Magic Sets for Disjunctive Datalog Programs

Mario Alviano Wolfgang Faber Gianluigi Greco Nicola Leone

Department of Mathematics, University of Calabria, 87036 Rende, Italy

Abstract

In this paper, a new technique for the optimization of (partially) bound queries over disjunctive Datalog programs with stratified negation is presented. The technique exploits the propagation of query bindings and extends the Magic Set optimization technique (originally defined for non-disjunctive programs).

An important feature of disjunctive Datalog programs is nonmonotonicity, which calls for nondeterministic implementations, such as backtracking search. A distinguishing characteristic of the new method is that the optimization can be exploited also during the nondeterministic phase. In particular, after some assumptions have been made during the computation, parts of the program may become irrelevant to a query under these assumptions. This allows for dynamic pruning of the search space. In contrast, the effect of the previously defined Magic Set methods for disjunctive Datalog is limited to the deterministic portion of the process. In this way, the potential performance gain by using the proposed method can be exponential, as could be observed empirically.

The correctness of the method is established and proved in a formal way thanks to a strong relationship between Magic Sets and unfounded sets that has not been studied in the literature before. This knowledge allows for extending the method and the correctness proof also to programs with stratified negation in a natural way.

The proposed method has been implemented in the DLV system and various experiments on synthetic as well as on real-world data have been conducted. The experimental results on synthetic data confirm the utility of Magic Sets for disjunctive Datalog, and they highlight the computational gain that may be obtained by the new method with respect to the previously proposed Magic Set method for disjunctive Datalog programs. Further experiments on data taken from a real-life application show the benefits of the Magic Set method within an application scenario that has received considerable attention in recent years, the problem of answering user queries over possibly inconsistent databases originating from integration of autonomous sources of information.

Key words: Logic Programming, Stable Models, Magic Sets, Answer Set Programming, Data Integration

1 Introduction

Disjunctive Datalog is a language that has been proposed for modeling incomplete data [48]. Together with a light version of negation, in this paper stratified negation, this language can in fact express any query of the complexity class Σ_2^P (i.e., $\mathrm{NP}^{\mathrm{NP}}$) [22], under the stable model semantics. It turns out that disjunctive Datalog with stratified negation is strictly more expressive (unless the polynomial hierarchy collapses to its first level) than normal logic programming (i.e., non-disjunctive Datalog with unstratified negation), as the latter can express "only" queries in NP. As shown in [22], the high expressive power of disjunctive Datalog has also some positive practical implications in terms of modelling knowledge, since many problems in NP can be represented more simply and naturally in stratified disjunctive Datalog than in normal logic programming. For this reason, it is not surprising that disjunctive Datalog has found several real-world applications [42,49,50,57,58], also encouraged by the availability of some efficient inference engines, such as DLV [43], GnT [37], Cmodels [46], or ClaspD [21]. As a matter of fact, these systems are continuously enhanced to support novel optimization strategies, enabling them to be effective over increasingly larger application domains. In this paper, we contribute to this development by providing a novel optimization technique, inspired by deductive database optimization techniques, in particular the Magic Set method [6,9,63].

The goal of the original Magic Set method (defined for non-disjunctive Datalog programs) is to exploit the presence of constants in a query for restricting the possible search space by considering only a subset of a hypothetical program instantiation that is sufficient to answer the query in question. In order to do this, a top-down computation for answering the query is simulated in an abstract way. This top-down simulation is then encoded by means of rules, defining new Magic Set predicates. The extensions of these predicates (sets of ground atoms) will contain the tuples that are calculated during a top-down computation. These predicates are inserted into the original program rules and can then be used by bottom-up computations to narrow the computation to what is needed for answering the query.

Extending these ideas to disjunctive Datalog faces a major challenge: While non-disjunctive Datalog programs are deterministic, which in terms of the stable model semantics means that any non-disjunctive Datalog program has

* Preliminary portions of this paper appeared in the proceedings of the 20th International Conference on Logic Programming (ICLP'04).

Email addresses: `alviano@mat.unical.it` (Mario Alviano), `faber@mat.unical.it` (Wolfgang Faber), `ggreco@mat.unical.it` (Gianluigi Greco), `leone@mat.unical.it` (Nicola Leone).

exactly one stable model, disjunctive Datalog programs are nondeterministic in the sense that they may have multiple stable models. Of course, the main goal is still isolating a subset of a hypothetical program instantiation, upon which the considered query will be evaluated in an equivalent way. There are two basic possibilities how this nondeterminism can be dealt with in the context of Magic Sets: The first is to consider *static* Magic Sets, in the sense that the definition of the Magic Sets is still deterministic, and therefore the extension of the Magic Set predicates is equal in each stable model. This static behavior is automatic for Magic Sets of non-disjunctive Datalog programs. The second possibility is to allow *dynamic* Magic Sets, which also introduce non-deterministic definitions of Magic Sets. This means that the extension of the Magic Set predicates may differ in various stable models, and thus can be viewed as being specialized for each stable model.

While the nature of dynamic Magic Sets intuitively seems to be more fitting for disjunctive Datalog than static Magic Sets, considering the architecture of modern reasoning systems for disjunctive Datalog substantiates this intuition: These systems work in two phases, which may be considered as a deterministic (grounding) and a non-deterministic (model search) part. The interface between these two is by means of a ground program, which is produced by the deterministic phase. Static Magic Sets will almost exclusively have an impact on the grounding phase, while dynamic Magic Sets also have the possibility to influence the model search phase. In particular, some assumptions made during the model search may render parts of the program irrelevant to the query, which may be captured by dynamic Magic Sets, but not (or only under very specific circumstances) by static Magic Sets.

In the literature, apart from our own work in [20], there is only one previous attempt for defining a Magic Set method for disjunctive Datalog, reported in [32,33], which will be referred to as Static Magic Sets (`SMS`) in this work. The basic idea of `SMS` is that bindings need to be propagated not only from rule heads to rule bodies (as in traditional Magic Sets), but also from one head predicate to other head predicates. In addition to producing definitions for the predicates defining Magic Sets, the method also introduces additional auxiliary predicates called *collecting* predicates. These collecting predicates however have a peculiar effect: Their use keeps the Magic Sets *static*. Indeed, both magic and collecting predicates are guaranteed to have deterministic definitions, which implies that disjunctive Datalog systems can exploit the Magic Sets only during the grounding phase. Most systems will actually produce a ground program which does contain neither magic nor collecting predicates.

In this article, we propose a *dynamic* Magic Set method for disjunctive Datalog with stratified negation under the stable model semantics, provide an implementation of it in the system DLV, and report on an extensive experimental evaluation. In more detail, the contributions are:

- ▶ We present a dynamic Magic Set method for disjunctive Datalog programs with stratified negation, referred to as Dynamic Magic Sets (`DMS`). Different from the previously proposed static method `SMS`, existing systems can exploit the information provided by the Magic Sets also during their nondeterministic model search phase. This feature allows for potentially exponential performance gains with respect to the previously proposed static method.
- ▶ We formally establish the correctness of `DMS`. In particular, we prove that the program obtained by the transformation `DMS` is query-equivalent to the original program. This result holds for both brave and cautious reasoning.
- ▶ We highlight a strong relationship between Magic Sets and unfounded sets, which characterize stable models. We can show that the atoms which are relevant for answering a query are either true or form an unfounded set, which eventually allows us to prove the query-equivalence results.
- ▶ Our results hold for a disjunctive Datalog language with stratified negation under the stable model semantics. In the literature, several works deal with non-disjunctive Datalog with stratified negation under the well-founded or the perfect model semantics, which are special cases of our language. For the static method `SMS`, an extension to disjunctive Datalog with stratified negation has previously only been sketched in [33].
- ▶ We have implemented a `DMS` optimization module inside the DLV system [43]. In this way, we could exploit the internal data-structures of the DLV system and embed `DMS` in the core of DLV. As a result, the technique is completely transparent to the end user. The system is available at `http://www.dlvsystem.com/magic/`.
- ▶ We have conducted extensive experiments on synthetic domains that highlight the potential of `DMS`. We have compared the performance of the DLV system without Magic Set optimization with `SMS` and with `DMS`. The results show that in many cases the Magic Set methods yield a significant performance benefit. Moreover, we can show that the dynamic method `DMS` can yield drastically better performance than the static `SMS`. Importantly, in cases in which `DMS` cannot be beneficial (if all or most of the instantiated program is relevant for answering a query), the overhead incurred is very light.
- ▶ We also report on experiments which evaluate the impact of `DMS` on an industrial application scenario on real-world data. The application involves data integration and builds on several results in the literature (for example [5,7,14,16,17,31]), which transform the problem of query answering over inconsistent databases (in this context stemming from integrating autonomous data sources) into query answering over disjunctive Datalog programs. By leveraging these results, `DMS` can be viewed as a query optimization method for inconsistent databases or for data integration systems. The results show that `DMS` can yield significant performance gains for queries of this application.

Organization. The main body of this article is organized as follows. In Section 2, preliminaries on disjunctive Datalog and on the Magic Set method for non-disjunctive Datalog queries are introduced. Subsequently, in Section 3 the extension `DMS` for the case of disjunctive Datalog programs is presented, and we show its correctness. In Section 4 we discuss the implementation and integration of the Magic Set method within the DLV system. Experimental results on synthetic benchmarks are reported in Section 5, while the application to data integration and its experimental evaluation is discussed in Section 6. Finally, related work is discussed in Section 7, and in Section 8 we draw our conclusions.

2 Preliminaries

In this section, (disjunctive) Datalog programs with (stratified) negation are briefly described, and the standard Magic Set method is presented together with the notion of sideways information passing strategy (SIPS) for Datalog rules.

2.1 Disjunctive Datalog Programs with Stratified Negation

In this paper, we adopt the standard Datalog name convention: Alphanumeric strings starting with a lowercase character are predicate or constant symbols, while alphanumeric strings starting with an uppercase character are variable symbols; moreover, we allow the use of positive integer constant symbols. Each predicate symbol is associated with a non-negative integer, referred to as its *arity*. An *atom* $p(\bar{t})$ is composed of a predicate symbol p and a list $\bar{t}=t_1,\ldots,t_k$ $(k \geq 0)$ of terms, each of which is either a constant or a variable. A *literal* is an atom $p(\bar{t})$ or a negated atom *not* $p(\bar{t})$; in the first case the literal is *positive*, while in the second it is *negative*.

A *disjunctive Datalog rule with negation* (short: Datalog$^{\vee,\neg}$ rule) r is of the form

$$p_1(\bar{t}_1) \ \vee \ \cdots \ \vee \ p_n(\bar{t}_n) \text{ :- } q_1(\bar{s}_1),\ \ldots,\ q_j(\bar{s}_j),$$
$$\qquad\qquad\qquad \mathit{not}\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ \mathit{not}\ q_m(\bar{s}_m).$$

where $p_1(\bar{t}_1),\ldots,\ p_n(\bar{t}_n),\ q_1(\bar{s}_1),\ldots,\ q_m(\bar{s}_m)$ are atoms and $n \geq 1$, $m \geq j \geq 0$. The disjunction $p_1(\bar{t}_1) \ \vee \ \cdots \ \vee \ p_n(\bar{t}_n)$ is the *head* of r, while the conjunction $q_1(\bar{s}_1),\ \ldots,\ q_j(\bar{s}_j),\ \mathit{not}\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ \mathit{not}\ q_m(\bar{s}_m)$ is the *body* of r. Moreover, $H(r)$ denotes the set of head atoms, while $B(r)$ denotes the set of body literals. We also use $B^+(r)$ and $B^-(r)$ for denoting the sets of atoms appearing in positive and negative body literals, respectively. If r is disjunction-free, that is $n = 1$, and negation-free, that is $B^{-}(r)$ is empty, then we say that r is a Datalog rule; if $B^{+}(r)$ is empty in addition, then we say that r is a *fact.* A *disjunctive Datalog program* $\mathcal{P}$ is a finite set of rules; if all the rules in it are disjunction- and negation-free, then $\mathcal{P}$ is a (standard) Datalog program.

Given a Datalog$^{\vee,\neg}$ program $\mathcal{P}$, a predicate belongs to the *Intensional Database* (IDB) if it is either in the head of a rule with non-empty body, or in the head of a disjunctive rule; otherwise, it belongs to the *Extensional Database* (EDB). The set of rules having IDB predicates in their heads is denoted by $IDB(\mathcal{P})$, while $EDB(\mathcal{P})$ denotes the remaining rules, that is, $EDB(\mathcal{P}) = \mathcal{P} \setminus IDB(\mathcal{P})$. For simplicity, we assume that predicates will always be of the same type (EDB or IDB) in any program.

The set of all constants appearing in a program $\mathcal{P}$ is the *universe* of $\mathcal{P}$ and is denoted by $U_{\mathcal{P}}$,[1] while the set of ground atoms constructable from predicates in $\mathcal{P}$ with constants in $U_{\mathcal{P}}$ is the *base* of $\mathcal{P}$, denoted by $B_{\mathcal{P}}$. We call an atom (rule, or program) *ground* if it does not contain any variables. A substitution ϑ is a function from variables to elements of $U_{\mathcal{P}}$. For an expression S (atom, literal, rule), by $S\vartheta$ we denote the expression obtained from S by substituting all occurrences of each variable X in S with $\vartheta(X)$. A ground atom $p(\bar{t})$ (resp. ground rule r_g) is an instance of an atom $p(\bar{t}')$ (resp. rule r) if there is a substitution ϑ from the variables in $p(\bar{t}')$ (resp. in r) to $U_{\mathcal{P}}$ such that $p(\bar{t}) = p(\bar{t}')\vartheta$ (resp. $r_g = r\vartheta$). Given a program $\mathcal{P}$, *Ground*$(\mathcal{P})$ denotes the set of all possible instances of rules in $\mathcal{P}$.

