqslant i \leqslant m$ **into** ESV;
  //*** *application of the Magic-Set technique to a normal program* ***
  let $Magic := Magic(G(t), ESV(\mathscr{P}))$;
  //*** *generation of the collecting rules* ***
  for each predicate symbol p defined in $\mathscr{P}$ with arity k **do**
    for each adornment α of p in $Magic$ **do**
      insert $p(X_1, .., X_k) \leftarrow p^{\alpha}(X_1, .., X_k)$ **into** $Coll$;
  return $\mathscr{SM}(RV \cup \mathscr{P}_C \cup Coll \cup Magic \cup D)$;
end.

Fig. 1. Algorithm Extended Magic-Set (*Magic_Partial*).

$Disj_Magic(g(t), \mathscr{P})_D$; obviously, this set can be used for answering the query under both the possible and certain semantics.

For Datalog$^{\vee}$ programs without constraints the correctness of Algorithm in Figure 1 follows from Fact 1. For Datalog$^{\vee}$ program with constraints, its correctness (under proper assumptions) will be provided below.

First of all observe that as in the case of disjunctive queries without constraints, the application of the Magic-Set technique to queries with constraints produces a query that can be evaluated more efficiently. Unfortunately, for Datalog$^{\vee,\leftarrow}$ queries the technique previously described produces a query that, generally, is not equivalent to the original one. This result is due to the fact that Magic-Set technique not only focuses on the models really relevant for answering the query, but also computes part of the models and not models in their entirety. In contrast, constraints express conditions that must hold for every ground instance of the program including atoms which are not relevant to the query. This observation will be clearer after the following example.

Example 11
Consider the query $\langle$`2col(1,2)`, `Coloring`$\rangle$ applied to the program of Example 8 on the graph shown in Figure 2 consisting of two disconnected components, say C_1 and C_2. Since both nodes 1 and 2 belong to component C_1, there is no way for propagating bindings from 1 and 2 into the component C_2. This means that the query goal only depends on component C_1 and the colorability of C_2 affects the result only in the case when the component C_2 is not colorable. □

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

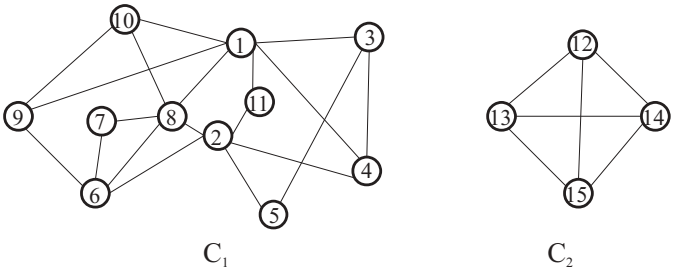


Fig. 2. A graph with two components, with C_2 being not 3-Colorable.

The above example suggests that the original query $\mathcal{Q} = \langle g(t), \mathcal{P} \rangle$ and the query $\mathcal{Q}'$, obtained by applying the Magic-Set technique to $\mathcal{Q}$, are equivalent if the program $\mathcal{P}_D$ admits stable models. Before formally proving such an intuition, some preliminary definitions are provided.

Given a Datalog$^\vee$ program with constraints $\mathcal{P}$ and an interpretation N for $\mathcal{P}$, then $\mathcal{P}/N$ denotes the set of rules in $ground(\mathcal{P})$ which are true w.r.t. N, and $\overline{\mathcal{P}/N}$ denotes the set of rules in $ground(\mathcal{P})$ which are false w.r.t. N, i.e. $\overline{\mathcal{P}/N} = ground(\mathcal{P}) - \mathcal{P}/N$.

To capture the meaning of the application of the Magic-Set technique in the case of disjunctive rules with constraints, use is made of the following concept.

Definition 9

Let $\mathcal{P}$ be a Datalog$^\vee$ program with constraints and D a database and let M be an interpretation for $\mathcal{P}_D$, then M is a *pre-model* for $\mathcal{P}_D$ if there exists $N \in \mathcal{SM}(\mathcal{P}_D)$ such that $M \subseteq N$. □

In the case of Datalog$^\vee$ programs without constraints, every model of the program rewritten by means of the Magic-Set technique can be extended to be a model of the original program, i.e. given a program $\mathcal{P}_D$, every model of $Disj_Magic(g(t), \mathcal{P}_D)$ restricted to the predicates in $\mathcal{P}_D$ is a pre-model for $\mathcal{P}_D$.

Lemma 2

Let $\langle g(t), \mathcal{P}_R \rangle$ be a disjunctive Datalog query without constraints. Then, for any database D, $M \in \mathcal{SM}(Disj_Magic(g(t), (\mathcal{P}_R)_D))$ if, and only if, there exists $N \in \mathcal{SM}(P_D)$ such that $M[(\mathcal{P}_R)_D] \subseteq N$.

Proof

From Fact 3 it is known that $\langle g(t), \mathcal{P}_R \rangle \equiv \langle g(t), Disj_Magic(g(t), \mathcal{P}_R) \rangle$. Indeed, for any model $M \in \mathcal{SM}(Disj_Magic(g(t), (\mathcal{P}_R)_D))$ the program $\overline{(\mathcal{P}_R)_D/M[(\mathcal{P}_R)_D]}$ consists of a set of ground rules that are false in $(\mathcal{P}_R)_D$ only because they are not necessary for answering the query, i.e. they are not used for propagating the binding. Thus, there exists a way for extending model $M[(\mathcal{P}_R)_D]$ into a new model N for $(\mathcal{P}_R)_D$ that satisfies the above rules. □

The above lemma states that from the rewritten program atoms not inferable from the source program cannot be inferred, apart from those introduced for binding propagation and to collect atoms with different adornments.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

In the case of a program $\mathscr{P}$ with constraint rules, the above observation does not hold as it is not always the case that for a given program $\mathscr{P}_D$ every model of $Disj_Magic(g(t), \mathscr{P}_D)$ restricted to the predicates in $\mathscr{P}_D$ is a pre-model for $\mathscr{P}_D$.

