Magic Sets and their Application to Data Integration*

Wolfgang Faber**, Gianluigi Greco, and Nicola Leone

Mathematics Department, University of Calabria, 87030 Rende, Italy,
{faber,ggreco,leone}@mat.unical.it

Abstract. We propose a generalization of the well-known Magic Sets technique to Datalog$^\neg$ programs with (possibly unstratified) negation under stable model semantics. Our technique produces a new program whose evaluation is generally more efficient (due to a smaller instantiation), while preserving soundness under cautious reasoning. Importantly, if the original program is consistent, then full query-equivalence is guaranteed for both brave and cautious reasoning, which turn out to be sound and complete.
In order to formally prove the correctness of our Magic Sets transformation, we introduce a novel notion of modularity for Datalog$^\neg$ under the stable model semantics, which is relevant per se. We prove that a module can be evaluated independently from the rest of the program, while preserving soundness under cautious reasoning. For consistent programs, both soundness and completeness are guaranteed for brave reasoning and cautious reasoning as well.
Our Magic Sets optimization constitutes an effective method for enhancing the performance of data-integration systems in which query-answering is carried out by means of cautious reasoning over Datalog$^\neg$ programs. In fact, preliminary results of experiments in the EU project INFOMIX, show that Magic Sets are fundamental for the scalability of the system.

1 Introduction

Datalog$^\neg$ programs are function-free logic programs where negation may occur in the bodies of rules [1]. Datalog$^\neg$ with stable model semantics [2, 3] [1] is a very expressive query language in a precise mathematical sense: under brave (cautious) reasoning Datalog$^\neg$ allows to express every query that is decidable in the complexity class NP (co-NP) [4]. In the 90s, Datalog$^\neg$ was not considered very much in the database community, mainly because of the high complexity of its evaluation (NP or co-NP depending on the reasoning modality [5–7]). However, the emerging of important database applications strictly requiring the co-NP expressiveness of Datalog$^\neg$ (see below and Sect. 5), along with the availability of a couple of effective Datalog$^\neg$ systems, like DLV [8] and Smodels [9], has renewed the interest in this language.

Our motivation to study optimization techniques for Datalog$^\neg$, comes from the data-integration area that we are investigating within the EU project "INFOMIX: Boosting

* This work was supported by the European Commission under projects IST-2002-33570 INFOMIX, and IST-2001-37004 WASP.
** Funded by an APART grant of the Austrian Academy of Sciences.

[1] Unless explicitly specified, Datalog$^\neg$ will always denote Datalog with negation under stable model semantics in this paper.

Information Integration". INFOMIX is a powerful data-integration system, which is able to deal with both inconsistent and incomplete information. Following many recent proposals (see, e.g., [10–15]), query answering in the INFOMIX data-integration system is reduced to cautious reasoning on Datalog$^\neg$ programs under stable model semantics. This reduction is possible since query answering in data-integration systems is co-NP-complete (in our setting and also in many other data-integration frameworks [15, 13]) like cautious reasoning on (unstratified) Datalog$^\neg$ programs under the stable model semantics [5–7].

Dealing with a co-NP-complete problem can appear unfeasible, or even crazy in a database setting where input may be very large. However, our present results show that suitable optimization techniques can "localize" the computation and limit the inefficient (co-NP) computation to a very small fragment of the input, obtaining fast query-answering, even in a powerful data-integration framework. The main contribution of the paper is the following.

- ▷ We define the new notions of *independent set* and *module* for Datalog$^\neg$, allowing us to identify program fragments which can be evaluated "independently", disregarding the rest of the program. The new notion of module is crucial for proving the correctness of our magic set method. It is strictly related to the splitting sets of [16], and to the modules of [17]; but we demonstrate that our notion has stronger semantic properties, which are useful for the computation.
- ▷ We design an extension of the Magic Set algorithm for general Datalog$^\neg$ programs (MS$^\neg$ algorithm for short). We show that different to stratified Datalog$^\neg$, where bindings are propagated only head-to-body in a rule, unstratified negation requires bindings to be propagated also body-to-head in general, in order to guarantee query equivalence. Such a body-to-head propagation, which has been carefully incorporated in our MS$^\neg$ method, allows us to properly deal with those rules (called *dangerous* rules) which may be the source of semantic problems. And, in fact, we prove that the rewriting generated by MS$^\neg$ is query equivalent to the input program $\mathcal{P}$ (under both brave and cautious semantics), provided that $\mathcal{P}$ is consistent. Even if the program is inconsistent soundness under cautious semantics and completeness under brave semantics are guaranteed by our transformation.
- ▷ We show that our method can be profitably exploited for query optimization in powerful data integration systems, where also incompleteness and inconsistency of data is dealt with; and we apply MS$^\neg$ in the EU project INFOMIX. Specifically, we show that our Magic Set technique can be employed for the optimization of the logic programs specifying the database repairs [2] [10–15] (the queries on the data-integration system are eventually evaluated on these programs). MS$^\neg$ always ensures the full query equivalence of the optimized program w.r.t. the original one, since such programs are guaranteed to be consistent (a database repair always exists). Preliminary results of experiments, that we carried out on a real application scenario, confirmed the viability and the effectiveness of our approach: the application of the Magic Set method allows us to "localize" the computation, and to obtain fast query-answering, even in a powerful data-integration framework.

[2] Note that no previous magic-set technique is applicable, since these programs are unstratified and are to be evaluated under stable models semantics.

2 Preliminaries and Notations

2.1 Datalog$^\neg$ Queries

An *atom* $p(t_1, \ldots, t_k)$ is composed of a predicate symbol p of *arity* k and terms $t_1, \ldots, t_k$, which can either be constants or variables. A (*Datalog$^\neg$ rule*) r is of the form h :– $b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n$., where $h, b_1, \cdots, b_n$ are atoms and $0 \leq m \leq n$.$H(r) = h$ is the *head* of r, while $B(r) = B^+(r) \cup B^-(r)$ is the body of r, where $B^+(r) = \{b_1, \ldots, b_m\}$ is the positive and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$ the negative body of r. Finally, let $Atoms(r) = \{H(r)\} \cup B(r)$ denote the set of atoms in r, and $Atoms(\mathcal{P}) = \{Atoms(r) \mid r \in \mathcal{P}\}$ for a program $\mathcal{P}$.

A rule r with $H(r) = p(t_1, \ldots, t_k)$ is a defining rule for predicate p. If for a rule r, $B(r) = \emptyset$ holds, the rule is a *fact*. If all defining rules of a predicate p are facts, then p is an *EDB predicate*, otherwise it is an *IDB predicate*. A rule r is *positive* if $B^-(r) = \emptyset$. Throughout this paper, we assume that rules are *safe*, that is, each variable of a rule r appears in $B^+(r)$ [1].

A *datalog program with negation* (Datalog$^\neg$ program for short) $\mathcal{P}$ is a finite set of rules. A *query* $\mathcal{Q}$ is simply an atom. We call an atom, rule, program, or query *ground*, if they do not contain variables. Given a program $\mathcal{P}$, we denote by $Ground(\mathcal{P})$ the set of all the rules obtained by applying to each rule $r \in \mathcal{P}$ all possible substitutions from the variables in r to the set of all the constants in $\mathcal{P}$.

Let the *base* $B_\mathcal{P}$ of $\mathcal{P}$ be the set of ground atoms constructible from predicates and constants in $\mathcal{P}$. A set of atoms $I \subseteq B_\mathcal{P}$ is an *interpretation* for $\mathcal{P}$. Given an interpretation I and set of rules T, let the restriction of I to T be defined as $I/_T = I \cap Atoms(T)$. In a similar way, let the restriction of a set S of interpretations to T be defined as $S/_T = \{I/_T \mid I \in S\}$, and let the restriction of a rule r to T ($R/_T$) be defined by dropping all body literals which are not in $Atoms(T)$. Given a positive rule $r \in Ground(\mathcal{P})$, an interpretation I satisfies r if $B(r) \subseteq I$ implies $H(r) \in I$. An interpretation I is a *model* of a Datalog program $\mathcal{P}$ if I satisfies all rules in $Ground(\mathcal{P})$. The *stable model* of a Datalog program $\mathcal{P}$ is the unique subset-minimal model $\text{MM}(\mathcal{P})$.