Given an atom $p(\bar{t})$ and a set of ground atoms A, by $A|_{p(\bar{t})}$ we denote the set of ground instances of $p(\bar{t})$ belonging to A. For example, $B_{\mathcal{P}}|_{p(\bar{t})}$ is the set of all ground atoms obtained by applying to $p(\bar{t})$ all the possible substitutions from the variables in $p(\bar{t})$ to $U_{\mathcal{P}}$, that is, the set of all the instances of $p(\bar{t})$. Abusing notation, if B is a set of atoms, by $A|_B$ we denote the union of all $A|_{p(\bar{t})}$, for each $p(\bar{t}) \in B$.

A desirable property of Datalog$^{\vee,\neg}$ programs is *safety.* A Datalog$^{\vee,\neg}$ rule r is safe if each variable appearing in r appears in at least one atom of $B^{+}(r)$. A Datalog$^{\vee,\neg}$ program is safe if all its rules are safe. Moreover, programs without recursion over negated literals constitute an interesting class of Datalog$^{\vee,\neg}$ programs. Without going into details, a predicate p in the head of a rule r *depends* on all the predicates q in the body of r; p depends on q positively if q appears in $B^{+}(r)$, and p depends on q negatively if q appears in $B^{-}(r)$. A program has recursion over negation if a cycle of dependencies with at least one negative dependency exists. If a program has no recursion over negation, then the program is stratified (short: Datalog$^{\vee,\neg_s}$). In this work only safe programs

[1] If $\mathcal{P}$ has no constants, an arbitrary constant is added to $U_{\mathcal{P}}$.

without recursion over negation are considered.

An *interpretation* for a program $\mathcal{P}$ is a subset I of $B_{\mathcal{P}}$. A positive ground literal $p(\bar{t})$ is true with respect to an interpretation I if $p(\bar{t}) \in I$; otherwise, it is false. A negative ground literal *not* $p(\bar{t})$ is true with respect to I if and only if $p(\bar{t})$ is false with respect to I, that is, if and only if $p(\bar{t}) \notin I$. The body of a ground rule r is true with respect to I if and only if all the body literals of r are true with respect to I, that is, if and only if $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. An interpretation I *satisfies* a ground rule $r \in Ground(\mathcal{P})$ if at least one atom in $H(r)$ is true with respect to I whenever the body of r is true with respect to I. An interpretation I is a *model* of a Datalog$^{\vee,\neg}$ program $\mathcal{P}$ if I satisfies all the rules in $Ground(\mathcal{P})$. Since an interpretation is a set of atoms, if I is an interpretation for a program $\mathcal{P}$, and $\mathcal{P}'$ is another program, then by $I|_{B_{\mathcal{P}'}}$ we denote the restriction of I to the base of $\mathcal{P}'$.

Given an interpretation I for a program $\mathcal{P}$, the reduct of $\mathcal{P}$ with respect to I, denoted by $Ground(\mathcal{P})^I$, is obtained by deleting from $Ground(\mathcal{P})$ all the rules r_g with $B^-(r_g) \cap I \neq \emptyset$, and then by removing all the negative literals from the remaining rules.

The semantics of a Datalog$^{\vee,\neg}$ program $\mathcal{P}$ is given by the set $\mathcal{SM}(\mathcal{P})$ of stable models of $\mathcal{P}$, where an interpretation M is a stable model for $\mathcal{P}$ if and only if M is a subset-minimal model of $Ground(\mathcal{P})^M$. It is well-known that there is exactly one stable model for any Datalog program, also in presence of stratified negation. However, for a Datalog$^{\vee,\neg_s}$ program $\mathcal{P}$, $|\mathcal{SM}(\mathcal{P})| \geq 1$ holds (Datalog$^{\vee,\neg}$ programs, instead, can also have no stable model).

Given a ground atom $p(\bar{t})$ and a Datalog$^{\vee,\neg}$ program $\mathcal{P}$, $p(\bar{t})$ is a *cautious* (or *certain*) consequence of $\mathcal{P}$, denoted by $\mathcal{P} \models_c p(\bar{t})$, if $p(\bar{t}) \in M$ for each $M \in \mathcal{SM}(\mathcal{P})$; $p(\bar{t})$ is a *brave* (or *possible*) consequence of $\mathcal{P}$, denoted by $\mathcal{P} \models_b p(\bar{t})$, if $p(\bar{t}) \in M$ for some $M \in \mathcal{SM}(\mathcal{P})$. Note that brave and cautious consequences coincide for Datalog programs, as these programs have a unique stable model. Moreover, cautious consequences of a Datalog$^{\vee,\neg_s}$ program $\mathcal{P}$ are also brave consequences of $\mathcal{P}$ because $|\mathcal{SM}(\mathcal{P})| \geq 1$ holds in this case.

Given a *query* $\mathcal{Q} = g(\bar{t})?$ (an atom),[2] $Ans_c(\mathcal{Q}, \mathcal{P})$ denotes the set of all substitutions ϑ for the variables of $g(\bar{t})$ such that $\mathcal{P} \models_c g(\bar{t})\vartheta$, while $Ans_b(\mathcal{Q}, \mathcal{P})$ denotes the set of substitutions ϑ for the variables of $g(\bar{t})$ such that $\mathcal{P} \models_b g(\bar{t})\vartheta$.

Let $\mathcal{P}$ and $\mathcal{P}'$ be two Datalog$^{\vee,\neg}$ programs and $\mathcal{Q}$ a query. Then $\mathcal{P}$ and $\mathcal{P}'$

[2] Note that more complex queries can still be expressed using appropriate rules. We assume that each constant appearing in $\mathcal{Q}$ also appears in $\mathcal{P}$; if this is not the case, then we can add to $\mathcal{P}$ a fact $p(\bar{t})$ such that p is a predicate not occurring in $\mathcal{P}$ and $\bar{t}$ are the arguments of $\mathcal{Q}$. Question marks will be usually omitted when referring to queries in the text.

are *brave-equivalent* with respect to $\mathcal{Q}$, denoted by $\mathcal{P} \equiv^{b}_{\mathcal{Q}} \mathcal{P}'$, if $Ans_b(\mathcal{Q}, \mathcal{P} \cup \mathcal{F}) = Ans_b(\mathcal{Q}, \mathcal{P}' \cup \mathcal{F})$ is guaranteed for each set of facts $\mathcal{F}$ defined over predicates which are EDB predicates of $\mathcal{P}$ or $\mathcal{P}'$; similarly, $\mathcal{P}$ and $\mathcal{P}'$ are *cautious-equivalent* with respect to $\mathcal{Q}$, denoted by $\mathcal{P} \equiv^{c}_{\mathcal{Q}} \mathcal{P}'$, if $Ans_c(\mathcal{Q}, \mathcal{P} \cup \mathcal{F}) = Ans_c(\mathcal{Q}, \mathcal{P}' \cup \mathcal{F})$ is guaranteed for each set of facts $\mathcal{F}$ defined over predicates which are EDB predicates of $\mathcal{P}$ or $\mathcal{P}'$.

2.2 Bottom-up Disjunctive Datalog Computation

Many Datalog$^{\vee,\neg}$ systems implement a two-phase computation. The first phase, referred to as *program instantiation* or *grounding*, is bottom-up. For an input program $\mathcal{P}$, it produces a ground program which is equivalent to $Ground(\mathcal{P})$, but significantly smaller. Most of the techniques used in this phase stem from bottom-up methods developed for classic and deductive databases; see for example [1] or [28,43] for details. Essentially, predicate instances which are known to be true or known to be false are identified and this knowledge is used for deriving further instances of this kind. Eventually, the truth values obtained in this way are used to produce rule instances which are not satisfied already. It is important to note that this phase behaves in a deterministic way with respect to stable models. No assumptions about truth or falsity of atoms are made, only definite knowledge is derived, which must hold in all stable models. For this reason, programs with multiple stable models cannot be solved by grounding.

The second phase is often referred to as *stable model search* and takes care of the non-deterministic computation. Essentially, one undefined atom is selected and its truth or falsity is assumed. The assumption might imply truth or falsity of other undefined atoms. Hence, the process is repeated until either an inconsistency is derived or all atoms have been interpreted. In the latter case an additional check is performed to ensure stability of the model. Details on this process can be found for example in [23]. Query answering is typically handled by storing all admissible answer substitutions as stable models are computed. For brave reasoning, each stable model can contribute substitutions to the set of answers. In this case the set of answers is initially empty. For cautious reasoning, instead, each stable model may eliminate some substitutions from the set of admissible answers. Therefore, in this case all possible substitutions for the input query are initially contained in the set of answers.

2.3 Sideways Information Passing for Datalog Rules

The Magic Set method aims at simulate a top-down evaluation of a query $\mathcal{Q}$, like for instance the one adopted by Prolog. According to this kind of evaluation, all the rules r such that $p(\bar{t}) \in H(r)$ and $H(r)\vartheta = \{\mathcal{Q}\vartheta'\}$ (for some substitution ϑ for all the variables of r and some substitution ϑ' for all the variables of $\mathcal{Q}$) are considered in a first step. Then the atoms in $B^+(r)\vartheta$ are taken as subqueries (we recall that standard Datalog rules have empty negative body), and the procedure is iterated. Note that, according to this process, if a (sub)query has some argument that is *bound* to a constant value, this information is "passed" to the atoms in the body. Moreover, the body is considered to be processed in a certain sequence, and processing a body atom may bind some of its arguments for subsequently considered body atoms, thus "generating" and "passing" bindings within the body. Whenever a body atom is processed, each of its argument is therefore considered to be either *bound* or *free*. We illustrate this mechanism by means of an example.

Example 2.1 Let $\texttt{path}(1,5)$ be a query for a program having the following inference rules:

r_1 : $\texttt{path}(\texttt{X},\texttt{Y}) \text{ :– } \texttt{edge}(\texttt{X},\texttt{Y}).$
r_2 : $\texttt{path}(\texttt{X},\texttt{Y}) \text{ :– } \texttt{edge}(\texttt{X},\texttt{Z}),\ \texttt{path}(\texttt{Z},\texttt{Y}).$

Since this is a Datalog program, brave and cautious consequences coincide. Moreover, let $\mathcal{F}_1 = \{\texttt{edge}(1,3), \texttt{edge}(2,4), \texttt{edge}(3,5)\}$ be the EDB of the program. A top-down evaluation scheme considers r_1 and r_2 with X and Y bound to 1 and 5, respectively. In particular, when considering r_1, the information about the binding of the two variables is passed to $\texttt{edge}(\texttt{X},\texttt{Y})$, which is indeed the only query atom occurring in r_1. Thus, the evaluation fails since $\texttt{edge}(1,5)$ does not occur in $\mathcal{F}_1$.

When considering r_2, instead, the binding information can be passed either to $\texttt{path}(\texttt{Z},\texttt{Y})$ or to $\texttt{edge}(\texttt{X},\texttt{Z})$. Suppose that atoms are evaluated according to their ordering in the rule (from left to right); then $\texttt{edge}(\texttt{X},\texttt{Z})$ is considered before $\texttt{path}(\texttt{Z},\texttt{Y})$. In particular, $\mathcal{F}_1$ contains the atom $\texttt{edge}(1,3)$, which leads us to map Z to 3. Eventually, this inferred binding information might be propagated to the remaining body atom $\texttt{path}(\texttt{Z},\texttt{Y})$, which hence becomes $\texttt{path}(3,5)$.

The process has now to be repeated by looking for an answer to $\texttt{path}(3,5)$. Again, rule r_1 can be considered, from which we conclude that this query is true since $\texttt{edge}(3,5)$ occurs in $\mathcal{F}_1$. Thus, $\texttt{path}(1,5)$ holds as well due to r_2. □

Note that in the example above we have two degrees of freedom in the specification of the top-down evaluation scheme. The first one concerns which ordering is used for processing the body atoms. While Prolog systems are usually required to follow the ordering in which the program is written, Datalog has a purely declarative semantics which is independent of the body ordering, allowing for an arbitrary ordering to be adopted. The second degree of freedom is slightly more subtle, and concerns the selection of the terms to be considered bound to constants from previous evaluations. Indeed, while we have considered the propagation of all the binding information that originates from previously processed body atoms, it is in general possible to restrict the top-down evaluation to partially propagate this information. For instance, one may desire to propagate only information generated from the evaluation of EDB predicates, or even just the information that is passed on via the head atom.

The specific propagation strategy adopted in the top-down evaluation scheme is called *sideways information passing strategy* (SIPS), which is just a way of formalizing a partial ordering over the atoms of each rule together with the specification of how the bindings originated and propagate [9,33]. To formalize this concept, in what follows, for each IDB atom $p(\bar{t})$, we shall denote its associated binding information (originated in a certain step of the top-down evaluation) by means of a string α built over the letters b and f, denoting "bound" and "free", respectively, for each argument of $p(\bar{t})$.

Definition 2.2 (SIPS for Datalog rules) *A* SIPS *for a Datalog rule r with respect to a binding α for the atom $p(\bar{t}) \in H(r)$ is a pair $(\prec_r^\alpha, f_r^\alpha)$, where:*

(1) $\prec_r^\alpha$ is a strict partial order over the atoms in $H(r) \cup B^+(r)$, such that $p(\bar{t}) \prec_r^\alpha q(\bar{s})$, for all atoms $q(\bar{s}) \in B^+(r)$; and,
(2) f_r^α is a function assigning to each atom $q(\bar{s}) \in H(r) \cup B^+(r)$ a subset of the variables in $\bar{s}$—intuitively, those made bound when processing $q(\bar{s})$.