Lemma 3
There exists a query $\langle g(t), \mathscr{P}\rangle$, with $\mathscr{P} = (\mathscr{P}_R, \mathscr{P}_C)$ and $\mathscr{P}_C \neq \emptyset$, a database D and a model M in $\mathscr{SM}(Disj_Magic(g(t), \mathscr{P}_D))$ such that $M[\mathscr{P}_D]$ is not a pre-model for $\mathscr{P}_D$.

Proof. Consider the query $\langle \texttt{2col}(1,2), \texttt{Coloring}\rangle$ presented in Example 7. Consider the database D modelling the graph in Example 11.

Then, $\mathscr{P}_D$ does not have stable models, since the graph is not 3-colorable. However, $\mathscr{SM}(Disj_Magic(2col(1,2), \mathscr{P}_D)) \neq \emptyset$, since the component of the graph containing both nodes 1 and 2 is 3-colorable – any such legal coloring corresponds in fact to a stable model. □

Hence, the Magic-Set technique (and any other similar technique to propagate bindings) for Datalog$^\vee$ programs with constraints does not produce a query equivalent to the original one. Nonetheless, it is natural to investigate some restrictions that may guarantee the soundness and/or the completeness of the answers.

Theorem 1
Let $\mathscr{Q} = \langle g(t), \mathscr{P}\rangle$ be a query, where $\mathscr{P} = (\mathscr{P}_R, \mathscr{P}_C)$ is Datalog$^\vee$ program with constraints, and D a database. Then, for each model M' of $\mathscr{P}_D$, there exists $M \in \mathscr{SM}(Disj_Magic(g(t), \mathscr{P}_D))$, with $M[\mathscr{P}_D] \subseteq M'$ being a pre-model for $\mathscr{P}_D$.

Proof. Recall that $Disj_Magic(g(t), (\mathscr{P}_R, \mathscr{P}_C))$ is defined as $RV(\mathscr{P}_R) \cup \mathscr{P}_C \cup Coll(ESV(\mathscr{P}),\ Magic(G(t), ESV(\mathscr{P}))) \cup Magic(G(t), ESV(\mathscr{P}))$. Given two programs S_1 and S_2 such that $S_2 = RV(S_1)$, then we also denote S_1 as $RV^{-1}(S_2)$.

Let $\mathscr{P}' = Disj_Magic(g(t), \mathscr{P})) - \mathscr{P}_C$. Then, $\langle g(t), \mathscr{P}'\rangle \equiv \langle g(t), \mathscr{P}_R\rangle \equiv \langle g(t), Disj_Magic(g(t), \mathscr{P}_R)\rangle$ as the additional rules in $\mathscr{P}'$ do not make any restrictions on the ground rules in $\mathscr{P}_R$ used to derive the goal $g(t)$. The program $\mathscr{P}_1 = ground(\mathscr{P}')$ consists of the three distinct sets: $\mathscr{P}_{11} = ground(RV(\mathscr{P}_R))$ containing restricted rules, $\mathscr{P}_{12} = ground(Coll(ESV(\mathscr{P}),\ Magic(G(t), ESV(\mathscr{P}))))$ containing collecting rules and $\mathscr{P}_{13} = ground(Magic(G(t), ESV(\mathscr{P})))$ containing adorned rules.

Consider now program $\mathscr{P}_2 = ground(\mathscr{P}_R) - RV^{-1}(\mathscr{P}_{11})$ containing all the rules in $ground(\mathscr{P}_R)$ not relevant for the query goal $g(t)$. It is obvious that, letting $\mathscr{P}_3 = ground(\mathscr{P}_R) - \mathscr{P}_2$, we have that $\langle g(t), \mathscr{P}_3\rangle \equiv \langle g(t), \mathscr{P}_1\rangle$ as both contain all the ground rules relevant for the query goal.

Therefore, for any database D, $\mathscr{SM}(\mathscr{P}_D) = \mathscr{SM}(ground(\mathscr{P})_D) = \mathscr{SM}((\mathscr{P}_3 \cup \mathscr{P}_C)_D) \times \mathscr{SM}((\mathscr{P}_2 \cup \mathscr{P}_C)_D)$, where for any two sets of stable models A and B, $A \times B = \{M_1 \cup M_2 \mid M_1 \in A \wedge M_2 \in B\}$.

Moreover, since $\langle g(t), \mathscr{P}_3 \cup \mathscr{P}_C\rangle \equiv \langle g(t), \mathscr{P}_1 \cup \mathscr{P}_C\rangle$, as the constraints act on atoms derived from both $\mathscr{P}_3$ and $\mathscr{P}_1$, and $\langle g(t), \mathscr{P}_1 \cup \mathscr{P}_C\rangle \equiv \langle g(t), \mathscr{P}' \cup \mathscr{P}_C\rangle$, we conclude that $\langle g(t), \mathscr{P}_3 \cup \mathscr{P}_C\rangle = \langle g(t), Disj_Magic(g(t), \mathscr{P}_D)\rangle$.