Given a Datalog$^\neg$ program $\mathcal{P}$ and an interpretation I, the *Gelfond-Lifschitz transform* $\mathcal{P}^I$ is defined as $\{H(r) \text{ :– } B^+(r) \mid r \in Ground(\mathcal{P}) : I \cap B^-(r) = \emptyset\}$. The set of *stable models* of a Datalog$^\neg$ program $\mathcal{P}$, denoted by $\text{SM}(\mathcal{P})$, is the set of interpretations I, such that $I = \text{MM}(\mathcal{P}^\text{I})$. $\mathcal{P}$ is *consistent* if $\text{SM}(\mathcal{P}) \neq \emptyset$, otherwise *inconsistent*.

Let a be a ground atom and a program $\mathcal{P}$, then a is *cautious consequence* of $\mathcal{P}$, denoted by $\mathcal{P} \models_c a$, if $\forall M \in \text{SM}(\mathcal{P}) : \text{a} \in \text{M}$; a is a *brave consequence* of $\mathcal{P}$, denoted by $\mathcal{P} \models_b a$, if $\exists M \in \text{SM}(\mathcal{P}) : \text{a} \in \text{M}$. Given a query $\mathcal{Q} = b$, $Ans_c(\mathcal{Q}, \mathcal{P})$ denotes the set of substitutions ϑ, such that $\mathcal{P} \models_c b\vartheta$; $Ans_b(\mathcal{Q}, \mathcal{P})$ denotes the set of substitutions ϑ, such that $\mathcal{P} \models_b b\vartheta$.

Let $\mathcal{P}$ be a Datalog$^\neg$ program and let $\mathcal{F}$ be a set of facts. Then, we denote by $\mathcal{P}_\mathcal{F}$ the program $\mathcal{P}_\mathcal{F} = \mathcal{P} \cup \mathcal{F}$. Let $\mathcal{P}$ and $\mathcal{P}'$ be Datalog$^\neg$ programs and $\mathcal{Q}$ be a query. Then, $\mathcal{P}$ is *brave-sound* w.r.t. $\mathcal{P}'$ and $\mathcal{Q}$, denoted $\mathcal{P} \sqsubseteq^b_\mathcal{Q} \mathcal{P}'$, if $Ans_b(\mathcal{Q}, \mathcal{P}_\mathcal{F}) \subseteq Ans_b(\mathcal{Q}, \mathcal{P}'_\mathcal{F})$ is guaranteed for all set of facts $\mathcal{F}$; $\mathcal{P}$ is *cautious-sound* w.r.t. $\mathcal{P}'$ and $\mathcal{Q}$, denoted $\mathcal{P} \sqsubseteq^c_\mathcal{Q} \mathcal{P}'$, if $Ans_c(\mathcal{Q}, \mathcal{P}_\mathcal{F}) \subseteq Ans_c(\mathcal{Q}, \mathcal{P}'_\mathcal{F})$ for all $\mathcal{F}$. $\mathcal{P}$ is *brave-complete* (resp., *cautious-complete*) w.r.t. $\mathcal{P}'$ and $\mathcal{Q}$, if $\mathcal{P} \sqsupseteq^b_\mathcal{Q} \mathcal{P}'$ (resp., $\mathcal{P} \sqsupseteq^c_\mathcal{Q} \mathcal{P}'$). Finally, $\mathcal{P}$ and

P' are *brave-equivalent* (resp., *cautious-equivalent*) w.r.t. $\mathcal{Q}$, denoted by $\mathcal{P} \equiv^b_{\mathcal{Q}} \mathcal{P}'$ (resp. $\mathcal{P} \equiv^c_{\mathcal{Q}} \mathcal{P}'$), if $\mathcal{P} \subseteq^b_{\mathcal{Q}} \mathcal{P}'$ and $\mathcal{P} \supseteq^b_{\mathcal{Q}} \mathcal{P}'$ (resp., $\mathcal{P} \subseteq^c_{\mathcal{Q}} \mathcal{P}'$ and $\mathcal{P} \supseteq^c_{\mathcal{Q}} \mathcal{P}'$).

With every program $\mathcal{P}$, we associate a marked directed graph $DG_{\mathcal{P}} = (N, E)$, called the *predicate dependency graph* of $\mathcal{P}$, where (i) each predicate of $\mathcal{P}$ is a node in N, and (ii) there is an arc (a, b) in E directed from node a to node b if there is a rule $r \in \mathcal{P}$ such that two predicates b and a of literals appear in $H(r)$ and $B(r)$, respectively. Such an arc is marked if a appears in $B^-(r)$. An *odd cycle* in $DG_{\mathcal{P}}$ is a cycle comprising an odd number of marked arcs. One can also define the *atom dependency graph* $DG^A_{\mathcal{P}}$ of a ground program $\mathcal{P}$, by considering atoms rather than predicates.

3 Modularity Results

The backbone of optimizations techniques like Magic Sets is to (automatically) identify a part of the (ground) program, which can be used instead of the entire program to single out the *query program* (the part which is sufficient to answer the query). In the negation-free or stratified setting it is sufficient to examine reachability in the head-to-body direction. Negation under the stable semantics also gives rise to (partial) inconsistency, which may be triggered by activating an inconsistent part of the program in the body-to-head direction. To this end we will first present a way to identify possibly inconsistent parts of a program. Note that in this section we deal with ground programs.

Definition 1. Let $\mathcal{P}$ be a program (resp., ground program), and d be an predicate (resp., atom) of $\mathcal{P}$. Then, we say that d is *dangerous* if either (i)d occurs in an odd cycle of $DG_{\mathcal{P}}$ (resp., $DG^A_{\mathcal{P}}$), or (ii) d occurs in the body of a rule with a dangerous head predicate (resp., atom). A rule r is *dangerous*, if it contains a dangerous predicate (resp., atom) in the head. □

In principle, one can differentiate between conditional and unconditional sources of inconsistencies. In the approach we present here, we are concerned with the first type. In particular, that "isolated" inconsistencies are not covered, though one could easily come up with a modified definition to account also for these. Intuitively, an *independent atom set* of a ground program $\mathcal{P}$ is a set S of atoms whose semantics is not affected (apart from unconditional inconsistencies) by the remaining atoms of $\mathcal{P}$, and can therefore be evaluated by disregarding the other atoms. Independent atom sets induce a corresponding module of $\mathcal{P}$.

Definition 2. An *independent atom set* of a program $\mathcal{P}$ is a set $S \subseteq B_{\mathcal{P}}$ such that for each atom $a \in S$ the following holds: (1) if $a = H(r)$ for a rule $r \in \mathcal{P}$ then $Atoms(r) \subseteq S$, and (2) if a appears in the body of a dangerous rule $r \in \mathcal{P}$ then $Atoms(r) \subseteq S$. A subset T of a program $\mathcal{P}$ is a *module* if $T = \{r \mid H(r) \in S\}$ for some independent set S. □

Example 1. Consider the following program $\mathcal{P}_1$:

$$z \text{ :- } y, \text{not } z. \quad y \text{ :- } q. \quad p \text{ :- not } q. \quad q \text{ :- not } p. \quad a \text{ :- } p, \text{not } b. \quad b \text{ :- } p, \text{not } a.$$

Independent sets for $\mathcal{P}_1$ are $\{p, q, y, z\}$, $\emptyset$ and $\{p, q, y, z, a, b\}$, of which the first is the only non-trivial one. The corresponding module T of $\mathcal{P}_1$ is

$$z \text{ :- } y, \text{not } z. \quad y \text{ :- } q. \quad p \text{ :- not } q. \quad q \text{ :- not } p.$$ □

We next state the relationships between stable models of a program and its modules.

Theorem 1. *Let T be a module of a program $\mathcal{P}$, then (i)* $\mathrm{SM}(\mathcal{P})/_{\mathrm{T}} \subseteq \mathrm{SM}(\mathrm{T})$. *Moreover, if $\mathcal{P}$ is consistent, then (ii)* $\mathrm{SM}(\mathrm{T}) = \mathrm{SM}(\mathcal{P})/_{\mathrm{T}}$.