Intuitively, for each atom $q(\bar{s})$ occurring in r, the strict partial order $\prec_r^\alpha$ specifies those atoms that have to be processed before processing atom $q(\bar{s})$. Eventually, an argument X of $q(\bar{s})$ is bound to a constant if there exists an atom $q'(\bar{s}')$ such that $q'(\bar{s}') \prec_r^\alpha q(\bar{s})$ and $X \in f_r^\alpha(q'(\bar{s}'))$. Note that the head atom $p(\bar{t})$ precedes all other atoms in $\prec_r^\alpha$.

Example 2.3 The SIPS we have adopted in Example 2.1 for r_1 with respect to the binding $\mathtt{bb}$ (originating from the query $\mathtt{path(1,5)}$) can be formalized as the pair $(\prec_{r_1}^{\mathtt{bb}}, f_{r_1}^{\mathtt{bb}})$, where $\mathtt{path(X,Y)} \prec_{r_1}^{\mathtt{bb}} \mathtt{edge(X,Y)}$, $f_{r_1}^{\mathtt{bb}}(\mathtt{path(X,Y)}) = \{\mathtt{X},\mathtt{Y}\}$, and $f_{r_1}^{\mathtt{bb}}(\mathtt{edge(X,Y)}) = \emptyset$. Instead, the SIPS we have adopted for r_2 with respect to the binding $\mathtt{bb}$ can be formalized as the pair $(\prec_{r_2}^{\mathtt{bb}}, f_{r_2}^{\mathtt{bb}})$, where $\mathtt{path(X,Y)} \prec_{r_2}^{\mathtt{bb}} \mathtt{edge(X,Z)} \prec_{r_2}^{\mathtt{bb}} \mathtt{path(Z,Y)}$, $f_{r_2}^{\mathtt{bb}}(\mathtt{path(X,Y)}) = \{\mathtt{X},\mathtt{Y}\}$, $f_{r_2}^{\mathtt{bb}}(\mathtt{edge(X,Z)}) = \{\mathtt{Z}\}$, and $f_{r_2}^{\mathtt{bb}}(\mathtt{path(Z,Y)}) = \emptyset$. □

All the algorithms and techniques we shall develop in this paper are orthogonal with respect to the underlying SIPSes to be used in the top-down evaluation. Thus, in Section 2.4, we shall assume that Datalog programs are provided in input together with some arbitrarily defined SIPS $(\prec_r^\alpha, f_r^\alpha)$, for each rule r and for each possible adornment α for the head atom in $H(r)$.

The Magic Set method is a strategy for simulating the top-down evaluation of a query by modifying the original program by means of additional rules, which narrow the computation to what is relevant for answering the query. We next provide a brief and informal description of the Magic Set rewriting technique. The reader is referred to [63] for a detailed presentation.

The method is structured in four main phases, which are informally illustrated below by means of Example 2.1.

(1) Adornment. The key idea is to materialize the binding information for IDB predicates that would be propagated during a top-down computation. In particular, the fact that an IDB predicate $p(\bar{t})$ is associated with a binding information α (i.e., a string over the letters b and f, one for each term in $\bar{t}$) is denoted by the atom obtained *adorning* the predicate symbol with the binding at hand, that is, by $p^{\alpha}(\bar{t})$. In what follows, the predicate p^{α} is said to be an adorned predicate.

First, adornments are created for query predicates so that an argument occurring in the query is adorned with the letter b if it is a constant, or with the letter f if it is a variable. For instance, the adorned version of the query atom $\texttt{path}(\texttt{1}, \texttt{5})$ is $\texttt{path}^{\texttt{bb}}(\texttt{1}, \texttt{5})$, which gives rise to the adorned predicate $\texttt{path}^{\texttt{bb}}$.

Each adorned predicate is eventually used to propagate its information into the body of the rules defining it according to a SIPS, thereby simulating a top-down evaluation. In particular, assume that the binding α has to be propagated into a rule r whose head is $p(\bar{t})$. Thus, the associated SIPS $(\prec_r^{\alpha}, f_r^{\alpha})$ determines which variables will be bound in the evaluation of the various body atoms. Indeed, a variable X of an atom $q(\bar{s})$ in r is bound if and only if either

(1) $X \in f_r^{\alpha}(q(\bar{s}))$ with $q(\bar{s}) = p(\bar{t})$; or,
(2) $X \in f_r^{\alpha}(b(\bar{z}))$ for an atom $b(\bar{z}) \in B^{+}(r)$ such that $b(\bar{z}) \prec_r^{\alpha} q(\bar{s})$ holds.

Adorning a rule r with respect to an adorned predicate p^{α} means propagating the binding information α, starting from the head predicate $p(\bar{t}) \in H(r)$, thereby creating a novel *adorned rule* where all the IDB predicates in r are substituted by the adorned predicates originating from the binding according to (1) and (2).

Example 2.4 Adorning the query $\texttt{path}(\texttt{1}, \texttt{5})$ generates $\texttt{path}^{\texttt{bb}}(\texttt{1}, \texttt{5})$. Then, propagating the binding information bb into the rule r_1, i.e., when adorning r_1 with $\texttt{path}^{\texttt{bb}}$, produces the following adorned rule (recall here that adornments apply only to IDB predicates, whereas edge is an EDB predicate):

$$r_1^a : \quad \texttt{path}^{\texttt{bb}}(\texttt{X}, \texttt{Y}) :- \texttt{edge}(\texttt{X}, \texttt{Y}).$$

Instead, when propagating $\texttt{bb}$ into the rule r_2 according to the SIPS $(\prec_{r_2}^{\texttt{bb}}, f_{r_2}^{\texttt{bb}})$ defined in Example 2.3, we obtain the following adorned rule:

$$r_2^a: \quad \texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y}) :- \texttt{edge}(\texttt{X},\texttt{Z}),\ \texttt{path}^{\texttt{bb}}(\texttt{Z},\texttt{Y}).$$
□

While adorning rules, novel binding information in the form of yet unseen adorned predicates may be generated, which should be used for adorning other rules. In fact, the adornment step is repeated until all bindings have been processed, yielding the *adorned program*, which is the set of all adorned rules created during the computation. For instance, in the above example, the adorned program just consists of r_1^a and r_2^a for no adorned predicate different from $\texttt{path}^{\texttt{bb}}$ is generated.

(2) Generation. In the second step of the Magic Set method, the adorned program is used to generate *magic rules*, which are used to simulate the top-down evaluation scheme and to single out the atoms relevant for answer the input query. For an adorned atom $p^\alpha(\bar{t})$, let $magic(p^\alpha(\bar{t}))$ be its *magic version* defined as the atom $magic\_p^\alpha(\bar{t}')$, where $\bar{t}'$ is obtained from $\bar{t}$ by eliminating all arguments corresponding to an f label in α, and where $magic\_p^\alpha$ is a new predicate symbol (for simplicity denoted by attaching the prefix "*magic_*" to the predicate symbol p^α). Intuitively, $magic\_p^\alpha(\bar{t}')\vartheta$ (ϑ a substitution) is inferred by the rules of the rewritten program whenever a top-down evaluation of the original program would process a subquery of the form $p^\alpha(\bar{t}'')$, where $\bar{t}''$ is obtained from $\bar{t}$ by applying ϑ to all terms in $\bar{t}'$.

Thus, if $q_i^{\beta_i}(\bar{s}_i)$ is an adorned atom (i.e., β_i is not the empty string) in the body of an adorned rule r^a having $p^\alpha(\bar{t})$ in head, a magic rule r^* is generated such that (i) $H(r^*) = \{magic(q_i^{\beta_i}(\bar{s}_i))\}$ and (ii) $B(r^*)$ is the union of $\{magic(p^\alpha(\bar{t}))\}$ and the set of all the atoms $q_j^{\beta_j}(\bar{s}_j) \in B^+(r)$ such that $q_j(\bar{s}_j) \prec_r^\alpha q_i(\bar{s}_i)$.

Example 2.5 In our running example, only one magic rule is generated,

$$r_2^*: \quad \texttt{magic\_path}^{\texttt{bb}}(\texttt{Z},\texttt{Y}) :- \texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y}),\ \texttt{edge}(\texttt{X},\texttt{Z}).$$

In fact, the adorned rule r_1^a does not produce any magic rule, since there is no adorned predicate in $B^+(r_1^a)$. □

(3) Modification. The adorned rules are subsequently modified by adding magic atoms to their bodies. These magic atoms limit the range of the head variables avoiding the inference of facts which cannot contribute to the derivation of the query. In particular, each adorned rule r^a, whose head atom is $p^\alpha(\bar{t})$, is modified by adding the atom $magic(p^\alpha(\bar{t}))$ to its body. The resulting rules are called *modified rules.*

Example 2.6 In our running example, the following modified rules are generated:

$r'_1:$ $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \texttt{ :- } \texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y}),\ \texttt{edge}(\texttt{X},\texttt{Y}).$
$r'_2:$ $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \texttt{ :- } \texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y}),\ \texttt{edge}(\texttt{X},\texttt{Z}),\ \texttt{path}^{\texttt{bb}}(\texttt{Z},\texttt{Y}).$ □

(4) Processing the Query. Finally, given the adorned predicate g^α obtained when adorning a query $g(\bar{t})$, (1) a *magic seed* $magic(g^\alpha(\bar{t}))$ (a fact) and (2) a rule $g(\bar{t})$:– $g^\alpha(\bar{t})$ are produced. In our example, $\texttt{magic\_path}^{\texttt{bb}}(1,5)$ and $\texttt{path}(\texttt{X},\texttt{Y}) \texttt{ :- } \texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y})$ are generated.

The complete rewritten program according to the Magic Set method consists of the magic, modified, and query rules (together with the original EDB). Given a Datalog program $\mathcal{P}$, a query $\mathcal{Q}$, and the rewritten program $\mathcal{P}'$, it is well-known that $\mathcal{P}$ and $\mathcal{P}'$ are equivalent with respect to $\mathcal{Q}$, i.e., $\mathcal{P}\equiv^b_\mathcal{Q}\mathcal{P}'$ and $\mathcal{P}\equiv^c_\mathcal{Q}\mathcal{P}'$ hold [63].

Example 2.7 The complete rewriting of our running example is as follows: [3]

$\texttt{magic\_path}^{\texttt{bb}}(1,5).$
$\texttt{path}(\texttt{X},\texttt{Y}) \texttt{ :- } \texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y}).$
$r^*_2:$ $\texttt{magic\_path}^{\texttt{bb}}(\texttt{Z},\texttt{Y}) \texttt{ :- } \texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y}), \texttt{edge}(\texttt{X},\texttt{Z}).$
$r'_1:$ $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \texttt{ :- } \texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y}),\ \texttt{edge}(\texttt{X},\texttt{Y}).$
$r'_2:$ $\texttt{path}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \texttt{ :- } \texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y}),\ \texttt{edge}(\texttt{X},\texttt{Z}),\ \texttt{path}^{\texttt{bb}}(\texttt{Z},\texttt{Y}).$

In this rewriting, $\texttt{magic\_path}^{\texttt{bb}}(\texttt{X},\texttt{Y})$ represents a potential sub-path of the paths from `1` to `5`. Therefore, when answering the query, only these sub-paths will be actually considered in the bottom-up computation. One can check that this rewriting is in fact equivalent to the original program with respect to the query `path(1,5)`. □

3 Magic Set Method for Datalog$^{\vee,\neg_s}$ Programs

In this section we present the Dynamic Magic Set algorithm (`DMS`) for the optimization of disjunctive programs with stratified negation. Before discussing the details of the algorithm, we informally present the main ideas that have been exploited for enabling the Magic Set method to work on disjunctive programs (without negation).

3.1 Overview of Binding Propagation in Datalog$^\vee$ Programs

As first observed in [33], while in non-disjunctive programs bindings are propagated only head-to-body, a Magic Set transformation for disjunctive programs

[3] The Magic Set rewriting of a program $\mathcal{P}$ affects only $IDB(\mathcal{P})$, so we usually omit $EDB(\mathcal{P})$ in examples.

has to propagate bindings also head-to-head in order to preserve soundness. Roughly, suppose that a predicate p is relevant for the query, and a disjunctive rule r contains $p(X)$ in the head. Then, besides propagating the binding from $p(X)$ to the body of r (as in the non-disjunctive case), the binding must also be propagated from $p(X)$ to the other head atoms of r. The reason is that any atom which is true in a stable model needs a supporting rule, which is a rule with a true body and in which the atom in question is the *only* true head atom. Therefore, r can yield support to the truth of $p(X)$ only if all other head atoms are false, which is due to the implicit minimality criterion in the semantics.

Consider, for instance, a Datalog$^{\vee}$ program $\mathcal{P}$ consisting of the rule $\mathtt{p(X)} \vee \mathtt{q(Y)} \mathrel{:-} \mathtt{a(X,Y)}, \mathtt{b(X)}$, and the query $\mathtt{p(1)}$. Even though the query propagates the binding for the predicate $\mathtt{p}$, in order to correctly answer the query we also need to evaluate the truth value of $\mathtt{q(Y)}$, which indirectly receives the binding through the body predicate $\mathtt{a(X,Y)}$. For instance, suppose that the program contains the facts $\mathtt{a(1,2)}$ and $\mathtt{b(1)}$; then the atom $\mathtt{q(2)}$ is relevant for the query $\mathtt{p(1)}$ (i.e., it should belong to the Magic Set of the query), since the truth of $\mathtt{q(2)}$ would invalidate the derivation of $\mathtt{p(1)}$ from the above rule, due to the minimality of the semantics. It follows that, while propagating the binding, the head atoms of disjunctive rules must be all adorned as well.