Hence, given a model M' of $\mathscr{P}_D$, there exist (i) $M \in \mathscr{SM}(Disj_Magic(g(t), \mathscr{P}_D))$ and (ii) $M'' \in \mathscr{SM}((\mathscr{P}_2 \cup \mathscr{P}_C)_D)$ such that $M' = M[\mathscr{P}_D] \times M''$. □

The above theorem can be restated in the following more explicative form.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

Input: A disjunctive Datalog query $Q = \langle g(t), \mathscr{P} \rangle$, a database D;
Output: The set of stable models of $\mathscr{P}_D$ in which $g(t)$ is evaluated true;
var: $\mathscr{M}$, $\mathscr{MP}$: **set of models**;
begin
  let $\mathscr{M} := \emptyset$;
  let $\mathscr{MP} :=$*Magic_Partial* $(\langle g(t), \mathscr{P} \rangle, D)$;
  for each $M \in \mathscr{MP}$ **do**
    let $\mathscr{M} := \mathscr{M} \cup M[\mathscr{P}_D] \times \mathscr{SM}(\overline{\mathscr{P}_D / M[\mathscr{P}_D]})$;
  return $\mathscr{M}$;
end.

Fig. 3. Algorithm *Magic_Total.*

Corollary 1
Let $\mathscr{Q}' = \langle g(t), \mathscr{P} = (\mathscr{P}_R, \mathscr{P}_C) \rangle$ be a query, D be a database, and $\mathscr{Q} = \langle g(t), Disj_Magic(g(t), \mathscr{P}) \rangle$ then

- $Ans_b(\mathscr{Q}', D) \subseteq Ans_b(\mathscr{Q}, D)$, and
- $Ans_c(\mathscr{Q}', D) \supseteq Ans_c(\mathscr{Q}, D)$

Proof
The relation $Ans_b(\mathscr{Q}', D) \subseteq Ans_b(\mathscr{Q}, D)$ straightforwardly derives from Theorem 1. For the answer under cautious semantics, we distinguish two cases. If $\mathscr{SM}(\mathscr{P}_D) = \emptyset$, then any ground atom is trivially in $Ans_c(\mathscr{Q}', D)$, and hence $Ans_c(\mathscr{Q}', D) \supseteq Ans_c(\mathscr{Q}, D)$. Assume, then, that $\mathscr{SM}(\mathscr{P}_D) \neq \emptyset$. In this case, $Ans_c(\mathscr{Q}', D) = \cap_{M'} A(g, M')$, where $A(g, M')$ denotes the set of substitutions for the variables in g such that g is true in M', for each M' stable model of $\mathscr{P}_D$. Since, for each stable model M', there exists $M \in \mathscr{SM}(Disj_Magic(g(t), \mathscr{P}_D)$, with $M[\mathscr{P}_D] \subseteq M'$, it follows that $Ans_c(\mathscr{Q}', D) = \cap_{M'} A(g, M') = \cap_M A(g, M[\mathscr{P}_D])$. As $Disj_Magic(g(t), \mathscr{P}_D)$ might contain additional stable models w.r.t. those of $\mathscr{P}_D$, $Ans_c(\mathscr{Q}, D)$ is a subset of $\cap_M A(g, M[\mathscr{P}_D])$. Hence, we have that $Ans_c(\mathscr{Q}', D) = \cap_M A(g, M[\mathscr{P}_D]) \supseteq Ans_c(\mathscr{Q}, D)$. □

The above result has shed some light into the effectiveness of the Magic-Set technique for disjunctive program with constraints. Indeed, the rewriting is both *bravely complete*, i.e., under the brave semantics it guarantees to compute all the answers for the original query, and *cautiously sound*, i.e., under the cautious semantics it guarantees that no false answers are in fact computed. Actually, since we cannot prove that soundness and completeness hold at the same time in any semantics, the algorithm presented in Figure 1 will be also called *Magic_Partial* algorithm.

A natural extension, called *Magic_Total* algorithm, is shown in Figure 3, and consists of a first application of the *Magic_Partial*, and in a successive evaluation of the stable models of the program in which the binding has not been propagated. It is worth noting that the *Magic_Total* algorithm returns the set of all the models; however, it is almost trivial to modify the algorithm, in order to implement the possible and certain semantics in a more direct and efficient way.

We conclude by observing that applying the algorithm *Magic_Total* (rather than *Magic_Partial*) produces an overhead in query answering. Then, in the next section the results of some experiments are presented which quantify this overhead.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

5 Experimental results

In this section some experimental results are presented to give an idea of the improvements which can be obtained by means of this technique; the proposed algorithm has been included in the system presented in (Greco 2003), and all the experiments have been carried out by means of the DLV system (Leone *et al.* 2002) on a PC with a Pentium 4, 1.7 GHz, 512 Mbyte of RAM under the operating system Linux.

It should be pointed out that this proposal is neither a new evaluation strategy nor a new implementation, but an optimization useful for efficiently evaluating bound queries in bottom-up engines; in fact, this contribution lies in having formally proved that the Magic-Set rewriting can also be extended to deal with constraint rules, and, hence, implementation issues are subject for further research.

Nonetheless, experimental evaluation makes sense, since some classical decision and optimization problems applied in "extreme" situations have been considered, representative of a wide spectrum of real cases, in which the improvements of our techniques are negligible and highly evident, respectively.

Note that the timings considered in all the following experimental results do not include the time for the rewriting. In any case, this time does not affect the overall result as an exponential speedup in the execution times is obtained between the source and optimized (rewritten) versions.