Proof. (i) If $\mathcal{P}$ is inconsistent, then the statement trivially holds as $\mathrm{SM}(\mathcal{P})/_{\mathrm{T}} = \emptyset$. So in the following we will assume that $\mathcal{P}$ is consistent.

We show that if any interpretation I is a stable model of $\mathcal{P}$, then $I/_T$ is also a stable model of T: Recall that $T \subseteq \mathcal{P}$ and note that all rules contain only atoms of $Atoms(T)$, by item 1 of Definition 2. Next, observe that $T^{I/_T} = \mathcal{P}^I/_T$, hence $T^{I/_T} \subseteq \mathcal{P}^I$, and therefore since I is a model of $\mathcal{P}^I$, it is also a model of $T^{I/_T}$. $I/_T$ can be shown to be the minimal model of $T^{I/_T}$ by observing that if a model $J \subset I$ of $T^{I/_T}$ would exist, one could construct $I_J = J \cup \{H(r) \mid r \in (\mathcal{P}^I - T^I) \wedge B(r) \subseteq (I - I/_T) \cup J\}$ (J extended with the part of I which is not from T, which still follows from J). Clearly, $I_J \subset I$ is then a model of $\mathcal{P}^I$, contradicting the assumption that $I \in \mathrm{SM}(\mathcal{P})$.
(ii, Sketch) Since (i) holds also for consistent programs, what remains to show is $\mathrm{SM}(\mathrm{T}) \subseteq \mathrm{SM}(\mathcal{P})/_{\mathrm{T}}$ for consistent $\mathcal{P}$. We show that for any stable model I of T, a stable model J exists such that $J/_T = I$. It has been shown that any odd-cycle-free Datalog$^\neg$ program is consistent [18]. Now observe that the only odd cycles in $\mathcal{P} - T$ are independent of T by item 2 of Definition 2. Since $\mathcal{P}$ is assumed to be consistent, such odd cycles can be deactivated by the presence of some atoms of $\mathcal{P} - T$, which, by Definition 2, are completely independent of T, such that a set of appropriate atoms K of $\mathcal{P} - T$ exists such that $K \cup I = J$. □

Corollary 1. *Let T be a module of a consistent program $\mathcal{P}$. Then, each stable model of $\mathcal{P}$ can be obtained by enlarging a stable model of T.*

From Thm. 1, we can obtain similar results for query answering:

Theorem 2. *Given a ground atom q belonging to a module T of $\mathcal{P}$, then (1)* $(T \models_c q) \Rightarrow (\mathcal{P} \models_c q)$, *and (2)* $(T \models_b q) \Leftarrow (\mathcal{P} \models_b q)$. *Moreover, if $\mathcal{P}$ is consistent, then (1)* $(T \models_c q) \Leftrightarrow (\mathcal{P} \models_c q)$, *and (2)* $(T \models_b q) \Leftrightarrow (\mathcal{P} \models_b q)$.

Proof. If $\mathrm{SM}(\mathcal{P}) = \emptyset$ then $\mathcal{P} \models_c q$ for any $q \in Atoms(T)$, while $\mathcal{P} \models_b q$ for no $q \in Atoms(T)$. Therefore in this case, the implications are trivially satisfied. So from now on, consider $\mathrm{SM}(\mathcal{P}) \neq \emptyset$. In this case, the set of cautious consequences is $\bigcap \mathrm{SM}(\mathcal{P})$ and the set of brave consequences is $\bigcup \mathrm{SM}(\mathcal{P})$ in any case. Using Thm. 1 we can obtain the following:
1. We have to show that if q is in all stable models of T, then it is also in all stable models of $\mathcal{P}$. Clearly, we have $(\bigcap \mathrm{SM}(\mathrm{T})) \subseteq (\bigcap \mathrm{SM}(\mathcal{P})/_{\mathrm{T}})$ and therefore the result follows.
2. We have to show that if q is in some stable models of $\mathcal{P}$, then it is also in some stable model of T. Symmetrically, $(\bigcup \mathrm{SM}(\mathcal{P})/_{\mathrm{T}}) \subseteq (\bigcup \mathrm{SM}(\mathrm{T}))$ and therefore the result follows.

The equivalence result then follows directly from Thm. 1, since in this case both $(\bigcap \mathrm{SM}(\mathrm{T})) = (\bigcap \mathrm{SM}(\mathcal{P})/_{\mathrm{T}})$ and $(\bigcup \mathrm{SM}(\mathcal{P})/_{\mathrm{T}}) \subseteq (\bigcup \mathrm{SM}(\mathrm{T}))$ hold. □

4 Magic Set Method for Datalog$^\neg$ Programs

In this section we present the Magic Set algorithm for general non-ground Datalog$^\neg$ programs (MS$^\neg$ algorithm for short). After recalling the Magic Set algorithm for positive Datalog queries, we discuss the key issues arising when dealing with Datalog$^\neg$ programs with unstratified negation. We then present the resulting MS$^\neg$ method, and finally we show some query equivalence results.

4.1 Datalog Programs

We will illustrate how the Magic-Set method simulates the top-down evaluation of a query by considering the program consisting of the rules $\texttt{path}(\texttt{X},\texttt{Y})$:– $\texttt{edge}(\texttt{X},\texttt{Y})$. and $\texttt{path}(\texttt{X},\texttt{Y})$:– $\texttt{edge}(\texttt{X},\texttt{Z}),\ \texttt{path}(\texttt{Z},\texttt{Y})$. together with query $\texttt{path}(1,5)$?.

Adornment Step: The key idea is to materialize, by suitable *adornments*, binding information for IDB predicates which would be propagated during a top-down computation. These are strings of the letters b and f, denoting bound or free for each argument of an IDB predicate. First, adornments are created for query predicates. The adorned version of the query above is $\texttt{path}^{\texttt{bb}}(1,5)$.

The query adornments are then used to propagate their information into the body of the rules defining it, simulating a top-down evaluation. Obviously various strategies can be pursued concerning the order of processing the body atoms and the propagation of bindings. These are referred to as Sideways Information Passing Strategies (*SIPS*), cf. [19]. Any SIPS must guarantee an iterative processing of all body atoms in r. Let q be an atom that has not yet been processed, and v be the set of already considered atoms, then a SIPS specifies a propagation $\texttt{v} \rightarrow_\chi \texttt{q}$, where χ is the set of the variables bound by v, passing their values to q. In this paper we consider the SIPS which propagates binding only through EDB atoms; IDB atoms *receive* the bindings, but do not bound any further variable.

In the first rule of the example ($\texttt{path}(\texttt{X},\texttt{Y})$:– $\texttt{edge}(\texttt{X},\texttt{Y})$.) a binding is only passed to the EDB predicate edge (which is not adorned), yielding the adorned rule $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y})$:– $\texttt{edge}(\texttt{X},\texttt{Y})$. In the second rule, $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y})$ passes its binding information to $\texttt{edge}(\texttt{X},\texttt{Z})$ by $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \rightarrow_{\{\texttt{X}\}} \texttt{edge}(\texttt{X},\texttt{Z})$. $\texttt{edge}(\texttt{X},\texttt{Z})$ itself is not adorned, but it gives a binding to Z. Then, we consider $\texttt{path}(\texttt{Z},\texttt{Y})$, for which we obtain the propagation $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y}),\ \texttt{edge}(\texttt{X},\texttt{Z}) \rightarrow_{\{\texttt{Y},\texttt{Z}\}} \texttt{path}(\texttt{Z},\texttt{Y})$. This causes the generation of the adorned atom $\texttt{path}^{\texttt{bb}}(\texttt{Z},\texttt{Y})$, and the resulting adorned rule is $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y})$:– $\texttt{edge}(\texttt{X},\texttt{Z}),\ \texttt{path}^{\texttt{bb}}(\texttt{Z},\texttt{Y})$.

In general, adorning a rule may generate new adorned predicates. This step is repeated until all adorned predicates have been processed, yielding the *adorned program*, in our example it consists of the rules $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y})$:– $\texttt{edge}(\texttt{X},\texttt{Y})$. and $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y})$:– $\texttt{edge}(\texttt{X},\texttt{Z}),\ \texttt{path}^{\texttt{bb}}(\texttt{Z},\texttt{Y})$.