However, the adornment of the head of one disjunctive rule r may give rise to multiple rules, having different adornments for the head predicates. This process can be somehow seen as "splitting" r into multiple rules. While this is not a problem in the non-disjunctive case, the semantics of a disjunctive program may be affected. Consider, for instance, the program consisting of the rule $\mathtt{p(X,Y)} \vee \mathtt{q(Y,X)} \mathrel{:-} \mathtt{a(X,Y)}$, in which $\mathtt{p}$ and $\mathtt{q}$ are mutually exclusive (due to minimality) since they do not appear in any other rule head. Assuming the adornments $\mathtt{p}^{\mathtt{bf}}$ and $\mathtt{q}^{\mathtt{bf}}$ to be propagated, we might obtain rules whose heads have the form $\mathtt{p}^{\mathtt{bf}}\mathtt{(X,Y)} \vee \mathtt{q}^{\mathtt{fb}}\mathtt{(Y,X)}$ (derived while propagating $\mathtt{p}^{\mathtt{bf}}$) and $\mathtt{p}^{\mathtt{fb}}\mathtt{(X,Y)} \vee \mathtt{q}^{\mathtt{bf}}\mathtt{(Y,X)}$ (derived while propagating $\mathtt{q}^{\mathtt{bf}}$). These rules could support two atoms $\mathtt{p}^{\mathtt{bf}}\mathtt{(m,n)}$ and $\mathtt{q}^{\mathtt{bf}}\mathtt{(n,m)}$, while in the original program $\mathtt{p(m,n)}$ and $\mathtt{p(n,m)}$ could not hold simultaneously (due to semantic minimality), thus changing the original semantics.

The method proposed in [33] circumvents this problem by using some auxiliary predicates that collect all facts coming from the different adornments. For instance, in the above example, two rules of the form $\mathtt{collect\_p(X,Y)} \mathrel{:-} \mathtt{p}^{\mathtt{fb}}\mathtt{(X,Y)}$ and $\mathtt{collect\_p(X,Y)} \mathrel{:-} \mathtt{p}^{\mathtt{bf}}\mathtt{(X,Y)}$ are added for the predicate $\mathtt{p}$. The main deficiency of this approach is that collecting predicates will store a sizable superset of all the atoms relevant to answer the given query.

An important observation is that these collecting predicates are defined in a deterministic way. Since these predicates are used for restricting the compu-

```
Algorithm DMS($\mathcal{Q}$,$\mathcal{P}$)
Input: A Datalog$^{\vee,\neg_s}$ program $\mathcal{P}$, and a query $\mathcal{Q} = g(\bar{t})$?
Output: The rewritten program DMS($\mathcal{Q},\mathcal{P}$);
var: $S$, $D$: set of adorned predicates; $modifiedRules_{\mathcal{Q},\mathcal{P}}$,$magicRules_{\mathcal{Q},\mathcal{P}}$: set of rules;
begin
  1. $S := \emptyset$; $D := \emptyset$; $modifiedRules_{\mathcal{Q},\mathcal{P}} := \emptyset$; $magicRules_{\mathcal{Q},\mathcal{P}} := \{\textbf{BuildQuerySeed}(\mathcal{Q}, S)\}$;
  2. while $S \neq \emptyset$ do
  3.     $p^\alpha$ := an element of $S$;   remove $p^\alpha$ from $S$;   add $p^\alpha$ to $D$;
  4.     for each rule $r \in \mathcal{P}$ and for each atom $p(\bar{t}) \in H(r)$ do
  5.         $r^a$:=$\textbf{Adorn}(r, p^\alpha(\bar{t}), S, D)$;
  6.         $magicRules_{\mathcal{Q},\mathcal{P}} := magicRules_{\mathcal{Q},\mathcal{P}} \cup \textbf{Generate}(r, p^\alpha(\bar{t}), r^a)$;
  7.         $modifiedRules_{\mathcal{Q},\mathcal{P}} := modifiedRules_{\mathcal{Q},\mathcal{P}} \cup \{ \textbf{Modify}(r, r^a) \}$;
  8.     end for
  9. end while
 10. DMS($\mathcal{Q},\mathcal{P}$):=$magicRules_{\mathcal{Q},\mathcal{P}} \cup modifiedRules_{\mathcal{Q},\mathcal{P}} \cup EDB(\mathcal{P})$;
 11. return DMS($\mathcal{Q},\mathcal{P}$);
end.
```

Fig. 1. Dynamic Magic Set algorithm (DMS) for Datalog$^{\vee,\neg_s}$ programs

tation in [33], a consequence is that assumptions during the computation cannot be exploited for determining the relevant part of the program. In terms of bottom-up systems, this implies that the optimization affects only the grounding portion of the solver. Intuitively, it would be beneficial to also have a form of conditional relevance, exploiting also relevance for assumptions. In fact, in Section 5, we provide experimental evidence for this intuition.

In the following, we propose a novel Magic Set method that guarantees query equivalence and also allows for the exploitation of conditional or dynamic relevance, overcoming a major drawback of SMS.

3.2 DMS *Algorithm*

Our proposal to enhance the Magic Set method for disjunctive Datalog programs has two crucial features compared to the one of [33]:

(1) First, the semantics of the program is preserved by stripping off the adornments from non-magic predicates in modified rules, and not by introducing collecting predicates that can introduce overhead in the grounding process, as discussed in Section 3.1.
(2) Second, the proposed Magic Set technique is not just a way to cut irrelevant rules from the ground program; in fact, it allows for dynamic determination of relevance, thus optimizing also the nondeterministic computation by disabling parts of the programs which are not relevant in any extension of the current computation state.

The algorithm DMS implementing these strategies is reported in Figure 1 as pseudo-code. We assume that all variables are passed to functions by reference, in particular the variable S is modified inside ***BuildQuerySeed*** and ***Adorn***.

Its input is a Datalog$^{\vee,\neg_s}$ program [4] $\mathcal{P}$ and a query $\mathcal{Q}$. The algorithm uses two sets, S and D, to store adorned predicates to be propagated and already processed, respectively. After all the adorned predicates have been processed, the method outputs a rewritten program $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ consisting of a set of *modified* and *magic* rules, stored by means of the sets $\mathit{modifiedRules}_{\mathcal{Q},\mathcal{P}}$ and $\mathit{magicRules}_{\mathcal{Q},\mathcal{P}}$, respectively (together with the original EDB). The main steps of the algorithm are illustrated by means of the following running example.

Example 3.1 (Strategic Companies [15]) Let $C = \{c_1, \ldots, c_m\}$ be a collection of companies producing some goods in a set G, such that each company $c_i \in C$ is controlled by a set of other companies $O_i \subseteq C$. A subset of the companies $C' \subseteq C$ is a *strategic set* if it is a minimal set of companies satisfying the following conditions: Companies in C' produce all the goods in G; and $O_i \subseteq C'$ implies $c_i \in C'$, for each $i = 1, \ldots, m$.

We assume that each product is produced by at most two companies and that each company is controlled by at most three companies. It is known that the problem retains its hardness (for the second level of the polynomial hierarchy; see [15]) under these restrictions. We assume that production of goods is represented by an EDB containing a fact $\mathtt{produced\_by}(\mathtt{p},\mathtt{c_1},\mathtt{c_2})$ for each product p produced by companies c_1 and c_2, and that the control is represented by facts $\mathtt{controlled\_by}(\mathtt{c},\mathtt{c_1},\mathtt{c_2},\mathtt{c_3})$ for each company c controlled by companies c_1, c_2, and c_3. [5] This problem can be modeled via the following disjunctive program $\mathcal{P}_{sc}$:

$$r_3: \ \mathtt{sc}(\mathtt{C_1}) \ \vee \ \mathtt{sc}(\mathtt{C_2}) \ \text{:-}\ \mathtt{produced\_by}(\mathtt{P},\mathtt{C_1},\mathtt{C_2}).$$

$$r_4: \ \mathtt{sc}(\mathtt{C}) \ \text{:-}\ \mathtt{controlled\_by}(\mathtt{C},\mathtt{C_1},\mathtt{C_2},\mathtt{C_3}),\ \mathtt{sc}(\mathtt{C_1}),\ \mathtt{sc}(\mathtt{C_2}),\ \mathtt{sc}(\mathtt{C_3}).$$

Moreover, given a company $c \in C$, we consider a query $\mathcal{Q}_{sc} = \mathtt{sc}(\mathtt{c})$ asking whether c belongs to some strategic set of C. □

The computation starts in step *1* by initializing S, D, and $\mathit{modifiedRules}_{\mathcal{Q},\mathcal{P}}$ to the empty set. Then, the function ***BuildQuerySeed***$(\mathcal{Q}, S)$ is used for storing in $\mathit{magicRules}_{\mathcal{Q},\mathcal{P}}$ the magic seed, and inserting in the set S the adorned predicate of $\mathcal{Q}$. Note that we do not generate any query rules because standard atoms in the transformed program will not contain adornments. Details of ***BuildQuerySeed***$(\mathcal{Q}, S)$ are reported in Figure 2.

Example 3.2 Given the query $\mathcal{Q}_{sc} = \mathtt{sc}(\mathtt{c})$ and the program $\mathcal{P}_{sc}$, function ***BuildQuerySeed***$(\mathcal{Q}_{sc}, S)$ creates the fact $\mathtt{magic\_sc}^{\mathtt{b}}(\mathtt{c})$ and inserts $\mathtt{sc}^{\mathtt{b}}$ in S. □

[4] Note that the algorithm can be used for non-disjunctive and/or positive programs as a special case.

[5] If a product is produced by only one company, $c_2 = c_1$, and similarly for companies controlled by fewer than three companies.

```
Function BuildQuerySeed($\mathcal{Q}$, $S$)
Input: $\mathcal{Q}$: query;   $S$ : set of adorned predicates;
Output: The query seed (a magic atom);
var: $\alpha$: adornment string;
begin
  1.  Let $p(\bar{t})$ be the atom in $\mathcal{Q}$.
  2.  $\alpha := \epsilon$;
  3.  for each argument $t$ in $\bar{t}$ do
  4.     if $t$ is a constant then  $\alpha := \alpha b$;  else  $\alpha := \alpha f$;  end if
  5.  end for
  6.  add $p^\alpha$ to $S$;
  7.  return $magic(p^\alpha(\bar{t}))$;
end.
```

Fig. 2. BuildQuerySeed function

```
Function Adorn($r$, $p^\alpha(\bar{t})$, $S$, $D$)
Input: $r$: rule;   $p^\alpha(\bar{t})$: adorned atom;   $S$, $D$ : set of adorned predicates;
Output: an adorned rule;
var: $r^a$: adorned rule;   $\alpha_i$: adornment string;
begin
  1.  Let $(\prec_r^{p^\alpha(\bar{t})}, f_r^{p^\alpha(\bar{t})})$ be the SIPS associated with $r$ and $p^\alpha(\bar{t})$.
  2.  $r^a := r$;
  3.  for each IDB atom $p_i(\bar{t}_i)$ in $H(r) \cup B^+(r) \cup B^-(r)$ do
  4.     $\alpha_i := \epsilon$;
  5.     for each argument $t$ in $\bar{t}$ do
  6.        if $t$ is a constant then
  7.           $\alpha_i := \alpha_i b$;
  8.        else
  9.           Argument $t$ is a variable. Let $X$ be this variable.
 10.           if $X \in f_r^{p^\alpha(\bar{t})}(p(\bar{t}))$ or there is $q(\bar{s})$ in $B^+(r)$ such that
 11.                         $q(\bar{s}) \prec_r^{p^\alpha(\bar{t})} p_i(\bar{t}_i)$ and $X \in f_r^{p^\alpha(\bar{t})}(q(\bar{s}))$ then
 12.              $\alpha_i := \alpha_i b$;
 13.           else
 14.              $\alpha_i := \alpha_i f$;
 15.           end if
 16.        end if
 17.     end for
 18.     substitute $p_i(\bar{t}_i)$ in $r^a$ with $p_i^{\alpha_i}(\bar{t}_i)$;
 19.     if set $D$ does not contain $p_i^{\alpha_i}$ then  add $p_i^{\alpha_i}$ to $S$;  end if
 20.  end for
 21.  return $r^a$;
end.
```

Fig. 3. Adorn function

The core of the algorithm (steps *3*–*8*) is repeated until the set S is empty, i.e., until there is no further adorned predicate to be propagated. In particular, an adorned predicate p^α is moved from S to D in step *3*, and its binding is propagated in each (disjunctive) rule $r \in \mathcal{P}$ of the form

$$r: \ p(\bar{t}) \ \vee \ p_1(\bar{t}_1) \ \vee \ \cdots \ \vee \ p_n(\bar{t}_n) \coloneq q_1(\bar{s}_1), \ \ldots, \ q_j(\bar{s}_j), \\ not \ q_{j+1}(\bar{s}_{j+1}), \ \ldots, \ not \ q_m(\bar{s}_m).$$

(with $n \geq 0$) having an atom $p(\bar{t})$ in the head (note that the rule r is processed a number of times that equals the number of head atoms with predicate p; steps *4*–*8*).

(1) Adornment. Step *5* in Figure 1 implements the adornment of the rule. Different from the case of non-disjunctive positive programs, the binding of the predicate p^α needs to be also propagated to the atoms $p_1(\bar{t}_1), \dots, p_n(\bar{t}_n)$ in the head. Therefore, binding propagation has to be extended to the head atoms different from $p(\bar{t})$, which are therefore adorned according to a SIPS specifically conceived for disjunctive programs. Notation gets slightly more involved here: Since in non-disjunctive rules there is a single head atom, it was sufficient to specify an order and a function for each of its adornments (omitting the head atom in the notation). With disjunctive rules, an order and a function need to be specified for each adorned head atom, so it is no longer sufficient to include only the adornment in the notation, but we rather include the full adorned atom.