Simple Examples. Consider disjunctive program $\mathscr{P}_1$ consisting of the following rule:

$$p(X) \vee q(X) \quad \leftarrow \quad a(X, Y)$$

and program $\mathscr{P}_2$ obtained by adding to $\mathscr{P}_1$ the constraint

$$\leftarrow p(X), a(X, Y), q(Y), X \leqslant 1.$$

Figure 4(i) shows the results obtained by considering query $\langle \mathtt{p}(1), \mathscr{P}_1 \rangle$, evaluated over the database D consisting of a set of facts $a(1,2), a(2,3), \ldots, a(k, k+1)$. The figure presents the execution time for the source program and the optimized version, obtained by applying the rewriting technique for positive queries. The experiments have been performed with databases, whose number of facts is shown on the x-axis, while the y-axis shows the time taken to evaluate the query (in seconds). The improvement of the optimized query is extremely high (observe that the scale of the y-axis is logarithmic), and is due to the fact that the optimized version propagates the binding of the query $p(1)$, with the effect of reducing the models to be computed from an exponential to a constant number.

In Figure 4(ii) the results obtained by evaluating query $\langle \mathtt{p}(1), \mathscr{P}_2 \rangle$ are presented. By comparing the results shown in Figure 4(i) and Figure 4(ii) it can be observed that also in this case there is an exponential speedup between the optimized query and the source query, as the number of ground rules in the optimized version is still constant.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

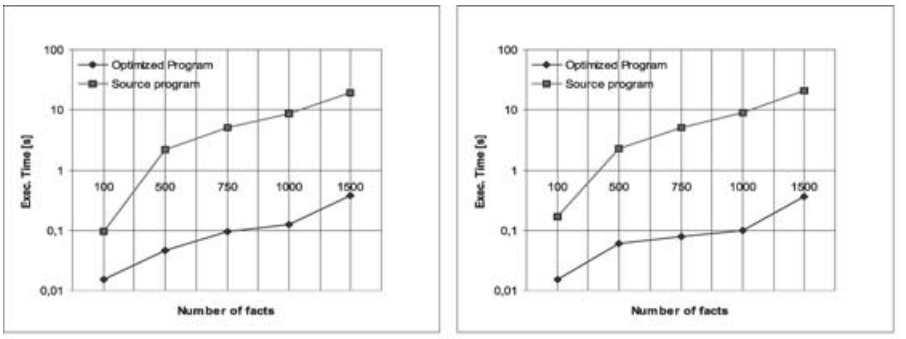


Fig. 4. Results for the query of (i) Example 1, and (ii) Example 2.

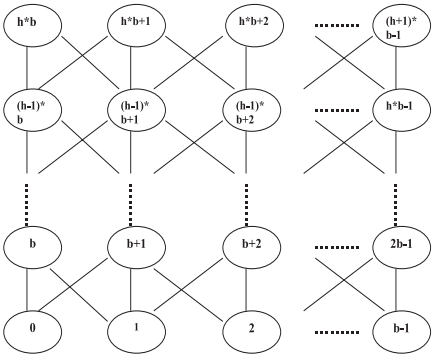


Fig. 5. Graph structures.

5.1 Search problems

For the following queries graphs have been used having the structure depicted in Figure 5 with *base* = *height* and output grade equal to 3 and 2, respectively. Here *base* denotes the number of nodes in the same layer, *height* the number of nodes in the same column and *grade* the number of arcs starting from each node not belonging to the top layer (or equivalently, the number of arcs ending at every node not belonging to the bottom layer). The number of nodes in the graph is $base \times height$, and the number of arcs is $(base-1) \times (height-1) \times grade + (base-1) + (height-1)$.

3-Coloring. The query of Example 7 is considered with input graphs consisting of two disconnected components with variable sizes; the results are shown in Figure 6 where the computation of the source query and the computation of the query rewritten using both the *Magic_Partial* and *Magic_Total* algorithm have been considered. In particular, in 6(i) the two components are of very different size, and the nodes in the query goal belong to the larger one. The graph shows the execution times as the size of the larger component changes. In Figure 6(ii) the same experiments repeated using two components with the same number of nodes are presented.

Note that in the first experiment, whose results are shown in Figure 6(i), there is no difference between the response time of the two optimized versions because

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

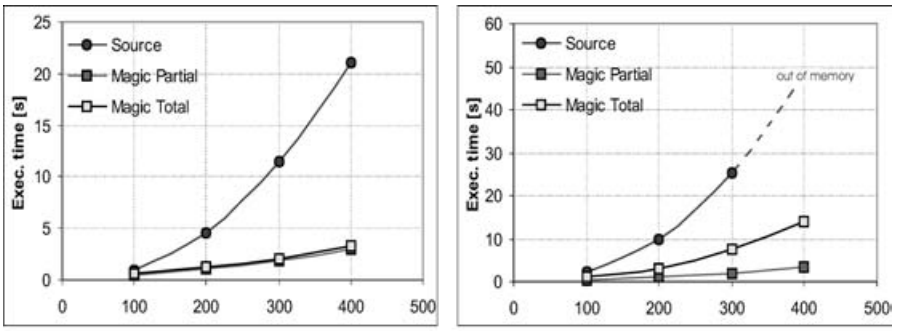


Fig. 6. Execution time for the 3-coloring problem.