Generation Step: The adorned program is used to generate *magic rules*, which simulate the top-down evaluation scheme. Let the *magic version* ***magic***$(\texttt{p}^\alpha)$ for an adorned atom $\texttt{p}^\alpha$ be defined as $\texttt{magic\_p}^\alpha$ in which all arguments labelled f in α are eliminated.

Then, for each adorned atom p in the body of an adorned rule r_a, a magic rule r_m is generated such that (i) the head of r_m consists of ***magic***(p), and (ii) the body of r_m consists of the magic version of the head atom of r_a, followed by all of the

predicates of r_a which can propagate the binding on p. In our example we generate $\texttt{magic\_path}^{\texttt{bb}}(\texttt{Z},\texttt{Y}) \texttt{ :- } \texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y}), \texttt{edge}(\texttt{X},\texttt{Z}).$

Modification Step: The adorned rules are subsequently modified by including magic atoms generated in Step 2 in the rule bodies. The resulting rules are called *modified rules*. For each adorned rule the head of which is h, we extend its rule body by inserting ***magic***(h) and by stripping off the adornments of the other predicates[3]. In our example, $\texttt{path}(\texttt{X},\texttt{Y}) \texttt{ :- } \texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y}),\ \texttt{edge}(\texttt{X},\texttt{Y}).$ and $\texttt{path}(\texttt{X},\texttt{Y}) \texttt{ :- } \texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y}),\ \texttt{edge}(\texttt{X},\texttt{Z}),\ \texttt{path}(\texttt{Z},\texttt{Y}).$ are generated.

Processing of the Query: For each adorned atom $\texttt{g}^{\alpha}$ of the query the *magic seed* ***magic***$(\texttt{g}^{\alpha})$. is asserted. In our example we generate $\texttt{magic\_path}^{\texttt{bb}}(1,5)$.

The complete rewritten program consists of the magic, modified, and query rules. Given a Datalog program $\mathcal{P}$, a query $\mathcal{Q}$, and the rewritten program $\mathcal{P}'$, it is well known (see e.g. [1]) that $\mathcal{P}$ and $\mathcal{P}'$ are equivalent w.r.t. $\mathcal{Q}$, i.e., $\mathcal{P} \equiv^{b}_{\mathcal{Q}} \mathcal{P}'$ and $\mathcal{P} \equiv^{c}_{\mathcal{Q}} \mathcal{P}'$ hold (since brave and cautious semantics coincide for Datalog programs).

4.2 Binding Propagation in Datalog$^{\neg}$ Programs: Some Key Issues

As argued in Sect. 3, different to positive Datalog, in which bindings are propagated only head-to-body in a rule, the problem with unstratified negation is that any rewriting for Datalog$^{\neg}$ programs, has to propagate bindings also body-to-head in general, in order to achieve query equivalence.

Example 2. Consider the program $\mathcal{P}_2$

```
z(X) :- y(X),not z(X).          y(X) :- q(X,Y).
p(X,Y) :- d(X,Y),not q(X,Y).    q(X,Y) :- d(X,Y),not p(X,Y).
a(X) :- p(X,Y),not b(X).        b(X) :- p(X,Y), not a(X).
```

together with the query $\mathcal{Q}_2 = p(a,X)?$, and the set of facts $F_2 = \{\texttt{d}(\texttt{a},\texttt{b})\}$. The stable models of $\mathcal{P}_2$ are $\{\texttt{p}(\texttt{a},\texttt{b}), \texttt{a}(\texttt{a}), \texttt{d}(\texttt{a},\texttt{b})\}$ and $\{\texttt{p}(\texttt{a},\texttt{b}), \texttt{b}(\texttt{a}), \texttt{d}(\texttt{a},\texttt{b})\}$, so we get $Ans_c(\mathcal{Q}_2, \mathcal{P}_{2,F_2}) = Ans_b(\mathcal{Q}_2, \mathcal{P}_{2,F_2}) = \{\{\texttt{X/b}\}\}$. Note that $\texttt{q}(\texttt{a},\texttt{b})$ cannot occur in any stable model.
When applying the Magic Set technique,[4] we obtain as adorned program:

```
p^bf(X,Y) :- d(X,Y),not q^bb(X,Y).    q^bb(X,Y) :- d(X,Y),not p^bb(X,Y).
p^bb(X,Y) :- d(X,Y),not q^bb(X,Y).
```

Then, the generation step produces the following magic program $Magic(\mathcal{Q}_2, \mathcal{P}_2)$:

```
magic_p^bf(a).                      magic_q^bb(X,Y) :- magic_p^bf(X),d(X,Y).
magic_p^bb(X,Y) :- magic_q^bb(X,Y). magic_q^bb(X,Y) :- magic_p^bb(X,Y).
```

Finally, the original rules are modified to $Modified(\mathcal{Q}_2, \mathcal{P}_2)$:

```
p(X,Y) :- magic_p^bf(X),d(X,Y),not q(X,Y).    q(X,Y) :- magic_q^bb(X,Y),d(X,Y),not p(X,Y).
p(X,Y) :- magic_p^bb(X,Y),d(X,Y),not q(X,Y).
```

[3] We do this only for facilitating the equivalence proofs, one can alternatively adorn the query.
[4] We do not consider any special technique for negative literals. We adorn negative literals last, since they receive bindings, but do not bind any further variables.

Input: A Datalog$^\neg$ program $\mathcal{P}$, and a query $\mathcal{Q} = \mathtt{g}(\overline{\mathtt{t}})$.
Output: The optimized program $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$.
var S: **stack** of adorned predicates; $modifiedRules$,$magicRules$: **set** of rules;
begin
1. $modifiedRules$:= $\emptyset$; $magicRules$:=***BuildQuerySeeds***$(\mathcal{Q}, S)$;
2. **while** $S \neq \emptyset$ **do**
3. p^α := S.**pop**();
4. **for each** rule $r \in \mathcal{P}$ with $H(r) = p(\overline{t}_p)$ **do**
5. r_a := ***Adorn***(r,p^α,S);
6. $magicRules$:= $magicRules \bigcup$ ***Generate***(r_a);
7. $modifiedRules$:= $modifiedRules \bigcup \{$***Modify***$(r_a)\}$;
8. **end for**
9. **for each** dangerous rule $d \in \mathcal{P}$ of the form $\mathtt{h}(\overline{\mathtt{t}}_\mathtt{h}) :- \mathtt{q}_1(\overline{\mathtt{t}}_1), \ldots, \mathtt{q}_\mathtt{m}(\overline{\mathtt{t}}_\mathtt{m})$ where $q_i = p$ **do**
10. **let** d_s be the rule $\mathtt{q}_\mathtt{i}(\overline{\mathtt{t}}_\mathtt{i}) :- \mathtt{h}(\overline{\mathtt{t}}_\mathtt{h}), \mathtt{q}_1(\overline{\mathtt{t}}_1), \ldots, \mathtt{q}_{\mathtt{i}-1}(\overline{\mathtt{t}}_1), \mathtt{q}_{\mathtt{i}+1}(\overline{\mathtt{t}}_1), \ldots, \mathtt{q}_\mathtt{m}(\overline{\mathtt{t}}_\mathtt{m})$;
11. **let** d_a:=***Adorn***$(d_s,\mathtt{p}^\alpha,S)$;
12. $magicRules$:= $magicRules \bigcup$ ***Generate***(d_a);
13. **end for**
14. **end while**
15. $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$:=$magicRules \cup modifiedRules$;
16. **return** $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$;
end.