Definition 3.3 (SIPS for Datalog$^{\vee,\neg_s}$ rules) *A* SIPS *for a Datalog$^{\vee,\neg_s}$ rule r with respect to a binding α for an atom $p(\bar{t}) \in H(r)$ is a pair $(\prec_r^{p^\alpha(\bar{t})}, f_r^{p^\alpha(\bar{t})})$, where:*

(1) $\prec_r^{p^\alpha(\bar{t})}$ is a strict partial order over the atoms in $H(r) \cup B^+(r) \cup B^-(r)$, such that:
  - *(a) $p(\bar{t}) \prec_r^{p^\alpha(\bar{t})} q(\bar{s})$, for all atoms $q(\bar{s}) \in H(r) \cup B^+(r) \cup B^-(r)$ different from $p(\bar{t})$;*
  - *(b) for each pair of atoms $q(\bar{s}) \in (H(r) \setminus \{p(\bar{t})\}) \cup B^-(r)$ and $b(\bar{z}) \in H(r) \cup B^+(r) \cup B^-(r)$, $q(\bar{s}) \prec_r^{p^\alpha(\bar{t})} b(\bar{z})$ does not hold; and,*

(2) $f_r^{p^\alpha(\bar{t})}$ is a function assigning to each atom $q(\bar{s}) \in H(r) \cup B^+(r) \cup B^-(r)$ a subset of the variables in $\bar{s}$—intuitively, those made bound when processing $q(\bar{s})$.

As for Datalog rules, for each atom $q(\bar{s})$ occurring in r, the strict partial order $\prec_r^{p^\alpha(\bar{t})}$ specifies those atoms that have to be processed before processing atom $q(\bar{s})$, and an argument X of $q(\bar{s})$ is bound to a constant if there exists an atom $q'(\bar{s}')$ occurring in r such that $q'(\bar{s}') \prec_r^{p^\alpha(\bar{t})} q(\bar{s})$ and $X \in f_r^{p^\alpha(\bar{t})}(q'(\bar{s}'))$. The difference with respect to SIPSes for Datalog rules is precisely in the dependency from $p(\bar{t})$ in addition to α, and in condition *(1.b)* stating that head atoms different from $p(\bar{t})$ and negative body literals cannot provide bindings to variables of other atoms.

The underlying idea is that a rule which is used to "prove" the truth of an atom in a top-down method will be a rule which supports that atom. This implies that all other head atoms in that rule must be false and that the body must be true. Head atoms and atoms occurring in the negative body cannot "create" bindings (that is, restrict the values of variables), but these atoms are still relevant to the query, which leads to the restrictions in Definition 3.3.

Note that this definition considers each rule in isolation and is therefore independent of the inter-rule structure of a program. In particular, it is not important for the SIPS definition whether a program is cyclic or contains head cycles.

In the following, we shall assume that each Datalog$^{\vee,\neg_s}$ program is provided in input together with some arbitrarily defined SIPS for Datalog$^{\vee,\neg_s}$ rules $(\prec_r^{p^\alpha(\bar{t})}, f_r^{p^\alpha(\bar{t})})$. In fact, armed with $(\prec_r^{p^\alpha(\bar{t})}, f_r^{p^\alpha(\bar{t})})$, the adornment can be carried out precisely as we discussed for Datalog programs; in particular, we recall here that a variable X of an atom $q(\bar{s})$ in r is bound if and only if either:

(1) $X \in f_r^{p^\alpha(\bar{t})}(q(\bar{s}))$ with $q(\bar{s}) = p(\bar{t})$; or,
(2) $X \in f_r^{p^\alpha(\bar{t})}(b(\bar{z}))$ for an atom $b(\bar{z}) \in B^+(r)$ such that $b(\bar{z}) \prec_r^{p^\alpha(\bar{t})} q(\bar{s})$ holds.

The function $\boldsymbol{Adorn}(r, p^\alpha(\bar{t}), S, D)$ produces an adorned disjunctive rule r^a from an adorned atom $p^\alpha(\bar{t})$ and a suitable unadorned rule r (according to the bindings defined in the points (1) and (2) above), by inserting all newly adorned predicates in S. Hence, in step *5* the rule r^a is of the form

$$\begin{aligned} r^a :\ & p^\alpha(\bar{t}) \vee p_1^{\alpha_1}(\bar{t}_1) \vee \cdots \vee p_n^{\alpha_n}(\bar{t}_n) \text{ :- } q_1^{\beta_1}(\bar{s}_1),\ \ldots,\ q_j^{\beta_j}(\bar{s}_j), \\ & \qquad \mathit{not}\ q_{j+1}^{\beta_{j+1}}(\bar{s}_{j+1}),\ \ldots,\ \mathit{not}\ q_m^{\beta_m}(\bar{s}_m). \end{aligned}$$

Details of $\boldsymbol{Adorn}(r, p^\alpha(\bar{t}), S, D)$ are reported in Figure 3.

Example 3.4 Let us resume from Example 3.2. We are supposing that the adopted SIPS is passing the bindings via `produced_by` and `controlled_by` to the variables of `sc` atoms, in particular

$$\begin{aligned} \mathtt{sc(C_1)} &\prec_{r_3}^{\mathtt{sc^b(C_1)}} \mathtt{produced\_by(P, C_1, C_2)} \\ \mathtt{sc(C_1)} &\prec_{r_3}^{\mathtt{sc^b(C_1)}} \mathtt{sc(C_2)} \\ \mathtt{produced\_by(P, C_1, C_2)} &\prec_{r_3}^{\mathtt{sc^b(C_1)}} \mathtt{sc(C_2)} \end{aligned}$$

$$\begin{aligned} \mathtt{sc(C_2)} &\prec_{r_3}^{\mathtt{sc^b(C_2)}} \mathtt{produced\_by(P, C_1, C_2)} \\ \mathtt{sc(C_2)} &\prec_{r_3}^{\mathtt{sc^b(C_2)}} \mathtt{sc(C_1)} \\ \mathtt{produced\_by(P, C_1, C_2)} &\prec_{r_3}^{\mathtt{sc^b(C_2)}} \mathtt{sc(C_1)} \end{aligned}$$

$$\begin{aligned} \mathtt{sc(C)} &\prec_{r_4}^{\mathtt{sc^b(C)}} \mathtt{controlled\_by(C, C_1, C_2, C_3)} \\ \mathtt{sc(C)} &\prec_{r_4}^{\mathtt{sc^b(C)}} \mathtt{sc(C_1)} \\ \mathtt{sc(C)} &\prec_{r_4}^{\mathtt{sc^b(C)}} \mathtt{sc(C_2)} \\ \mathtt{sc(C)} &\prec_{r_4}^{\mathtt{sc^b(C)}} \mathtt{sc(C_3)} \\ \mathtt{controlled\_by(C, C_1, C_2, C_3)} &\prec_{r_4}^{\mathtt{sc^b(C)}} \mathtt{sc(C_1)} \end{aligned}$$

$$\texttt{controlled\_by}(\texttt{C},\texttt{C}_1,\texttt{C}_2,\texttt{C}_3) \prec_{r_4}^{\texttt{sc}^{\texttt{b}}(\texttt{C})} \texttt{sc}(\texttt{C}_2)$$
$$\texttt{controlled\_by}(\texttt{C},\texttt{C}_1,\texttt{C}_2,\texttt{C}_3) \prec_{r_4}^{\texttt{sc}^{\texttt{b}}(\texttt{C})} \texttt{sc}(\texttt{C}_3)$$

$$f_{r_3}^{\texttt{sc}^{\texttt{b}}(\texttt{C}_1)}(\texttt{sc}(\texttt{C}_1)) = \{\texttt{C}_1\}$$
$$f_{r_3}^{\texttt{sc}^{\texttt{b}}(\texttt{C}_1)}(\texttt{produced\_by}(\texttt{P},\texttt{C}_1,\texttt{C}_2)) = \{\texttt{P},\texttt{C}_2\}$$
$$f_{r_3}^{\texttt{sc}^{\texttt{b}}(\texttt{C}_1)}(\texttt{sc}(\texttt{C}_2)) = \emptyset$$

$$f_{r_3}^{\texttt{sc}^{\texttt{b}}(\texttt{C}_2)}(\texttt{sc}(\texttt{C}_2)) = \{\texttt{C}_2\}$$
$$f_{r_3}^{\texttt{sc}^{\texttt{b}}(\texttt{C}_2)}(\texttt{produced\_by}(\texttt{P},\texttt{C}_1,\texttt{C}_2)) = \{\texttt{P},\texttt{C}_1\}$$
$$f_{r_3}^{\texttt{sc}^{\texttt{b}}(\texttt{C}_2)}(\texttt{sc}(\texttt{C}_1)) = \emptyset$$

$$f_{r_4}^{\texttt{sc}^{\texttt{b}}(\texttt{C})}(\texttt{sc}(\texttt{C})) = \{\texttt{C}\}$$
$$f_{r_4}^{\texttt{sc}^{\texttt{b}}(\texttt{C})}(\texttt{controlled\_by}(\texttt{C},\texttt{C}_1,\texttt{C}_2,\texttt{C}_3)) = \{\texttt{C}_1,\texttt{C}_2,\texttt{C}_3\}$$
$$f_{r_4}^{\texttt{sc}^{\texttt{b}}(\texttt{C})}(\texttt{sc}(\texttt{C}_1)) = f_{r_4}^{\texttt{sc}^{\texttt{b}}(\texttt{C})}(\texttt{sc}(\texttt{C}_2)) = f_{r_4}^{\texttt{sc}^{\texttt{b}}(\texttt{C})}(\texttt{sc}(\texttt{C}_3)) = \emptyset$$

When $\texttt{sc}^{\texttt{b}}$ is removed from the set S, we first select rule r_3 and the head predicate $\texttt{sc}(\texttt{C}_1)$. Then the adorned version is

$$r_{3,1}^a : \ \texttt{sc}^{\texttt{b}}(\texttt{C}_1) \ \vee \ \texttt{sc}^{\texttt{b}}(\texttt{C}_2) \ \texttt{:-} \ \texttt{produced\_by}(\texttt{P},\texttt{C}_1,\texttt{C}_2).$$

Next, r_3 is processed again, this time with head predicate $\texttt{sc}(\texttt{C}_2)$, producing

$$r_{3,2}^a : \ \texttt{sc}^{\texttt{b}}(\texttt{C}_2) \ \vee \ \texttt{sc}^{\texttt{b}}(\texttt{C}_1) \ \texttt{:-} \ \texttt{produced\_by}(\texttt{P},\texttt{C}_1,\texttt{C}_2).$$

Finally, processing r_4 we obtain

$$r_4^a : \ \texttt{sc}^{\texttt{b}}(\texttt{C}) \ \texttt{:-} \ \texttt{controlled\_by}(\texttt{C},\texttt{C}_1,\texttt{C}_2,\texttt{C}_3),\ \texttt{sc}^{\texttt{b}}(\texttt{C}_1),\ \texttt{sc}^{\texttt{b}}(\texttt{C}_2),\ \texttt{sc}^{\texttt{b}}(\texttt{C}_3). \ \Box$$

(2) Generation. The algorithm uses the adorned rule r^a for generating and collecting the magic rules in step *6* (Figure 1). More specifically, ***Generate***$(r, p^\alpha(\bar{t}), r^a)$ produces magic rules according to the following schema: if $p_i^{\alpha_i}(\bar{t}_i)$ is an adorned atom (i.e., α_i is not the empty string) occurring in r^a and different from $p^\alpha(\bar{t})$, a magic rule r^* is generated such that (i) $H(r^*) = \{magic(p_i^{\alpha_i}(\bar{t}_i))\}$ and (ii) $B(r^*)$ is the union of $\{magic(p^\alpha(\bar{t}))\}$ and the set of all the atoms $q_j^{\beta_j}(\bar{s}_j) \in B^+(r)$ such that $q_j(\bar{s}_j) \prec_r^\alpha p_i(\bar{t}_i)$. Details of ***Generate***$(r, p^\alpha(\bar{t}), r^a)$ are reported in Figure 4.

Example 3.5 Continuing with our running example, by invoking ***Generate***$(r_3, \texttt{sc}^{\texttt{b}}(\texttt{C}_1), r_{3,1}^a)$, the following magic rule is produced:

$$r_{3,1}^* : \ \texttt{magic\_sc}^{\texttt{b}}(\texttt{C}_2) \ \texttt{:-} \ \texttt{magic\_sc}^{\texttt{b}}(\texttt{C}_1),\ \texttt{produced\_by}(\texttt{P},\texttt{C}_1,\texttt{C}_2).$$

```
Function Generate($r$, $p^{\alpha}(\bar{t})$, $r^a$)
Input: $r$: rule;   $p^{\alpha}(\bar{t})$: adorned atom;   $r^a$: adorned rule;
Output: a set of magic rules;
var: $R$: set of rules;   $r^*$: rule;
begin
  1.  Let $(\prec_r^{p^{\alpha}(\bar{t})}, f_r^{p^{\alpha}(\bar{t})})$ be the SIPS associated with $r$ and $p^{\alpha}(\bar{t})$.
  2.  $R := \emptyset$;
  3.  for each atom $p_i^{\alpha_i}(\bar{t}_i)$ in $H(r^a) \cup B^+(r^a) \cup B^-(r^a)$ different from $p^{\alpha}(\bar{t})$ do
  4.    if $\alpha_i \neq \epsilon$ then
  5.      $r^* := magic(p_i^{\alpha_i}(\bar{t}_i))$ :− $magic(p^{\alpha}(\bar{t}))$;
  6.      for each atom $p_j(\bar{t}_j)$ in $B^+(r)$ such that $p_j(\bar{t}_j) \prec_r^{p^{\alpha}(\bar{t})} p_i(\bar{t}_i)$ do
  7.        add atom $p_j(\bar{t}_j)$ to $B^+(r^*)$;
  8.      end for
  9.      $R := R \cup \{r^*\}$;
 10.    end if
 11.  end for
 12.  return $R$;
end.
```