the Magic-Set technique propagates the binding in the greatest component that dominates the size of the graph. For both optimized programs the advantage with respect to the source program is evident, since the size of the component on which the binding does not propagate is quite irrelevant. In Figure 6(ii) graphs with two components of the same size are being considered and, therefore, obtaining a full solution (i.e. Magic-Total) requires almost twice the time required for the partial solution (i.e. Magic-Partial).

k-Dominating Set. Given a graph $G = \langle V, E \rangle$, a subset of the vertex set $V' \subseteq V$ is a dominating set if for all $u \in V - V'$ there is a $v \in V'$ for which $(u, v) \in E$. The k-dominating set problem consists in finding a partition of the nodes into $V_1, \ldots, V_k$ disjoint dominating sets for G. The 3-Dominating Set problem, denoted by 3PDS, can be formalized by means of the following set of logic rules:

```
v1(X) ∨ nv1(X)  ←  node(X).
v2(X) ∨ nv2(X)  ←  node(X).
v3(X) ∨ nv3(X)  ←  node(X).

    ←  v1(X), v2(X).
    ←  v1(X), v3(X).
    ←  v2(X), v3(X).

    ←  nv1(X), not connected1(X).
    ←  nv2(X), not connected2(X).
    ←  nv3(X), not connected3(X).

connected1(Y)  ←  v1(X), edge(X, Y).
connected2(Y)  ←  v2(X), edge(X, Y).
connected3(Y)  ←  v3(X), edge(X, Y).
```

Note that the first group of rules and the strong constraints induce a partition of the nodes into 3 disjoint sets, $v1$, $v2$ and $v3$.

The query $\langle (\texttt{v1(1)}, \texttt{v2(2)}, \texttt{v3(3)}), \texttt{3DS} \rangle$ is supplied on a graph G consisting of two components C_1 and C_2. The size of C_1 is fixed and it is assumed that this component contains nodes 1, 2 and 3. In some experiments the size of C_2 was varied and the response time calculated for the source program described above, and the optimized program produced by the Algorithm in Fig. 1.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

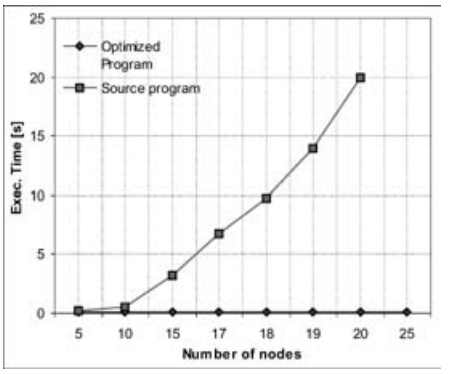


Fig. 7. Execution time for the 3-Dominating Set problem.

The results, presented in Figure 7, show that the optimized program is not affected by the size of C_2 as there is no way of propagating the binding from nodes 1,2 and 3.

5.2 Optimization problems

In this section the possibility of using Magic-Set techniques for the optimization of queries over disjunctive Datalog programs is explored. In order to also express optimization problems the approach used in the DLV system is considered as well as the consideration, in addition to strong constraints, weak constraints. Weak constraints represent constraints which should be respected, but if they cannot be eventually enforced, then they only invalidate the portion of the program which they are concerned with Greco (1998). Therefore, strong constraints express a set of conditions that have to be satisfied, while weak constraints express a set of desiderable conditions that may be violated and their informal semantics is to minimize the number of violated instances.

A *weak constraint* is a rule of the form $\Leftarrow b_1, \ldots, b_k, \neg b_{k+1}, \ldots, \neg b_{k+m}$. Given a program $P \cup W$ where P is a set of rules and W a set of weak constraints, an interpretation M is a stable model for $P \cup W$ if M is a stable model for P which minimizes the number of rules not satisfied in $ground(W)$. Thus, the stable models of P can be ordered w.r.t. the number of weak constraints not satisfied in $ground(W)$; the preferred stable models are those which minimize this number.

Like strong constraints, weak constraints are not used to infer atoms, but only to check that the computed set verifies a given property. In Buccafurri *et al.* (2000) and Greco (1998) it is proved that the introduction of weak constraints allows the solution of optimization problems since each weak constraint can be regarded as an "objective function" of an optimization problem.

Example 12

Given a graph $G = \langle V, E \rangle$, defined by means of the unary predicate *node* and the binary predicate *edge*, we can model the `MAX_CLIQUE` problem, asking for the clique of G having maximum size, by means of the following disjunctive Datalog program

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

with both *strong* and *weak* constraints:

$$\mathtt{c(X)} \vee \mathtt{nc(X)} \quad \leftarrow \quad \mathtt{node(X)}.$$

$$\leftarrow \quad \mathtt{c(X)}, \mathtt{c(Y)}, \mathtt{X} \neq \mathtt{Y}, \mathtt{not\ edge(X,Y)}.$$

$$\Leftarrow \quad \mathtt{nc(X)}.$$

Note that the first rule is used for creating all possible partitions of nodes into c and nc, and the second one (i.e. the strong constraint) is used for ensuring that c is a clique, i.e. each couple of nodes in the clique must be connected by an edge, while the weak constraint minimizes the number of vertices that are not in the clique, or equivalently it maximizes the size of c.

Consider the query $\langle c(1), \mathtt{MAX_CLIQUE}\rangle$ over the graph of Figure 2, asking whether node 1 belongs to a clique of maximum size. It is easy to see that nodes 1, 9 and 10 form a clique of size 3, which is also the size of the maximum clique in the component C_1. Component C_2 is a clique of size 4, and, hence, the above query is *false*. In contrast, the query $\langle c(12), \mathtt{MAX_CLIQUE}\rangle$ asking whether node 12 belongs to a clique of maximum size is evaluated *true*.