Fig. 1. Magic Set Algorithm

Together with the fact $\mathtt{d}(\mathtt{a},\mathtt{b})$, $\mathtt{MS}(\mathcal{P}_2) = Magic(\mathcal{Q}_2, \mathcal{P}_2) \cup Modified(\mathcal{Q}_2, \mathcal{P}_2)$ admits two stable models, say M_1 and M_2, such that $M_1/_{\mathcal{P}_2} = \{\mathtt{p}(\mathtt{a},\mathtt{b})\}$ and $M_2/_{\mathcal{P}_2} = \{\mathtt{q}(\mathtt{a},\mathtt{b})\}$. Therefore, $Ans_c(\mathcal{Q}_2, MS(\mathcal{P}_{2,F_2})) = \emptyset$, and $Ans_b(\mathcal{Q}_2, MS(\mathcal{P}_{2,F_2})) = \{\{\mathtt{X}/\mathtt{b}\}\}$. Hence, $\mathtt{MS}(\mathcal{P}_2)$ is not cautious-complete w.r.t. $\mathcal{P}_2$. □

In general the application of the traditional Magic Set method on unstratified programs would guarantee cautious-soundness and brave-completeness, but it would not ensure cautious-completeness and brave-soundness.

The reason for this semantic problem lies in the fact that the first rule of $\mathcal{P}_2$ acts as a constraint imposing any atom of the form $\mathtt{y}(\mathtt{X})$ to be not entailed in any model. Then, from the second rule we also conclude that we cannot derive any fact of the form $\mathtt{q}(\mathtt{X},\mathtt{Y})$. It follows that the constraint "indirectly" influences the query on predicate $\mathtt{p}$, since the model M_2 of the rewritten program such that $M_2/_{\mathcal{P}_2} = \{\mathtt{q}(\mathtt{a},\mathtt{b})\}$ cannot be extended to be a model for program $\mathcal{P}_2$.

In order to overcome this semantic problem, we next present a Magic Set rewriting which deals correctly with dangerous rules. In the above example, our method recognizes that the second rule is *dangerous* and propagates the binding coming from $\mathtt{q}$ (in the body) to $\mathtt{y}$ (in the head).

4.3 $\mathtt{MS}^\neg$ Algorithm

We next describe the peculiarities of our rewriting technique. We assume the existence of four auxiliary functions: ***BuildQuerySeeds***$(\mathcal{Q}, S)$ adorns the given query $\mathcal{Q}$, creates an appropriate fact, and pushes newly adorned predicates onto the stack S, which is a variable parameter. ***Adorn***(r,p^α,S) adorns the rule r using p^α and pushes new adorned predicates onto S. ***Generate***(r_a) creates the magic rules for the adorned rule r_a, and ***Modify***(r_a) creates the modified rule for r_a. These functions implement what was informally described in Sect. 4.1 for the Magic Set method. In particular, we assume that these functions implement the *basic* Magic Set method, propagating bindings only

through EDB predicates [22, 1] (as stated above, we do not consider any special technique for negative literals, which are simply adorned last in the rule).

The algorithm $\texttt{MS}^\neg$, reported in Fig. 1, implements the Magic Set method for Datalog$^\neg$ programs. Its input is a Datalog$^\neg$ program $\mathcal{P}$ and a query $\mathcal{Q}$. (Note that the algorithm can be used for positive rules as a special case.) If the query contains some constants, $\texttt{MS}^\neg$ outputs a (optimized) program $\texttt{MS}^\neg(\mathcal{Q}, \mathcal{P})$ consisting of a set of *modified* and *magic* rules (denoted by $modifiedRules$and $magicRules$, respectively). The algorithm generates modified and magic rules on a rule-by-rule basis. To this end, it exploits a stack S of predicates for storing all the adorned predicates that are still to be used for propagating the query binding (the ***Adorn*** function pushes on S each adorned predicates it generates, which has not been previously rewritten). At each step, an element p^α is removed from S, and the rules defining p are processed one-at-a-time.

The main steps of the algorithm $\texttt{MS}^\neg$ are illustrated by means of the program $\mathcal{P}_2$ in Example 2, and the query $\mathcal{Q}_2 = \texttt{p}(\texttt{a}, \texttt{X})$.

The computation starts in step *2* by initializing $modifiedRules$ to the empty set. Then, the function ***BuildQuerySeeds*** is used for storing in $magicRules$ the magic seeds, and pushing on the stack S the adorned predicates of $\mathcal{Q}$. For instance, given the query $\mathcal{Q}_2$ and the program $\mathcal{P}_2$, ***BuildQuerySeeds*** creates $\texttt{magic\_p}^{\texttt{bf}}(\texttt{a})$. and pushes $\texttt{p}^{\texttt{bf}}$ onto the stack S.

The core of the technique (steps *2-13*) is repeated until the stack S is empty, i.e., until there is no further adorned predicate to be propagated. Specifically, an adorned predicate $\texttt{p}^\alpha$ is removed from the stack S in step *3*, and its binding is propagated.

In the steps *4-8*, the binding of $\texttt{p}^\alpha$ is propagated in a traditional way, to each rule r of $\mathcal{P}$ having an atom $\texttt{p}(\overline{\texttt{t}})$ in the head. This propagation is as in the standard Magic Set method for stratified Datalog$^\neg$ programs.

Example 3. Consider again Example 2. Taking the predicate $\texttt{p}^{\texttt{bf}}$ from the stack entails the adornment of the rule $\texttt{p}(\texttt{X},\texttt{Y})$:– $\texttt{d}(\texttt{X},\texttt{Y}), \text{not } \texttt{q}(\texttt{X},\texttt{Y})$.. This yields the rule $\texttt{p}^{\texttt{bf}}(\texttt{X},\texttt{Y})$:– $\texttt{d}(\texttt{X},\texttt{Y}), \text{not } \texttt{q}^{\texttt{bb}}(\texttt{X},\texttt{Y})$., and the predicate $\texttt{q}^{\texttt{bb}}$ is eventually pushed on the stack. Then, we can proceed (by using the standard algorithms) with the generation of one magic ($\texttt{magic\_q}^{\texttt{bb}}(\texttt{X},\texttt{Y})$:– $\texttt{magic\_p}^{\texttt{bf}}(\texttt{X}), \texttt{d}(\texttt{X},\texttt{Y})$.) and one modified rule ($\texttt{p}(\texttt{X},\texttt{Y})$:– $\texttt{magic\_p}^{\texttt{bf}}, \texttt{d}(\texttt{X},\texttt{Y}), \text{not } \texttt{q}(\texttt{X},\texttt{Y})$.). □

Steps *9-13* performs the propagation of the binding through each *dangerous* rule d in $\mathcal{P}$ of the form $\texttt{h}(\overline{\texttt{t}}_\texttt{h})$:– $\texttt{p}(\overline{\texttt{t}}_\texttt{p}), \texttt{q}_1(\overline{\texttt{t}}_1), \ldots, \texttt{q}_\texttt{m}(\overline{\texttt{t}}_\texttt{m})$., having an atom $\texttt{p}(\overline{\texttt{t}}_\texttt{p})$ in the body. These steps are, in fact, required for avoiding the semantic problems that we have described in the previous section. In this case, in order to simulate the body-to-head propagation, the rule d is first replaced by an "inverted" rule d_s of the form $\texttt{p}(\overline{\texttt{t}}_\texttt{p})$:– $\texttt{h}(\overline{\texttt{t}}_\texttt{h}), \texttt{q}_1(\overline{\texttt{t}}_1), \ldots, \texttt{q}_\texttt{m}(\overline{\texttt{t}}_\texttt{m})$., which has been obtained by swapping the head predicate with the body predicate propagating the binding. Then, the adornment can be carried out as usual by means of the function ***Adorn***. Since this "inverted" rule was not part of the original program and its only purpose is generating binding information, it will not give rise to a modified rule, but only to magic rules.

Example 4. When $\texttt{q}^{\texttt{bb}}$ is removed from the stack, it can be used for adorning the body of the dangerous rule $\texttt{y}(\texttt{X})$:– $\texttt{q}(\texttt{X},\texttt{Y})$. Hence, we obtain first the "inverted" rule

$\mathtt{q(X,Y)}$:– $\mathtt{y(X)}$. and adorn it, obtaining $\mathtt{q^{bb}(X,Y)}$:– $\mathtt{y^{b}(X)}$. which gives rise to one magic rule: $\mathtt{magic\_y^{b}(X)}$:– $\mathtt{magic\_q^{bb}(X,Y)}$. □

Finally, after all the adorned predicates have been processed the algorithm outputs the program $\mathtt{MS}^{\neg}(\mathcal{Q},\mathcal{P})$.