Fig. 4. Generate function

Similarly, by invoking ***Generate***$(r_3, \mathtt{sc}^{\mathtt{b}}(\mathtt{C_2}), r^a_{3,2})$, the following magic rule is produced:

$$r^*_{3,2}: \ \mathtt{magic\_sc}^{\mathtt{b}}(\mathtt{C_1}) :- \mathtt{magic\_sc}^{\mathtt{b}}(\mathtt{C_2}),\ \mathtt{produced\_by}(\mathtt{P}, \mathtt{C_1}, \mathtt{C_2}).$$

Finally, the following magic rules are produced by ***Generate***$(r_4, \mathtt{sc}^{\mathtt{b}}(\mathtt{C}), r^a_4)$:

$$r^*_{4,1}: \ \mathtt{magic\_sc}^{\mathtt{b}}(\mathtt{C_1}) :- \mathtt{magic\_sc}^{\mathtt{b}}(\mathtt{C}),\ \mathtt{controlled\_by}(\mathtt{C}, \mathtt{C_1}, \mathtt{C_2}, \mathtt{C_3}).$$

$$r^*_{4,2}: \ \mathtt{magic\_sc}^{\mathtt{b}}(\mathtt{C_2}) :- \mathtt{magic\_sc}^{\mathtt{b}}(\mathtt{C}),\ \mathtt{controlled\_by}(\mathtt{C}, \mathtt{C_1}, \mathtt{C_2}, \mathtt{C_3}).$$

$$r^*_{4,3}: \ \mathtt{magic\_sc}^{\mathtt{b}}(\mathtt{C_3}) :- \mathtt{magic\_sc}^{\mathtt{b}}(\mathtt{C}),\ \mathtt{controlled\_by}(\mathtt{C}, \mathtt{C_1}, \mathtt{C_2}, \mathtt{C_3}). \quad \Box$$

(3) Modification. In step *7* the modified rules are generated and collected. The only difference with respect to the Datalog case is that the adornments are stripped off the original atoms. Specifically, given an adorned rule r^a associated with a rule r, a modified rule r' is obtained from r by adding to its body an atom $magic(p^{\alpha}(\bar{t}))$ for each atom $p^{\alpha}(\bar{t})$ occurring in $H(r^a)$. Hence, the function ***Modify***(r, r^a), reported in Figure 5, constructs a rule r' of the form

$$\begin{aligned} r': \ & p(\bar{t}) \lor p_1(\bar{t}_1) \lor \cdots \lor p_n(\bar{t}_n) :- magic(p^{\alpha}(\bar{t})), magic(p_1^{\alpha_1}(\bar{t}_1)), \ldots, \\ & \quad magic(p_n^{\alpha_n}(\bar{t}_n)), q_1(\bar{s}_1), \ldots, q_j(\bar{s}_j), not\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ not\ q_m(\bar{s}_m). \end{aligned}$$

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

Example 3.6 In our running example, we derive the following set of modified rules:

```
Function Modify($r$, $r^a$)
Input: $r$: rule;   $r^a$: adorned rule;
Output: a modified rule;
var: $r'$: rule;
begin
  1.  $r' := r$;
  2.  for each atom $p^\alpha(\bar{t})$ in $H(r^a)$ do
  3.     add $magic(p^\alpha(\bar{t}))$ to $B^+(r')$;
  4.  end for
  5.  return $r'$;
end.
```

Fig. 5. Modify function

$$r'_{3,1}:\ \mathtt{sc(C_1)}\ \vee\ \mathtt{sc(C_2)}\ \texttt{:-}\ \mathtt{magic\_sc^b(C_1),\ magic\_sc^b(C_2),}$$
$$\mathtt{produced\_by(P, C_1, C_2).}$$
$$r'_{3,2}:\ \mathtt{sc(C_2)}\ \vee\ \mathtt{sc(C_1)}\ \texttt{:-}\ \mathtt{magic\_sc^b(C_2),\ magic\_sc^b(C_1),}$$
$$\mathtt{produced\_by(P, C_1, C_2).}$$
$$r'_4:\ \mathtt{sc(C)}\ \texttt{:-}\ \mathtt{magic\_sc^b(C),\ controlled\_by(C, C_1, C_2, C_3),}$$
$$\mathtt{sc(C_1),\ sc(C_2),\ sc(C_3).}$$

Here, $r'_{3,1}$ (resp. $r'_{3,2}$, r'_4) is derived by adding magic predicates and stripping off adornments for the rule $r^a_{3,1}$ (resp. $r^a_{3,2}$, r^a_4). Thus, the optimized program $\mathtt{DMS}(\mathcal{Q}_{sc}, \mathcal{P}_{sc})$ comprises the above modified rules as well as the magic rules in Example 3.5, and the magic seed $\mathtt{magic\_sc^b(c)}$ (together with the original EDB). □

Before establishing the correctness of the technique, we briefly present an example of the application of DMS on a program containing disjunction and stratified negation.

Example 3.7 Let us consider a slight variant of the Strategic Companies problem described in Example 3.1 in which we have to determine whether a given company c does not belong to any strategic set. We can thus consider the query nsc(c) for the program $\mathcal{P}_{nsc}$ obtained by adding to $\mathcal{P}_{sc}$ the following rule:

$$r_{nsc}:\ \mathtt{nsc(C)}\ \texttt{:-}\ \mathtt{company(C),\ not\ sc(C).}$$

where company is an EDB predicate. Company c does not belong to any strategic set if the query is cautiously false.

In this case, processing the query produces the query seed $\mathtt{magic\_nsc^b(c)}$ (a fact) and the adorned predicate $\mathtt{nsc^b}$ (which is added to set S). After that, $\mathtt{nsc^b}$ is moved from S to D and rule r_{nsc} is considered. Assuming the following

SIP:

$$\texttt{nsc(C)} \prec_{r_{nsc}}^{\texttt{nsc}^{\texttt{b}}\texttt{(C)}} \texttt{company(C)} \qquad\qquad \texttt{nsc(C)} \prec_{r_{nsc}}^{\texttt{nsc}^{\texttt{b}}\texttt{(C)}} \texttt{sc(C)}$$

$$f_{r_{nsc}}^{\texttt{nsc}^{\texttt{b}}\texttt{(C)}}(\texttt{nsc(C)}) = \{\texttt{C}\} \qquad f_{r_{nsc}}^{\texttt{nsc}^{\texttt{b}}\texttt{(C)}}(\texttt{company(P)}) = f_{r_{nsc}}^{\texttt{nsc}^{\texttt{b}}\texttt{(C)}}(\texttt{sc(C)}) = \emptyset$$

by invoking ***Adorn***$(r_{nsc}, \texttt{nsc}^{\texttt{b}}\texttt{(C)}, S, D)$ we obtain the following adorned rule:

r^a_{nsc} : $\texttt{nsc}^{\texttt{b}}\texttt{(C)}$:– $\texttt{company(C)}$, $\texttt{not sc}^{\texttt{b}}\texttt{(C)}$.

The new adorned predicate $\texttt{sc}^{\texttt{b}}$ is added to S. Then, ***Generate***$(r_{nsc}, \texttt{nsc}^{\texttt{b}}\texttt{(C)}, r^a_{nsc})$ and ***Modify***(r_{nsc}, r^a_{nsc}) produce the following magic and modified rules:

r^*_{nsc} : $\texttt{magic\_sc}^{\texttt{b}}\texttt{(C)}$:– $\texttt{magic\_nsc}^{\texttt{b}}\texttt{(C)}$.

r'_{nsc} : $\texttt{nsc(C)}$:– $\texttt{magic\_nsc}^{\texttt{b}}\texttt{(C)}$, $\texttt{company(C)}$, $\texttt{not sc(C)}$.

The algorithm then processes the adorned atom $\texttt{sc}^{\texttt{b}}$. Hence, if the SIPS presented in Example 3.4 is assumed, the rewritten program comprises the following rules: r'_{nsc}, $r'_{3,1}$, $r'_{3,2}$, r'_4, r^*_{nsc}, $r^*_{3,1}$, $r^*_{3,2}$, $r^*_{4,1}$, $r^*_{4,2}$ and $r^*_{4,3}$. □

3.3 Query Equivalence Result

We conclude the presentation of the DMS algorithm by formally proving its correctness. We would like to point out that all of these results hold for any kind of SIPS, as long as it conforms to Definition 3.3. Therefore, in the remainder of this section, we assume that any program comes with some associated SIPS. In the proofs, we use the well established notion of unfounded set for disjunctive Datalog programs (possibly with negation) defined in [44]. Before introducing unfounded sets, however, we have to define partial interpretations, that is, interpretations for which some atoms may be undefined.

Definition 3.8 (Partial Interpretation) *Let $\mathcal{P}$ be a Datalog$^{\vee,\neg}$ program. A* partial interpretation *for $\mathcal{P}$ is a pair $\langle T, N\rangle$ such that $T \subseteq N \subseteq B_{\mathcal{P}}$. The atoms in T are interpreted as true, while the atoms in N are not false and those in $N \setminus T$ are undefined. All other atoms are false.*

Note that total interpretations are a special case in which $T = N$. We can then formalize the notion of unfounded set.

Definition 3.9 (Unfounded Sets) *Let $\langle T, N\rangle$ be a partial interpretation for a Datalog$^{\vee,\neg}$ program $\mathcal{P}$, and $X \subseteq B_{\mathcal{P}}$ be a set of atoms. Then, X is an* unfounded set *for $\mathcal{P}$ with respect to $\langle T, N\rangle$ if and only if, for each ground rule $r_g \in Ground(\mathcal{P})$ with $X \cap H(r_g) \neq \emptyset$, at least one of the following conditions holds:* (1.a) $B^+(r_g) \not\subseteq N$; (1.b) $B^-(r_g) \cap T \neq \emptyset$; (2) $B^+(r_g) \cap X \neq \emptyset$; (3) $H(r_g) \cap (T \setminus X) \neq \emptyset$.

Intuitively, conditions (1.a), (1.b) and (3) check if the rule is satisfied by $\langle T, N\rangle$ regardless of the atoms in X, while condition (2) checks whether the rule can be satisfied by taking the atoms in X as false.

Example 3.10 Consider again the program $\mathcal{P}_{sc}$ of Example 3.1 and assume $EDB(\mathcal{P}_{sc}) = \{\mathtt{produced\_by}(\mathtt{p}, \mathtt{c}, \mathtt{c_1})\}$. Then $Ground(\mathcal{P}_{sc})$ consists of the rule

$$r_{sc}: \ \mathtt{sc}(\mathtt{c}) \vee \mathtt{sc}(\mathtt{c_1}) \ {:-}\ \mathtt{produced\_by}(\mathtt{p}, \mathtt{c}, \mathtt{c_1}).$$

(together with facts, and rules having some ground instance of EDB predicate not occurring in $EDB(\mathcal{P}_{sc})$, omitted for simplicity). Consider now a partial interpretation $\langle M_{sc}, B_{\mathcal{P}_{sc}}\rangle$ such that $M_{sc} = \{\mathtt{produced\_by}(\mathtt{p}, \mathtt{c}, \mathtt{c_1}), \mathtt{sc}(\mathtt{c})\}$. Thus, $\{\mathtt{sc}(\mathtt{c_1})\}$ is an unfounded set for $\mathcal{P}$ with respect to $\langle M_{sc}, B_{\mathcal{P}_{sc}}\rangle$ (r_{sc} satisfies condition (3) of Definition 3.9), while $\{\mathtt{sc}(\mathtt{c}), \mathtt{sc}(\mathtt{c_1})\}$ is not (r_{sc} violates all conditions). □

The following is an adaptation of Theorem 4.6 in [44] to our notation.

Theorem 3.11 ([44]) *Let $\langle T, N\rangle$ be a partial interpretation for a Datalog$^{\vee,\neg}$ program $\mathcal{P}$. Then, for any stable model M of $\mathcal{P}$ such that $T \subseteq M \subseteq N$, and for each unfounded set X of $\mathcal{P}$ with respect to $\langle T, N\rangle$, $M \cap X = \emptyset$ holds.*

Example 3.12 In Example 3.10, we have shown that $\{\mathtt{sc}(\mathtt{c_1})\}$ is an unfounded set for $\mathcal{P}$ with respect to $\langle M_{sc}, B_{\mathcal{P}_{sc}}\rangle$. Note that the total interpretation M_{sc} is a stable model of $\mathcal{P}_{sc}$, and that the unfounded set $\{\mathtt{sc}(\mathtt{c_1})\}$ is disjoint from M_{sc}. □

Equipped with these notions and Theorem 3.11, we now proceed to prove the correctness of the DMS strategy. In particular, we shall first show that the method is *sound* in that, for each stable model M of $\mathtt{DMS}(\mathcal{Q}, \mathcal{P})$, there is a stable model M' of $\mathcal{P}$ such that $M'|_{\mathcal{Q}} = M|_{\mathcal{Q}}$ (i.e., the two models coincide when restricted to the query). Then, we prove that the method is also *complete*, i.e., for each stable model M' of $\mathcal{P}$, there is a stable model M of $\mathtt{DMS}(\mathcal{Q}, \mathcal{P})$ such that $M'|_{\mathcal{Q}} = M|_{\mathcal{Q}}$.

In both parts of the proof, we shall exploit the following (syntactic) relationship between the original program and the transformed one.