Now, observe that both the above queries are evaluated true, by using the Magic-Set rewriting; in fact, when we ask whether 1 belongs to a clique of maximum size, there is no way for propagating the binding in component C_2, where the maximum clique actually is. □

From the above example, it is clear that any query optimization technique applied in the presence of weak constraints, will eventually lead to a different semantics consisting in a 'local' optimization, rather than in a global one.

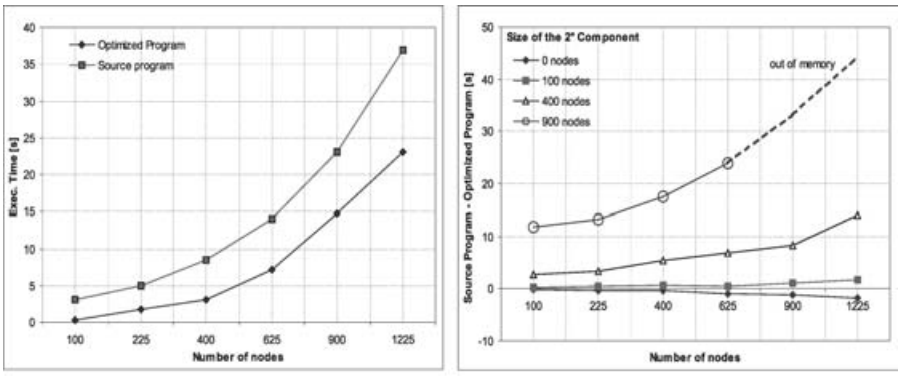
In many circumstances, this semantics can also be desirable. In all other circumstances, the global solution can be simply obtained by comparing the different solutions obtained while making local optimizations, by exploiting the same approach used in developing the *Magic_Total* algorithm.

The extension of the technique to deal also with weak constraints is outside the scope of this paper. Therefore, in the next two paragraphs some hints for future research are presented. In particular, two optimization problems are considered where the propagation of bindings defines a partition of the input graph G into two separated components G_1 and G_2 and the optimal solution can be obtained by first computing an optimal solution using component G_1 and next finding the global optimal solution starting from the partial solution obtained in the first step and considering component G_2.

However, in many cases an optimization problem Opt over a given graph G can be defined by decomposing the graph G into separated subgraphs $G_1, \ldots, G_k$ and to compute the k subproblems. Thus, let $Opt(G_i, O_i)$ be the optimization problem over the (sub-)graph G_i using the partial solution O_i, we say that Opt is decomposable if $Opt(G, \emptyset) = Opt(G_k, O_k)$ where $O_1 = Opt(G_1, \emptyset)$ and $O_i = Opt(G_i, O_{i-1})$ for $i \in [2..k]$.

Max Clique. Let us consider the program of Example 12, and let us supply the query $\langle \mathtt{c(1)}, \mathtt{MAX_CLIQUE}\rangle$, by using a graph consisting of two components with the structure shown in Figure 5. The experimental results are shown in Figure 8. In particular Figure 8(i) shows the execution time for a graph consisting of two

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press



(i) (ii)

Fig. 8. The Max Clique problem.

components having the same size. In Figure 8(ii) the difference in the execution time between the source program and the optimized one for different sizes of the second component it is shown. Note that in Figure 8(ii), if the second component is empty (0 nodes) the source program performs a little better than the optimized one as the second program presents an overhead due to the instantiation and the computation of the magic rules. The advantage of using the optimized program becomes more evident as the size of the second component increases.

It is worth noting that the previous example is a prototype of the *guess* and *check* paradigm, that, as already pointed out, has been proved to be the most intuitive way for expressing NP (optimization) problems.

Min Coloring. Given a graph $G = \langle V, E \rangle$ a coloring for G is minimum if, in the assignment of colors to vertices, it uses the minimum number of colors. The disjunctive Datalog program modelling the `MIN_COLORING` problem is the following:

```
col(X, I) ∨ ncol(X, I)   ←   node(X), color(I).

  ←   col(X, I), col(Y, I), edgeT(X, Y).
  ←   col(X, I), col(X, J), I ! = J.
  ←   node(X), not colored(X).

colored(X)   ←   col(X, I).
used(I)      ←   col(X, I).

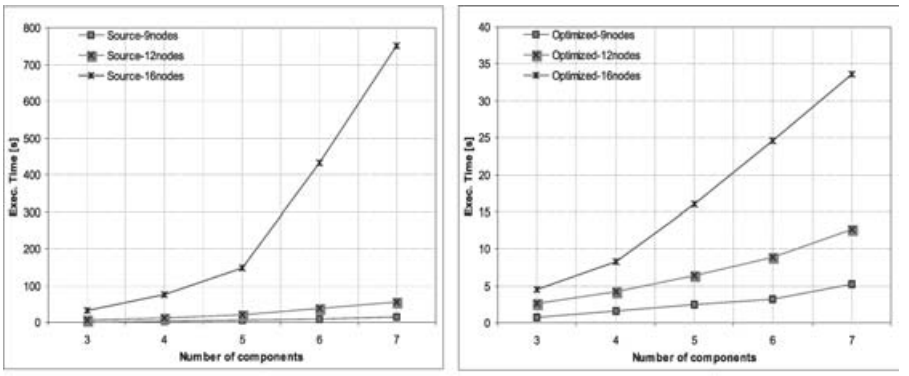
⇐   used(I).
```

The first rule guesses a coloring for the graph; the set of strong constraints checks the guess that two joined vertices do not have the same color, and that each vertex is assigned to exactly one color; the weak constraint requires that the number of colors used is minimum.