Example 5. The complete rewriting of program $\mathcal{P}_2$ w.r.t. query $\mathcal{Q}_2$ ($\mathtt{MS}^{\neg}(\mathcal{Q}_2,\mathcal{P}_2)$) consists of the magic rules:

$\mathtt{magic\_p^{bf}(a)}$.
$\mathtt{magic\_p^{bb}(X,Y)}$:– $\mathtt{magic\_q^{bb}(X,Y)}$.
$\mathtt{magic\_q^{bf}(X)}$:– $\mathtt{magic\_y^{b}(X)}$.
$\mathtt{magic\_z^{b}(X)}$:– $\mathtt{magic\_y^{b}(X)}$,$\mathtt{z(X)}$.
$\mathtt{magic\_q^{bb}(X,Y)}$:– $\mathtt{magic\_p^{bb}(X,Y)}$.
$\mathtt{magic\_q^{bb}(X,Y)}$:– $\mathtt{magic\_p^{bf}(X)}$,$\mathtt{d(X,Y)}$.
$\mathtt{magic\_y^{b}(X)}$:– $\mathtt{magic\_q^{bb}(X,Y)}$.
$\mathtt{magic\_z^{b}(X)}$:– $\mathtt{magic\_y^{b}(X)}$.
$\mathtt{magic\_p^{bb}(X,Y)}$:– $\mathtt{magic\_q^{bf}(X)}$,$\mathtt{d(X,Y)}$.

plus the rewritten rules:

$\mathtt{p(X,Y)}$:– $\mathtt{magic\_p^{bf}(X)}$,$\mathtt{d(X,Y)}$,not $\mathtt{q(X,Y)}$.
$\mathtt{y(X)}$:– $\mathtt{magic\_y^{b}(X)}$,$\mathtt{q(X,Y)}$.
$\mathtt{q(X,Y)}$:– $\mathtt{magic\_q^{bf}(X)}$,$\mathtt{d(X,Y)}$,not $\mathtt{p(X,Y)}$.
$\mathtt{q(X,Y)}$:– $\mathtt{magic\_q^{bb}(X,Y)}$,$\mathtt{d(X,Y)}$,not $\mathtt{p(X,Y)}$.
$\mathtt{z(X)}$:– $\mathtt{magic\_z^{b}(X)}$,$\mathtt{y(X)}$,not $\mathtt{z(X)}$.
$\mathtt{p(X,Y)}$:– $\mathtt{magic\_p^{bb}(X,Y)}$,$\mathtt{d(X,Y)}$,not $\mathtt{q(X,Y)}$.

It is worth noting that the rewritten program does not contain rules for predicates a and b, since they are not relevant for answering $\mathcal{Q}_2$. $\mathtt{MS}^{\neg}(\mathcal{Q}_2,\mathcal{P}_2)$ admits only one stable model M, such that $M/_{\mathcal{P}_2} = \{\mathtt{p(a,b)}\}$. Hence, $Ans_c(\mathcal{Q}_2,\mathcal{P}_{2,F_2}) = Ans_b(\mathcal{Q}_2,\mathcal{P}_{2,F_2}) = \mathtt{X/b}$.the original semantics is preserved. □

4.4 Query Equivalence Results

We conclude the presentation of the MS algorithm by formally proving its soundness. The result is shown by establishing correspondences between a program $\mathcal{P}$ and its transformed program $\mathtt{MS}^{\neg}(\mathcal{Q},\mathcal{P},)$ with respect to some query $\mathcal{Q}$.

To show this result, we will employ the notion of *simplification*: Given a ground program $\mathcal{P}$ and a subprogram $U \subseteq \mathcal{P}$, which admits exactly one stable model S. Then $\mathrm{simplify}(\mathcal{P},U)$ denotes the program $\{r/_{\mathcal{P}-U} \mid r \in (\mathcal{P}-U) \wedge B^{+}(r)/_{U} \subseteq S \wedge B^{-}(r)/_{U} \cap S = \emptyset\}$, which can be thought of as the partial evaluation w.r.t. S. This is needed to get rid of the magic predicates, which are not present in the original program.

Lemma 1. *Let $\mathcal{P}$ be a Datalog$^{\neg}$ program $\mathcal{P}$, $\mathcal{Q}$ a query. Furthermore, we denote by* $\mathrm{magic}(\mathcal{Q},\mathcal{P})$ *the set of magic rules in* $\mathtt{MS}^{\neg}(\mathcal{Q},\mathcal{P})$. *Then it holds that* $\mathcal{P}'' = \mathrm{simplify}(Ground(\mathtt{MS}^{\neg}(\mathcal{Q},\mathcal{P})),\mathrm{magic}(\mathcal{Q},\mathcal{P}) \cup EDB(\mathcal{P}))$ *is a module of* $\mathcal{P}' = \mathrm{simplify}(Ground(\mathcal{P}),EDB(\mathcal{P}))$.

Proof (Sketch). Observe that $\mathcal{P}'' \subseteq \mathcal{P}'$ holds. Assume that $\mathcal{P}''$ is not a module of $\mathcal{P}'$. Then at least one of the following condition holds: (1) $\exists r' \in \mathcal{P}' - \mathcal{P}'', r'' \in \mathcal{P}'' : H(r') = H(r'')$ (2) $\exists r'' \in \mathcal{P}'' : \exists b \in B(r'') : \exists r' \in \mathcal{P}' - \mathcal{P}'' : b = H(r')$ (3) $\exists r'' \in \mathcal{P}'' : \exists r' \in \mathcal{P}' - \mathcal{P}'' : H(r'') \in B(r')$ and r' is dangerous. One can show that all of (1), (2), and (3) lead to contradictions, and hence the result follows.

(1) For all rules in $\mathcal{P}$ with head predicate h, in $\mathtt{MS}^{\neg}(\mathcal{Q},\mathcal{P})$ there exists a copy for each adornment that was generated for h. So for any simplified ground instance r of

such a rule with head atom $h(c_1, \ldots, c_n)$, either $magic\_h^a(c_1, ..., c_m)$ holds for at least one adornment a of h, or it does not hold for any adornment of h. In the former case, for each rule in $\mathcal{P}$ with h in its head, a corresponding rule with $magic\_h^a$ in its body exists in $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$. If $magic\_h^a(c1, ..., cm)$ holds for no adornment a, no simplified ground version of r is in $\mathcal{P}''$. In total, for each ground atom $h(c1, ..., cn)$ in $\mathcal{P}'$, either all or none of its defining rules are in $\mathcal{P}''$.

(2) Assume that r'' (the head of which is $h(c_1, \ldots, c_h)$) stems from a rule r''_o, which was adorned by a, such that $magic\_h^a(c_1, \ldots, c_{h_1})$ follows from the magic rules. Each IDB body atom of r''_o has received some adornment based on a, in which bound arguments either directly share bound variables (w.r.t. a) with h, or via some EDB atoms. For any body predicate b, this gives rise to a magic rule r_m : $magic\_b^{a_1}(\overline{t}_{b_1})$:– $magic\_h^a(\overline{t}_a), B$. where B contains in particular all EDB atoms relevant for bound arguments of b. Concerning r' it contains some $b(d_1, \ldots, d_b)$ of the body of r'' in its head, and its originating rule r'_o is adorned by a_1. So in $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$ a rule r'_m : $b(\overline{t}_{b_2})$:– $magic\_b^{a_1}(\overline{t}_{b_3}), B'$. occurs. Note that for all bound arguments $d_1, \ldots, d_k$ of $b(d_1, \ldots, d_b)$ w.r.t. a_1, $magic\_b^{a_1}(d_1, \ldots, d_k)$ follows from the magic rules because of r_m. So whenever a simplified ground instance of r'_o with $b(d_1, \ldots, d_b)$ in the head exists, so does one of r'_m, which is hence in $\mathcal{P}''$.