Lemma 3.13 *Let $\mathcal{P}$ be a Datalog$^{\vee,\neg_s}$ program, $\mathcal{Q}$ a query, and let $magic(p^\alpha(\bar{t}))$ be a ground atom*[6] *in $B_{\mathtt{DMS}(\mathcal{Q},\mathcal{P})}$ (the base of the transformed*

[6] Note that in this way the lemma refers only to rules that contain a head atom

program). Then the ground rule

$$r_g: \; p(\bar{t}) \vee p_1(\bar{t}_1) \vee \cdots \vee p_n(\bar{t}_n) \;\text{:--}\; q_1(\bar{s}_1),\; \ldots,\; q_j(\bar{s}_j), \\ \quad \textit{not}\; q_{j+1}(\bar{s}_{j+1}),\; \ldots,\; \textit{not}\; q_m(\bar{s}_m).$$

belongs to Ground($\mathcal{P}$) if and only if the ground rule

$$r'_g: \; p(\bar{t}) \vee p_1(\bar{t}_1) \vee \cdots \vee p_n(\bar{t}_n) \;\text{:--}\; \textit{magic}(p^{\alpha}(\bar{t})), \textit{magic}(p_1^{\alpha_1}(\bar{t}_1)), \ldots, \\ \textit{magic}(p_n^{\alpha_n}(\bar{t}_n)), q_1(\bar{s}_1), \ldots, q_j(\bar{s}_j), \textit{not}\; q_{j+1}(\bar{s}_{j+1}),\; \ldots,\; \textit{not}\; q_m(\bar{s}_m).$$

belongs to Ground($\mathtt{DMS}(\mathcal{Q}, \mathcal{P})$)*.*

Proof. ($\Rightarrow$) Consider the following rule $r \in \mathcal{P}$ such that $r_g = r\vartheta$ for some substitution ϑ:

$$r: \; p(\bar{t}') \vee p_1(\bar{t}'_1) \vee \cdots \vee p_n(\bar{t}'_n) \;\text{:--}\; q_1(\bar{s}'_1),\; \ldots,\; q_j(\bar{s}'_j), \\ \textit{not}\; q_{j+1}(\bar{s}'_{j+1}),\; \ldots,\; \textit{not}\; q_m(\bar{s}'_m).$$

Since $\textit{magic}(p^{\alpha}(\bar{t}))$ is a ground atom in $B_{\mathtt{DMS}(\mathcal{Q},\mathcal{P})}$, p^{α} has been inserted in the set S at some point of the Magic Set transformation, and it has eventually been used to adorn and modify r, thereby producing the following rule $r' \in \mathtt{DMS}(\mathcal{Q}, \mathcal{P})$:

$$r': \; p(\bar{t}') \vee p_1(\bar{t}'_1) \vee \cdots \vee p_n(\bar{t}'_n) \;\text{:--}\; \textit{magic}(p^{\alpha}(\bar{t}')), \textit{magic}(p_1^{\alpha_1}(\bar{t}'_1)), \ldots, \\ \textit{magic}(p_n^{\alpha_n}(\bar{t}'_n)), q_1(\bar{s}'_1), \ldots, q_j(\bar{s}'_j), \textit{not}\; q_{j+1}(\bar{s}'_{j+1}),\; \ldots,\; \textit{not}\; q_m(\bar{s}'_m).$$

Clearly enough, the substitution ϑ mapping r into r_g can also be used to map r' into r'_g, since the magic atoms added into the positive body of r' are defined over a subset of the variables occurring in head atoms.

($\Leftarrow$) Let $r' \in \mathtt{DMS}(\mathcal{Q}, \mathcal{P})$ be a rule such that $r'_g = r'\vartheta$ for some substitution ϑ:

$$r': \; p(\bar{t}') \vee p_1(\bar{t}'_1) \vee \cdots \vee p_n(\bar{t}'_n) \;\text{:--}\; \textit{magic}(p^{\alpha}(\bar{t}')), \textit{magic}(p_1^{\alpha_1}(\bar{t}'_1)), \ldots, \\ \textit{magic}(p_n^{\alpha_n}(\bar{t}'_n)), q_1(\bar{s}'_1), \ldots, q_j(\bar{s}'_j), \textit{not}\; q_{j+1}(\bar{s}'_{j+1}),\; \ldots,\; \textit{not}\; q_m(\bar{s}'_m).$$

By the construction of $\mathtt{DMS}(\mathcal{Q}, \mathcal{P})$, r' is a modified rule produced by adding some magic atom to the positive body of a rule $r \in \mathcal{P}$ of the form:

$$r: \; p(\bar{t}') \vee p_1(\bar{t}'_1) \vee \cdots \vee p_n(\bar{t}'_n) \;\text{:--}\; q_1(\bar{s}'_1),\; \ldots,\; q_j(\bar{s}'_j), \\ \textit{not}\; q_{j+1}(\bar{s}'_{j+1}),\; \ldots,\; \textit{not}\; q_m(\bar{s}'_m).$$

for which a magic predicate has been generated during the transformation.

Thus, the substitution ϑ mapping r' to r'_g can also be used to map r to r_g, since r and r' have the same variables. □

3.3.1 Soundness of the Magic Set Method

Let us now start with the first part of the proof, in particular, by stating some further definitions and notations. Given a model M' of $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$, and a model $N' \subseteq M'$ of $Ground(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$, we next define the set of atoms which are relevant for $\mathcal{Q}$ but are false with respect to N'.

Definition 3.14 (Killed Atoms) *Given a model M' for $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$, and a model $N' \subseteq M'$ of $Ground(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$, the set $killed^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ of the* killed atoms *with respect to M' and N' is defined as:*

$$\{k(\bar{t}) \in B_{\mathcal{P}} \setminus N' \mid \text{ either } k \text{ is an EDB predicate, or} \\ \text{there is a binding } \alpha \text{ such that } magic(k^{\alpha}(\bar{t})) \in N'\}.$$

Example 3.15 We consider the program $\mathtt{DMS}(\mathcal{Q}_{sc},\mathcal{P}_{sc})$ presented in Section 3.2 (we recall that $\mathcal{Q}_{sc} = \mathtt{sc(c)}$), the EDB $\{\mathtt{produced\_by(p,c,c_1)}\}$ introduced in Example 3.10, and a stable model $M'_{sc} = \{\mathtt{produced\_by(p,c,c_1)}, \mathtt{sc(c)}, \mathtt{magic\_sc^b(c)}, \mathtt{magic\_sc^b(c_1)}\}$ for $\mathtt{DMS}(\mathcal{Q}_{sc},\mathcal{P}_{sc})$. Thus, $Ground(\mathtt{DMS}(\mathcal{Q}_{sc},\mathcal{P}_{sc}))^{M_{sc}}$ consists of the following rules:

$$\mathtt{magic\_sc^b(c)}. \qquad \mathtt{magic\_sc^b(c_1)} \coloneq \mathtt{magic\_sc^b(c)}.$$

$$\mathtt{sc(c)} \vee \mathtt{sc(c_1)} \coloneq \mathtt{magic\_sc^b(c)},\ \mathtt{magic\_sc^b(c_1)},\ \mathtt{produced\_by(p,c,c_1)}.$$

Since M'_{sc} is also a model of the program above, we can compute $killed^{M'_{sc}}_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M'_{sc})$ and check that $\mathtt{sc(c_1)}$ belongs to it because of $\mathtt{magic\_sc^b(c_1)}$ in M'_{sc}. Note that, by definition, also false ground instances of EDB predicates like $\mathtt{produced\_by(p,c_1,c)}$ or $\mathtt{controlled\_by(c,c_1,c_1,c_1)}$ belong to $killed^{M'_{sc}}_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M'_{sc})$. Moreover, note that no other atom belongs to this set. □

The intuition underlying the definition above is that killed atoms are either false ground instances of some EDB predicate, or false atoms which are relevant with respect to $\mathcal{Q}$ (for there exists an associated magic atom in the model N'); since N' is a model of $Ground(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$ contained in M', we expect that these atoms are also false in any stable model for $\mathcal{P}$ containing $M'|_{B_{\mathcal{P}}}$ (which, we recall here, is the model M' restricted on the atoms originally occurring in $\mathcal{P}$).

Example 3.16 Let us resume from Example 3.15. We have that $M'_{sc}|_{\mathcal{P}_{sc}} = \{\mathtt{produced\_by(p,c,c_1)}, \mathtt{sc(c)}\}$, which coincides with model M_{sc} of Example 3.10. Hence, we already know that $\{\mathtt{sc(c_1)}\}$ is an unfounded set for $\mathcal{P}_{sc}$ with respect to $\langle M_{sc}, B_{\mathcal{P}_{sc}}\rangle$. Since each other atom $k(\bar{t})$ in $killed^{M'_{sc}}_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M'_{sc})$

is such that k is an EDB predicate, we also have that $killed^{M'_{sc}}_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M'_{sc})$ is an unfounded set for $\mathcal{P}_{sc}$ with respect to $\langle M_{sc}, B_{\mathcal{P}_{sc}}\rangle$. Therefore, as a consequence of Theorem 3.11, each stable model M of $\mathcal{P}_{sc}$ such that $M_{sc} \subseteq M \subseteq B_{\mathcal{P}_{sc}}$ (in this case only M_{sc} itself) is disjoint from $killed^{M'_{sc}}_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M'_{sc})$. $\Box$

This intuition is formalized below.

Proposition 3.17 *Let M' be a model for* $\texttt{DMS}(\mathcal{Q},\mathcal{P})$*, and $N' \subseteq M'$ be a model of $Ground(\texttt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$. Then, $killed^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ is an unfounded set for $\mathcal{P}$ with respect to $\langle M'|_{B_\mathcal{P}}, B_\mathcal{P}\rangle$.*

Proof. According to Definition 3.9 of unfounded sets (for $\mathcal{P}$ with respect to $\langle M'|_{B_\mathcal{P}}, B_\mathcal{P}\rangle$), given any rule r_g in $Ground(\mathcal{P})$ of the form

$$\begin{aligned} r_g:\ k(\bar{t}) \vee p_1(\bar{t}_1) \vee \cdots \vee p_n(\bar{t}_n) \mathrel{:-} & \ q_1(\bar{s}_1),\ \ldots,\ q_j(\bar{s}_j), \\ & not\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ not\ q_m(\bar{s}_m). \end{aligned}$$

we have to show that if $k(\bar{t}) \in killed^{M'}_{\mathcal{Q},\mathcal{P}}(N') \cap H(r_g)$, then at least one of the following conditions holds: (1.a) $B^+(r_g) \not\subseteq B_\mathcal{P}$; (1.$b$) $B^-(r_g) \cap M'|_{B_\mathcal{P}} \neq \emptyset$; (2) $B^+(r_g) \cap killed^{M'}_{\mathcal{Q},\mathcal{P}}(N') \neq \emptyset$; (3) $H(r_g) \cap (M'|_{B_\mathcal{P}} \setminus killed^{M'}_{\mathcal{Q},\mathcal{P}}(N')) \neq \emptyset$.

Note that the properties above refer to the original program $\mathcal{P}$. However, our hypothesis is formulated over the transformed one $\texttt{DMS}(\mathcal{Q},\mathcal{P})$ (for instance, we know that M' is a model of $\texttt{DMS}(\mathcal{Q},\mathcal{P})$). The line of the proof is then to analyze $\texttt{DMS}(\mathcal{Q},\mathcal{P})$ in the light of its syntactic relationships with $\mathcal{P}$ established via Lemma 3.13. In particular, recall first that, by Definition 3.14, there is a binding α such that $magic(\mathtt{k}^\alpha(\bar{\mathtt{t}})) \in N'$ (and, hence, $magic(k^\alpha(\bar{t}))$ is a ground atom in $B_{\texttt{DMS}(\mathcal{Q},\mathcal{P})}$). Thus, we can apply Lemma 3.13 and conclude the existence of a ground rule $r'_g \in Ground(\texttt{DMS}(\mathcal{Q},\mathcal{P}))$ such that:

$$\begin{aligned} r'_g:\ k(\bar{t}) \vee p_1(\bar{t}_1) \vee \cdots \vee p_n(\bar{t}_n) \mathrel{:-} & \ magic(k^\alpha(\bar{t})), magic(p_1^{\alpha_1}(\bar{t}_1)), \ldots, \\ & magic(p_n^{\alpha_n}(\bar{t}_n)), q_1(\bar{s}_1), \ldots, q_j(\bar{s}_j), not\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ not\ q_m(\bar{s}_m). \end{aligned}$$

Since M' is a model of $\texttt{DMS}(\mathcal{Q},\mathcal{P})$, the proof is just based on analyzing the following three scenarios that exhaustively cover all possibilities (concerning the fact that the rule r'_g is satisfied by M'):

(S1) $B^-(r'_g) \cap M' \neq \emptyset$, i.e., the negative body of r'_g is false with respect to M';

(S2) $B^+(r'_g) \not\subseteq M'$, i.e., the positive body of r'_g is false with respect to M';

(S3) $B^-(r'_g) \cap M' = \emptyset$, $B^+(r'_g) \subseteq M'$, and $H(r'_g) \cap M' \neq \emptyset$, i.e., none of the previous cases holds, and hence the head of r'_g is true with respect to M'.

In the remaining, we shall show that **(S1)** implies condition (1.b), **(S2)** implies condition (2), and **(S3)** implies either (2) or (3). In fact, note that condition (1.a) cannot hold.

(S1) Assume that $B^-(r'_g) \cap M' \neq \emptyset$. Since $B^-(r_g) = B^-(r'_g)$ and $B^-(r_g) \subseteq B_{\mathcal{P}}$, from $B^-(r'_g) \cap M' \neq \emptyset$ we immediately conclude $B^-(r_g) \cap M'|_{B_{\mathcal{P}}} \neq \emptyset$, i.e., (1.$b$) holds.