The query $\langle$`col(1, c1), MIN_COLORING`$\rangle$ is supplied on different graph topologies. Some of the results are shown in Figure 9.

In particular the results obtained by augmenting the number of disconnected components, and by augmenting the cardinality of the components have been investigated. Figures 9(i) and 9(ii) show, respectively, the performance of the optimized

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press



(i) (ii)

Fig. 9. Execution time for the Min coloring problem.

program and of the source program with a different number of components. In all the experiments the first component contains node 1.

6 Conclusions

In this paper a technique has been proposed for the optimization of bound queries over disjunctive deductive databases with constraints (a simple and expressive form of unstratified negation), which extends a previous technique suitable for disjunctive Datalog programs. As the usual way of expressing declaratively hard problems 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, the technique proposed here is highly relevant for the optimization of queries expressing hard problems.

The proposed approach is based on the use of a binding propagation technique which, by reducing the size of the data relevant to answer the query, is suitable for minimizing the complexity of computing a single model and the whole number of models to be considered.

The main contribution of the paper lies in the definition of a rewriting algorithm which systematically utilizes the query goal to propagate the binding through both the rules and the constraints thereby avoiding the computation of useless models. An interesting peculiarity of the formalization proposed here is that it is completely independent of the particular strategy adopted for propagating the binding: in this way, the results are completely orthogonal to the Magic-Set technique in itself, and, hence, to the results in Greco (2003). The value of the technique has been proved by several experiments.

Acknowledgement

The authors thank Nicola Leone and Wolfgang Faber for useful suggestions.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

References

Bancilhon, F., Maier, D., Sagiv, Y. and Ullman, J. D. 1986. Magic Sets and Other Strange Ways to Implement Logic Programs. *Proceedings Int. Symposium on Principles of Database Systems.* 1–16.

Baral, C., Lobo, J. and Minker, J. 1992. Generalised disjunctive well-founded semantics for logic programs. *Annals of Mathematics and Artificial Intelligence 5,* 19, 89–132.

Beeri, C. and Ramakrisnhan, R. 1991. On the power of magic. *Journal of Logic Programming 10,* 1–4, 255–259.

Brass, S. and Dix, J. 1994. A general Approach to Bottom-Up Computation of Disjunctive Semantics. *In Proceedings of the* Nonmonotonic Extensions of Logic Programming *(ICLP '94 Workshop)*, pp.127–155.

Brass, S. and Dix, J. 1999. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming 38,* 3, 167–213.

Buccafurri, F., Leone, N. and Rullo, P. 2000. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering 12,* 5 (September/October), 845–860.

Chen, W. and Warren, D. S. 1993. Query Evaluation under the Well-Founded Semantics. *In Proceedings of the ACM Symposium on Principles of Database Systems*, pp. 168–179.

Chen, W. and Warren, D. S. 1996. Efficient top-down computation of queries under the well-founded semantics. *Journal of the ACM 43,* 1, 20–74.

Cholewiński, P., Marek, V. W. and Truszczyński, M. 1996. Default Reasoning System DeReS. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR '96)*. Morgan Kaufmann, Cambridge, MA, pp. 518–528.

Clark, K. 1978. Negation as Failure. In: *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Plenum Press, New York, pp. 293–322.

Eiter, T., Gottlob, G. and Mannila, H. 1997. Disjunctive Datalog. *ACM Transactions on Database Systems 22,* 3 (September), 364–418.

Eiter, T., Leone, N. and Saccà, D. 1997. On the Partial Semantics for Disjunctive Deductive Databases. *Annals of Mathematics and Artificial Intelligence 19,* 1–2 (April), 59–96.

Fernández, J. and Minker, J. 1995. Bottom-Up Computation of Perfect Models for Disjunctive Theories. *Journal of Logic Programming 25,* 1, 33–51.

Fernández, J., Minker, J. and Yahya, A. H. 1995. Computing perfect and stable models using ordered model trees. *Computational Intelligence 11*, 89–112.

Furfaro, F., Greco, G. and Greco, S. 2004. Minimal Founded Semantics for Disjunctive Logic Programming. *Journal of the Theory and Practice of Logic Programming 4,* 1, 75–93.

Gelfond, M. and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In: *Logic Programming: Proceedings Fifth Intl Conference and Symposium.* MIT Press, Cambridge, MA, pp. 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing 9*, 365–385.

Grant, J. and Minker, J. 1986. Answering Queries in Indefinite Databases and the Null Value Problem. *Advances in Computing Research 3*, 89–112.

Greco, S. 1998. Non-Determinism and Weak Constraints in Datalog. *New Generation Computing 16,* 4, 373–396.

Greco, S. 1999. Optimization of Disjunctive Queries. *Proceedings of the 16th International Conference on Logic Programming*, pp. 441–455.

Greco, S. 2003. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Transactions on Knowledge and Data Engineering 15,* 2, 368–385.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

Greco, S., Sacca, D. and Zaniolo, C. 1995. The PushDown Method to Optimize Chain Logic Programs (Extended Abstract). *Proceedings Int. Colloquim on Automata, Languages and Programming*, pp. 523–534.