(3) Observe first that the set of instantiations of dangerous rules in $\mathcal{P}$ is a superset of the set of dangerous rules in $Ground(\mathcal{P})$, which is in turn a superset the set of dangerous rules in any simplification of $Ground(\mathcal{P})$. So any dangerous rule in $\mathcal{P}'$ is also dangerous in $\mathcal{P}$. Therefore, the originating rule $r'_o \in \mathcal{P}$ of r' must have been adorned and "inverted", adorning in the following also the head of r'_o. So the dangerous rule r'_o eventually also gives rise to a modified rule in $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$. Then, by the same argument as in (2), a magic rule obtained from the "inverted" rule must exist in $\mathtt{MS}^\neg(\mathcal{Q}, \mathcal{P})$, such that one of its instantiations matches the bound arguments of $H(r')$. So r' is in $\mathcal{P}''$ iff it is in $\mathcal{P}'$. □

Theorem 3. *Let $\mathcal{P}$ be a Datalog$^\neg$ program, let $\mathcal{Q}$ be a query. Then, it holds that (1) $\mathtt{MS}^\neg(\langle\mathcal{Q}, \mathcal{P}\rangle) \subseteq^c_{\mathcal{Q}} \mathcal{P}$ and $\mathtt{MS}^\neg(\langle\mathcal{Q}, \mathcal{P}\rangle) \supseteq^b_{\mathcal{Q}} \mathcal{P}$, and (2) if $\mathrm{SM}(\mathcal{P}) \neq \emptyset$, $\mathtt{MS}^\neg(\langle\mathcal{Q}, \mathcal{P}\rangle) \equiv^b_{\mathcal{Q}} \mathcal{P}$ and $\mathtt{MS}^\neg(\langle\mathcal{Q}, \mathcal{P}\rangle) \equiv^c_{\mathcal{Q}} \mathcal{P}$.*

5 An Application to Data Integration

In this section we show an application of the Magic Set method for optimizing query answering in data integration systems, and report on the experience we are doing in the EU project INFOMIX on data integration. Let us first recall some basic notions.

A data integration system $\mathcal{I}$ is a triple $\langle\mathcal{G}, \mathcal{S}, \mathcal{M}\rangle$, where $\mathcal{G}$ is the global (relational) schema of the form $\mathcal{G} = \langle\Psi, \Sigma\rangle$, $\mathcal{S}$ is the source (relational) schema of the form $\mathcal{S} = \langle\Psi', \emptyset\rangle$, i.e., there are no integrity constraints on the sources, and $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$.

Example 6. Consider the data integration system $\mathcal{I}_0 = \langle\mathcal{G}_0, \mathcal{S}_0, \mathcal{M}_0\rangle$, a simplification of the Demo Scenario in the EU project INFOMIX described below. The global schema $\mathcal{G}_0$ consists of the relations $professor(IDP, Pname, Phomepage)$,

$student(IDS, Sname, Saddress)$, $exam\_data(IDP, IDS, Exam, Mark)$. The associated constraints in Σ_0 state that: (i) (*key constraints*) the keys of $professor$, $student$, and $exam\_data$ are the attributes IDP, IDS, and $(IDP, IDS, Exam)$, respectively, (ii) (*exclusion dependency*) a professor cannot be a student, and (iii) (*inclusion dependencies*) the identifiers of professors and students in the relation $exam\_data$ must be in the relations $professor$ and $student$, respectively. The source schema $\mathcal{S}_0$ comprises the relations s_1, s_2, s_3, and s_4. Finally, the mapping $\mathcal{M}_0$ is defined by the datalog program formed by $professor(X, Y, Z)$:- $s_1(X, Y, Z)$., $professor(X, Y, Z)$:- $s_4(Z, Y, , X)$., $student(X, Y, Z)$:- $s_2(Y, X, Z)$., $exam\_data(X, Y, Z, W)$:- $s_3(Y, X, Z, W)$. □

Given a database $\mathcal{D}$ for the source schema $\mathcal{S}$, the user might issue a query q on the global schema which is populated by retrieving the data from $\mathcal{D}$ according to the mapping $\mathcal{M}$. However, while carrying out such an integration, it often happens that the retrieved (global) database, denoted by $ret(\mathcal{I}, \mathcal{D})$, is inconsistent w.r.t. Σ since data stored in local and autonomous sources are not in general required to satisfy constraints expressed on the global schema.

To remedy this problem, several approaches (see, e.g., [10–15, 20]) defined the semantics of a data integration system $\mathcal{I}$ in terms of the repairs $rep(\mathcal{I}, \mathcal{D})$ of the database $ret(\mathcal{I}, \mathcal{D})$. Intuitively, each repair $\mathcal{R} \in rep(\mathcal{I}, \mathcal{D})$ is obtained by properly adding and deleting facts from $ret(\mathcal{I}, \mathcal{D})$ in order to satisfy constraints in Σ, as long as we "minimize" such additions and deletions.

These repairs depend on the interpretation of the mappings in $\mathcal{M}$, which, in fact, impose restrictions or preferences on the possibility of adding or removing facts from $ret(\mathcal{I}, \mathcal{D})$ to repair constraint violations. In the INFOMIX project, we have considered the *loosely-sound* semantics according to which mappings might retrieve only a subset of the tuples needed for answering the query. Hence, we can add an unbounded number of tuples to repair violations of inclusion dependencies; nonetheless, the semantics is loose in the sense that, in order to repair keys and exclusion dependencies, we are also allowed to delete a minimal set of tuples.

Given a data integration system $\mathcal{I}$ and a source database $\mathcal{D}$, a *query* q for $\mathcal{I}$ is an atom comprising a global relation in $\mathcal{I}$. Then, the answer to q is defined as the set $ans(q, \mathcal{I}, \mathcal{D})$ of all substitutions ϑ, such that, for each $1 \leq i \leq k$, $b_i\vartheta$ is true in each repair $\mathcal{R}$ in $rep(\mathcal{I}, \mathcal{D})$.

In order to design effective systems for query answering in data integration settings, the repair semantics has been formalized in the INFOMIX project (as well as in other approaches) by using logic programs, i.e., by encoding the constraints Σ of $\mathcal{G}$ and the mapping assertions $\mathcal{M}$ into a logic program, $\Pi(\mathcal{I}, \mathcal{D})$, using unstratified negation, such that the stable models of this program yield the repairs of the global database. The correctness of the rewriting is shown by the following theorem.

Theorem 4 ([13]). *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, $\mathcal{D}$ be a database for $\mathcal{S}$, and q be a query over $\mathcal{G}$. Then, $ans(q, \mathcal{I}, \mathcal{D})$ coincides with $Ans_c(q, \Pi(\mathcal{I}, \mathcal{D}))$.*

An attractive feature of this approach is that logic programs serve as executable logical specifications of repairs, and thus allow to state repair policies in a declarative rather than a procedural manner. However, a drawback of this approach is that with current implementations of stable model engines, such as DLV or Smodels, the evaluation

of queries over large data sets quickly becomes infeasible, which calls for suitable optimization methods that help in speeding up the evaluation of queries expressed as logic programs [14].

To this aim, the binding propagation techniques proposed in this paper can be profitably exploited to isolate the relevant part of a database by "pushing down" the query constants to the sources. Importantly, our optimization fully preserves the original semantics of the data-integration query. Indeed, the loosely-sound semantics for data integration always guarantees the existence of a database repair no matter of the types of constraints in Σ, provided that the schema is *non-key-conflicting* [21]. Consequently, $\Pi(\mathcal{I}, \mathcal{D})$ is guaranteed to be consistent, and the correctness of the application of the Magic Set technique follows immediately from Thm. 3.

Theorem 5. *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, $\mathcal{D}$ be a database for $\mathcal{S}$, and q be a query over $\mathcal{G}$. Then, $ans(q, \mathcal{I}, \mathcal{D})$ coincides with $Ans_c(q, \mathtt{MS}^{\neg}(q, \Pi(\mathcal{I}, \mathcal{D})))$.*

In order to test the effectiveness of the Magic Set technique for query optimization in data integration systems, we have carried out some experiments on the demonstration scenario of the INFOMIX project, which refers to the information system of the University "La Sapienza" in Rome. The global schema consists of 14 global relations with 29 constraints, while the source schema includes 29 relations (in 3 legacy databases) and 12 web wrappers (generating relational data) for more than 24MB of data.