Janhunen, T., Niemelä, I., Simons, P. and You, J.-H. 2000. Unfolding Partiality and Disjunctions in Stable Model Semantics. In *In Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pp. 411–419.

Johnson, C. A. 1999. Processing deductive databases under the disjunctive stable model semantics. *Fundamenta Informaticae 40,* 1, 31–51.

Johnson, C. A. 2001. On the computation of the disjunctive well-founded semantics. *Journal of Automated Reasoning 26,* 4, 333–356.

Johnson, C. A. 2002. Processing indefinite deductive databases under the possible model semantics. *Fundamenta Informaticae 49,* 4, 325–347.

Kowalski, R. A. 1984. Predicate Logic as Programming Language. *Computers for Artificial Intelligence Applications 3*, 68–73.

Leone, N., Pfeifer, G., Faber, W., Calimeri, F., Dell'Armi, T., Eiter, T., Gottlob, G., Ianni, G., Ielpa, G., Koch, C., Perri, S. and Polleres, A. 2002. The DLV System. *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, S. Flesca, S. Greco, G. Ianni, and N. Leone, Eds. Number 2424 in Lecture Notes in Computer Science, pp. 537–540.

Lin, F. and Zhao, Y. 2002. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*. AAAI Press/MIT Press, Edmonton, Alberta, Canada.

Liu, K. C. and Sunderraman, R. 1990. Indefinite and Maybe Information in Relational Databases. *ACM Transactions on Database Systems 15*, 1–39.

Lobo, J., Minker, J. and Rajasekar, A. 1992. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, MA.

Minker, J. 1982. On Indefinite Data Bases and the Closed World Assumption. In *Proceedings of the 6th Conference on Automated Deduction (CADE '82)*, D. Loveland, Ed. Number 138 in Lecture Notes in Computer Science. Springer, New York, pp. 292–308.

Minker, J. and Rajasekar, A. 1990. A Fixpoint Semantics for Disjunctive Logic Programming. *Journal of Logic Programming 9*, 45–74.

Niemelä, I. 1996. A tableau calculus for minimal model reasoning. *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*. 278–294.

Niemelä, I. 2003. Answer set programming: an approach to declarative problem solving. In *Proceedings of the Scandinavian Conference on Artificial Intelligence*, pp. 189–191.

Niemelä, I. and Simons, P. 1997. Smodels – An Implementation of the Stable Model and Well-founded Semantics for Normal Logic Programs. *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, J. Dix, U. Furbach, and A. Nerode, Eds. Lecture Notes in AI (LNAI), vol. 1265. Springer Verlag, Dagstuhl, Germany, pp. 420–429.

Przymusinski, T. C. 1991. Stable Semantics for Disjunctive Programs. *New Generation Computing 9*, 401–424.

Przymusinski, T. C. 1995. Static Semantics for Normal and Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence 14*, 323–357.

Ramakrishnan, R., Sagiv, Y., Ullman, J. D. and Vardi, M. Y. 1993. Logical Query Optimization by Proof-Tree Transformation. *Journal of Computer and System Sciences 47,* 1, 222–248.

Rao, P., Sagonas, K. F., Swift, T., Warren, D. S. and Freire, J. 1997. XSB: A System for Efficiently Computing Well-Founded Semantics. *Proceedings of the 4th International*

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press

Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97), J. Dix, U. Furbach, and A. Nerode, Eds. Number 1265 in Lecture Notes in AI (LNAI). Springer Verlag, Dagstuhl, Germany, pp. 2–17.

Ross, K. 1990. The Well-Founded Semantics for Disjunctive Logic Programs. In: *Deductive and Object-Oriented Databases*, W. Kim, J.-M. Nicolas, and S. Nishio, Eds. Elsevier, pp. 385–402.

Shen, Y. D., Yuan, Y. D. and You, J. H. 2002. SLT-resolution for the well founded semantics. *Journal of Automated Reasoning 28,* 1, 53–97.

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence 138*, 181–234.

Ullman, J. D. 1989a. Bottom up beats top-down for datalog. *Proceedings Int. Symposium on Principles of Database Systems*. 140–149.

Ullman, J. D. 1989b. *Principles of Database and Knowledge Base Systems*. Computer Science Press.

Wang, K. 2000. Argumentation-based abduction in disjunctive logic programming. *Journal of Logic Programming 45*, 105–141.

Wang, K. 2001. A Top-down Procedure for Disjunctive Well-founded Semantics. In *Proceedings International Joint Conference on Automated Reasoning*, pp. 305–317.

Yahya, A. H. 2000. Minimal Model Generation for Refined Answering of Generalized Queries in Disjunctive Deductive Databases. *Journal of Data and Knowledge Engineering*, 219–249.

Yahya, A. H. 2002. Duality for goal-driven query processing in disjunctive deductive databases. *Journal of Automated Reasoning 28,* 1, 1–34.

Yahya, A. H. and Henschen, L. 1985. Deduction in Non-Horn Databases. *Journal of Automated Reasoning 1,* 2, 141–160.

Yuan, L. Y. and Chiang, D. A. 1998. A Sound and Complete Query Evaluation Algorithm for Relational Databases with Disjunctive Information. *Proceedings Int. Symposium on Principles of Database Systems*, pp. 66–74.

Zumpano, E., Greco, S., Trubitsyna, I. and Veltri, P. 2004. On the Semantics and Expressive Power of Datalog-like Languages for NP Search and Optimization Problems. *Proceedings of the ACM Symposium on Applied Computing*. 692–697.

https://doi.org/10.1017/S1471068404002273 Published online by Cambridge University Press