Once a query q on $\mathcal{I}$ is submitted, a number of wrappers are executed to retrieve the data from the relevant sources for q, storing it in a Postgres database. Then, the Datalog$^{\neg}$ system DLV imports the Postgres data, and computes the answers for q w.r.t. $\Pi(\mathcal{I}, \mathcal{D})$. We measured the execution times of DLV for $\Pi(\mathcal{I}, \mathcal{D})$ and its magic-set rewritten version $\mathtt{MS}^{\neg}(q, \Pi(\mathcal{I}, \mathcal{D}))$. Several experiments confirmed that on various practical queries, the performance is greatly improved by Magic Sets (in some cases, the query evaluation time passes from more than 20 minutes to a few seconds), while in the other cases we have observed no or only a minor overheads.

We finally observe that similar arguments can be also used to prove that our magic set technique can be profitably exploited in other approaches to data integration such as [10–12, 14]. In fact, all these approaches reduce answering a user query, q, to cautious reasoning over a logic program $\Pi(\mathcal{I}, \mathcal{D})$ which is guaranteed to be consistent.

Some of these approaches actually use disjunctive datalog programs, possibly with unstratified negation. We point out that the algorithm of this paper can be coupled with the method in [30], which is defined on positive disjunctive programs, obtaining a magic set method for arbitrary disjunctive programs.

6 Related Work and Conclusions

The Magic-Set method [22, 19, 1, 23] is one of the best known techniques for the optimization of Datalog queries. Many extensions and refinements of Magic-Sets have been proposed, addressing e.g. query constraints [24], modular stratification and well-founded semantics [25, 26], integration into cost-based query optimization [27]. The research on enhancements to the Magic-Set method is still going on. For instance, in the last-year ACM-PODS conference a magic-set technique for the class of *soft-stratifiable*

programs was presented [28], and in [29, 30] magic sets techniques for disjunctive programs were proposed.

An extension of the Magic Set technique for *positive* Datalog programs with integrity constraints has been presented in [31]. The proposed method is shown to be *brave complete* and *cautious sound*. Comparing this method to our approach, we observe that: (1) Our method is more general than the method in [31], since the latter deals only with a strict subset of Datalog$^\neg$ (recall that an integrity constraint $\texttt{:-}\, C.$ is just a shorthand for $p \texttt{:-}\, C, \mathrm{not}\ p$); while our method supports full Datalog$^\neg$, allowing for unstratified negation. (2) Our method has much better semantic properties than [31]. Indeed, [31] do not ensure query equivalence in any case; while we guarantee full query equivalence, unless the input program is inconsistent (see Thm. 3). Such a query equivalence is in fact very relevant for data integration applications (see the previous Section).

Our modularity results are strictly related to splitting sets, as defined in [16], or equivalently to modules as defined in [17]. The main difference is that our notion of modules and independent sets guarantee query equivalence for consistent programs, which does not hold for these previous notions. In fact, in general, one can prove that only the first two items of Thm. 2 hold for splitting-set modules.

A different kind of query optimization for data integration has been done in [32]. This approach does not exploit constants that appear in the query, but only inconsistent (w.r.t. constraints of the global schema) portions of the retrieved database. In fact, no systematic technique for query optimization in data integration systems exploiting binding propagations has been proposed in the literature so far.

Concluding, we believe that our results are relevant to both theory and practice. On the theory side, our modularity results provide a better understanding of the structural properties of Datalog$^\neg$, complementing and advancing on previous works on modularity properties of this language. Moreover, the MS$^\neg$ algorithm generalizes Magic Sets, enlarging significantly their range of applicability to the full class of Datalog$^\neg$ programs under the stable model semantics. Importantly, our work can be profitably exploited for data-integration systems. Preliminary results of experiments show that the application of our techniques allows us to solve very advanced data-integration tasks.

References

1. Ullman, J.D.: Principles of Database and Knowledge Base Systems. Computer Science Press (1989)
2. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: ICLP/SLP'88, Cambridge, Mass., MIT Press (1988) 1070–1080
3. Bidoit, N., Froidevaux, C.: Negation by Default and Unstratifiable Logic Programs. Theoretical Computer Science **78** (1991) 85–112
4. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys **33** (2001) 374–425
5. Bidoit, N., Froidevaux, C.: General Logical Databases and Programs: Default Logic Semantics and Stratification. Information and Computation **91** (1991) 15–54
6. Marek, V.W., Truszczyński, M.: Autoepistemic Logic. JACM **38** (1991) 588–619
7. Schlipf, J.: The Expressive Powers of Logic Programming Semantics. JCSS **51** (1995) 64–86 Abstract in Proc. PODS 90, pp. 196–204.

8. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL (2004) To appear.
9. Niemelä, I., Simons, P., Syrjänen, T.: Smodels: A System for Answer Set Programming. In: NMR'2000 (2000)
10. Arenas, M., Bertossi, L.E., Chomicki, J.: Specifying and querying database repairs using logic programs with exceptions. In: Proc. of FQAS 2000, Springer (2000) 27–41
11. Greco, G., Greco, S., Zumpano, E.: A logic programming approach to the integration, repairing and querying of inconsistent databases. In: Proc. of ICLP'01, Springer (2001) 348–364
12. Barceló, P., Bertossi, L.: Repairing databases with annotated predicate logic. In: Proc. the 10th Int. Workshop on Non-Monotonic Reasoning (NMR 2002). (2002) 160–170
13. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: Proc. of IJCAI 2003. (2003) 16–21
14. Bravo, L., Bertossi, L.: Logic programming for consistently querying data integration systems. In: Proc. of IJCAI 2003. (2003) 10–15
15. Chomicki, J., Marcinkowski, J.: Minimal-Change Integrity Maintenance Using Tuple Deletions. Information and Computation (2004) to Appear.
16. Lifschitz, V., Turner, H.: Splitting a Logic Program. In Van Hentenryck, P., ed.: ICLP'94, MIT Press (1994) 23–37
17. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM TODS **22** (1997) 364–418
18. Dung, P.M.: On the Relations between Stable and Well-Founded Semantics of Logic Programs. Theoretical Computer Science **105** (1992) 7–25
19. Beeri, C., Ramakrishnan, R.: On the power of magic. JLP **10** (1991) 255–259
20. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: Data integration under integrity constraints. Information Systems **29** (2004) 147–163
21. Calì, A., Lembo, D., Rosati, R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In: PODS '03. (2003) 260–271
22. Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.D.: Magic Sets and Other Strange Ways to Implement Logic Programs. In: PODS'86. (1986) 1–16
23. Mumick, I.S., Finkelstein, S.J., Pirahesh, H., Ramakrishnan, R.: Magic is relevant. In: SIGMOD Conference 1990. (1990) 247–258
24. Stuckey, P.J., Sudarshan, S.: Compiling query constraints. In: PODS'94, ACM Press (1994) 56–67
25. Ross, K.A.: Modular Stratification and Magic Sets for Datalog Programs with Negation. JACM **41** (1994) 1216–1266
26. Kemp, D.B., Srivastava, D., Stuckey, P.J.: Bottom-up evaluation and query optimization of well-founded models. Theoretical Computer Science **146** (1995) 145–184
27. Seshadri, P., Hellerstein, J.M., Pirahesh, H., Leung, T.Y.C., Ramakrishnan, R., Srivastava, D., Stuckey, P.J., Sudarshan, S.: Cost-based optimization for magic: Algebra and implementation. In: SIGMOD Conference 1996, ACM Press (1996) 435–446
28. Behrend, A.: Soft stratification for magic set based query evaluation in deductive databases. In: PODS 2003, ACM Press (2003) 102–110
29. Greco, S.: Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. IEEE TKDE **15** (2003) 368–385
30. Cumbo, C., Faber, W., Greco, G.: Enhancing the magic-set method for disjunctive datalog programs. In: Proc. of ICLP'04, Saint-Malo, France (2004) Forthcoming.
31. Greco, G., Greco, S., Trubitsyna, I., Zumpano, E.: Optimization of Bound Disjunctive Queries with Constraints. TPLP (to appear (CoRR report cs.LO/0406013))
32. Eiter, T., Fink, M., Greco, G., Lembo, D.: : Efficient evaluation of logic programs for querying data integration systems. In: Proc. of ICLP'03. (2003) 163–177