(S2) Assume that $B^+(r'_g) \not\subseteq M'$, and let $r' \in \mathtt{DMS}(\mathcal{Q}, \mathcal{P})$ be a modified rule such that $r'_g = r'\vartheta$ for some substitution ϑ:

$$r' : \; k(\bar{t}') \vee p_1(\bar{t}'_1) \vee \cdots \vee p_n(\bar{t}'_n) \text{ :- } \mathit{magic}(k^\alpha(\bar{t}')), \mathit{magic}(p_1^{\alpha_1}(\bar{t}'_1)), \ldots,$$
$$\mathit{magic}(p_n^{\alpha_n}(\bar{t}'_n)), q_1(\bar{s}'_1), \ldots, q_j(\bar{s}'_j), \mathit{not}\ q_{j+1}(\bar{s}'_{j+1}), \; \ldots, \; \mathit{not}\ q_m(\bar{s}'_m).$$

We first claim that $B^+(r'_g)|_{B_{\mathcal{P}}} \not\subseteq N'$ must hold in this case. To prove the claim, observe that during the *Generation* step preceding the production of r', a magic rule r^*_i such that $H(r^*_i) = \{\mathit{magic}(p_i^{\alpha_i}(\bar{t}'_i))\}$ and $B^+(r^*_i) \subseteq \{\mathit{magic}(k^\alpha(\bar{t}')),\ q_1(\bar{s}'_1), \ldots, q_j(\bar{s}'_j)\}$ has been produced for each $1 \leq i \leq n$ (we recall that magic rules have empty negative bodies). Hence, since the variables of r^*_i are a subset of the variables of r', by applying the substitution ϑ to r^*_i we obtain a ground rule $r^*_{i,g}$ such that $H(r^*_{i,g}) = \{\mathit{magic}(p_i^{\alpha_i}(\bar{t}_i))\}$ and $B^+(r^*_{i,g}) \subseteq \{\mathit{magic}(k^\alpha(\bar{t})), q_1(\bar{s}_1), \ldots, q_j(\bar{s}_j)\} = \{\mathit{magic}(k^\alpha(\bar{t}))\} \cup B^+(r'_g)|_{B_{\mathcal{P}}}$. Thus, if $B^+(r'_g)|_{B_{\mathcal{P}}} \subseteq N'$, from the above magic rules and since N' is a model containing $\mathit{magic}(k^\alpha(\bar{t}))$ by assumption, then we would conclude that $B^+(r'_g) \subseteq N'$. However, this is impossible, since $N' \subseteq M'$ and $B^+(r'_g) \not\subseteq M'$ imply $B^+(r'_g) \not\subseteq N'$.

Now, $B^+(r'_g)|_{B_{\mathcal{P}}} \not\subseteq N'$ implies the existence of an atom $q_i(\bar{s}_i) \in B^+(r'_g)|_{B_{\mathcal{P}}}$ such that $q_i(\bar{s}_i) \notin N'$, that is, $q_i(\bar{s}_i) \in B_{\mathcal{P}} \setminus N'$. In particular, we can assume w.l.o.g. that, for any $q(\bar{s}) \in B^+(r'_g)|_{B_{\mathcal{P}}}$ with $q(\bar{s}') \prec_r^{k^\alpha(\bar{t}')} q_i(\bar{s}'_i)$, it is the case that $q(\bar{s}) \in N'$, where r is the rule in $\mathcal{P}$ from which the modified rule r' has been generated (just take a $\prec_r^{k^\alpha(\bar{t}')}$-minimum element in $B^+(r'_g)|_{B_{\mathcal{P}}} \setminus N'$). If q_i is an EDB predicate, the atom $q_i(\bar{s}_i)$ belongs to $\mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ by the definition of killed atoms. Otherwise, q_i is an IDB predicate. In this case, there is a magic rule r^*_i, produced during the *Generation* step preceding the production of r', such that $H(r^*_i) = \{\mathit{magic}(q_i^{\beta_i}(\bar{s}'_i))\}$ and $B(r^*_i) = \{\mathit{magic}(k^\alpha(\bar{t}'))\} \cup \{q(\bar{s}') \in B^+(r) \mid q(\bar{s}') \prec_r^{k^\alpha(\bar{t}')} q_i(\bar{s}'_i)\}$. Thus, $r^*_{i,g} = r^*_i\vartheta$ belongs to $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q}, \mathcal{P}))$. In particular, $B^+(r^*_{i,g}) \subseteq N'$ holds because $\mathit{magic}(k^\alpha(\bar{t}))$ belongs to N' and by the properties of $q_i(\bar{s}_i)$. Therefore, since N' is a model of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q}, \mathcal{P}))^{M'}$, $\mathit{magic}(q_i^{\beta_i}(\bar{s}_i))$ belongs to N', from which $q_i(\bar{s}_i) \in \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ follows from the definition of killed atoms. Thus, independently of the type (EDB, IDB) of q_i, (2) holds.

(S3) Assume that $B^+(r'_g) \subseteq M'$, $B^-(r'_g) \cap M' = \emptyset$, and $H(r'_g) \cap M' \neq \emptyset$. First, observe that from $B^-(r'_g) \cap M' = \emptyset$ we can conclude that there is a rule in $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q}, \mathcal{P}))^{M'}$ obtained from r'_g by removing its negative body literals. Consider now the rules $r^*_{i,g}$ produced during the *Generation* step, for each $1 \leq i \leq n$ (as in **(S2)**). We distinguish two cases.

If $\{q_1(\bar{s}_1), \ldots, q_j(\bar{s}_j)\} \subseteq N'$, since $magic(k^\alpha(\bar{t})) \in N'$, we can conclude that $B^+(r^*_{i,g}) \subseteq N'$, for each $1 \leq i \leq n$. Moreover, since N' is a model of $Ground(\texttt{DMS}(\mathcal{Q}, \mathcal{P}))^{M'}$, the latter implies that $magic(p_i^{\alpha_i}(\bar{t}_i)) \in N'$, for each $1 \leq i \leq n$. Then $B^+(r'_g) \subseteq N'$ holds, and so $H(r'_g) \cap N' \neq \emptyset$ (because N' is a model of $Ground(\texttt{DMS}(\mathcal{Q}, \mathcal{P}))^{M'}$). We now observe that $H(r'_g) \cap (M'|_{B_\mathcal{P}} \setminus killed^{M'}_{\mathcal{Q},\mathcal{P}}(N')) \neq \emptyset$ is equivalent to $(H(r'_g) \cap M'|_{B_\mathcal{P}}) \setminus killed^{M'}_{\mathcal{Q},\mathcal{P}}(N') \neq \emptyset$. Moreover, the latter is equivalent to $(H(r_g) \cap M') \setminus killed^{M'}_{\mathcal{Q},\mathcal{P}}(N') \neq \emptyset$ because $H(r'_g)$ contains only standard atoms and $H(r'_g) = H(r_g)$. In addition, from $N' \subseteq M'$ we conclude $H(r_g) \cap N' \subseteq H(r_g) \cap M'$, and by Definition 3.14, $N' \cap killed^{M'}_{\mathcal{Q},\mathcal{P}}(N') = \emptyset$ holds. Hence, $(H(r_g) \cap M') \setminus killed^{M'}_{\mathcal{Q},\mathcal{P}}(N') \supseteq H(r_g) \cap N'$, which is not empty, and so condition (3) holds.

Otherwise, $\{q_1(\bar{s}_1), \ldots, q_j(\bar{s}_j)\} \not\subseteq N'$. Let $i \in \{1, \ldots, j\}$ be such that $q_i(\bar{s}_i) \notin N'$ and, for any $q(\bar{s}) \in B^+(r'_g)|_{B_\mathcal{P}}$, $q(\bar{s}') \prec_r^{k^\alpha(\bar{t}')} q_i(\bar{s}'_i)$ implies $q(\bar{s}) \in N'$ (where r is the rule in $\mathcal{P}$ from which the modified rule r' has been generated). If q_i is an EDB predicate, the atom $q_i(\bar{s}_i)$ belongs to $killed^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ by the definition of killed atoms. Otherwise, q_i is an IDB predicate and there is a magic rule $r^*_{i,g} \in Ground(\texttt{DMS}(\mathcal{Q}, \mathcal{P}))$ having an atom $magic(q_i^{\beta_i}(\bar{s}_i))$ in head, and such that $B^+(r^*_{i,g}) \subseteq N'$. Therefore, $magic(q_i^{\beta_i}(\bar{s}_i))$ belongs to N', from which $q_i(\bar{s}_i) \in killed^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ follows from the definition of killed atoms. Thus, independently of the type (EDB, IDB) of q_i, (2) holds. $\Box$

We can now complete the first part of the proof.

Lemma 3.18 *For each stable model M' of $\texttt{DMS}(\mathcal{Q}, \mathcal{P})$, there is a stable model M of $\mathcal{P}$ such that $M \supseteq M'|_{B_\mathcal{P}}$.*

Proof. Let M be a stable model of $\mathcal{P} \cup M'|_{B_\mathcal{P}}$, the program obtained by adding to $\mathcal{P}$ a fact for each atom in $M'|_{B_\mathcal{P}}$. We shall show that M is in fact a stable model of $\mathcal{P}$ such that $M \supseteq M'|_{B_\mathcal{P}}$. Of course, M is a model of $\mathcal{P}$ such that $M \supseteq M'|_{B_\mathcal{P}}$. So, the line of the proof is to show that if M is not stable, then it is possible to build a model N' of $Ground(\texttt{DMS}(\mathcal{Q}, \mathcal{P}))^{M'}$ such that $N' \subset M'$, thereby contradicting the minimality of M' over the models of $Ground(\texttt{DMS}(\mathcal{Q}, \mathcal{P}))^{M'}$.

Assume, for the sake of contradiction, that M is not stable and let $N \subset M$ be a model of $Ground(\mathcal{P})^M$. Define N' as the interpretation $(N \cap M'|_{B_\mathcal{P}}) \cup (M' \setminus B_\mathcal{P})$. By construction, note that $N' \subseteq M'$, since M' coincides with $M'|_{B_\mathcal{P}} \cup (M' \setminus B_\mathcal{P})$. In fact, in the case where $N' = M'$, we would have that $N \supseteq M'|_{B_\mathcal{P}}$, since $(N \cap M'|_{B_\mathcal{P}})$ and $(M' \setminus B_\mathcal{P})$ are disjoint. Hence, N would not only be a model for $Ground(\mathcal{P})^M$ but also a model for $Ground(\mathcal{P} \cup M'|_{B_\mathcal{P}})^M$, while on the other hand $N \subset M$ holds. However, this is impossible, since M is a stable model of $\mathcal{P} \cup M'|_{B_{\mathcal{P}}}$. So, $N' \subset M'$ must hold. Hence, to complete the proof and get a contradiction, it remains to show that N' is actually a model of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$, i.e., it satisfies all the rules in $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$. To this end, we have to consider the following two kinds of rules:

(1) Consider a ground magic rule $r^*_g \in \mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$ such that $B^+(r^*_g) \subseteq N'$, and let $\mathit{magic}(p^\alpha(\bar{t}))$ be the (only) atom in $H(r^*_g)$. Since $N' \subset M'$, $B^+(r^*_g) \subseteq N'$ implies that $B^+(r^*_g) \subset M'$. In fact, since M' is a model of $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ and $|H(r^*_g)| = 1$, $\mathit{magic}(p^\alpha(\bar{t})) \in M'$ must hold (we recall that $B^-(r^*_g) = \emptyset$). Moreover, since $B_{\mathcal{P}}$ does not contain any magic atom, $\mathit{magic}(p^\alpha(\bar{t}))$ is also contained in $M' \setminus B_{\mathcal{P}}$. Thus, by the construction of N', we can conclude that $H(r^*_g) \cap N' \neq \emptyset$.

(2) Consider a rule obtained by removing the negative literals from a ground modified rule $r'_g \in \mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))$ where

$$\begin{aligned} r'_g:\ p(\bar{t}) \vee p_1(\bar{t}_1) \vee \cdots \vee p_n(\bar{t}_n) &\coloneq \mathit{magic}(p^\alpha(\bar{t})), \mathit{magic}(p_1^{\alpha_1}(\bar{t}_1)), \ldots, \\ \mathit{magic}(p_n^{\alpha_n}(\bar{t}_n)), q_1(\bar{s}_1), \ldots, q_j(\bar{s}_j), &\mathit{not}\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ \mathit{not}\ q_m(\bar{s}_m). \end{aligned}$$

and where $B^+(r'_g) \subseteq N'$. Observe that $B^-(r'_g) \cap M' = \emptyset$ holds by the definition of reduct. Moreover, let r_g be the rule of $\mathit{Ground}(\mathcal{P})$ associated with r'_g (according to Lemma 3.13):

$$\begin{aligned} r_g:\ p(\bar{t}) \vee p_1(\bar{t}_1) \vee \cdots \vee p_n(\bar{t}_n) \coloneq q_1(\bar{s}_1),\ \ldots,\ q_j(\bar{s}_j), \\ \mathit{not}\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ \mathit{not}\ q_m(\bar{s}_m). \end{aligned}$$

We have to show that $H(r'_g) \cap N' \neq \emptyset$. The proof is based on establishing the following properties on r'_g and r_g:

- $M \cap \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M') = \emptyset;$ (1)
- $(H(r'_g) \setminus M') \cap M = \emptyset;$ (2)
- $B^-(r'_g) \cap M = \emptyset;$ (3)
- $H(r'_g) \cap M' = H(r'_g) \cap M'|_{B_{\mathcal{P}}} = H(r'_g) \cap M;$ (4)
- $H(r_g) \cap N \neq \emptyset.$ (5)

In particular, we shall directly prove (1), and show the following implications: (1)→(2)∧(3), (2)→(4), and (3)→(5). Eventually, based on (4) and (5), the fact that $H(r'_g) \cap N' \neq \emptyset$ can be easily derived as follows: Since $H(r_g) \subseteq B_{\mathcal{P}}$, by the definition of N' we can conclude that $H(r_g) \cap N' = H(r_g) \cap (N \cap M'|_{B_{\mathcal{P}}}) = (H(r_g) \cap N) \cap (H(r_g) \cap M'|_{B_{\mathcal{P}}})$. Moreover, because of (4) and the fact that $H(r_g) = H(r'_g)$, $H(r_g) \cap N'$ coincides in turn with $(H(r_g) \cap N) \cap (H(r_g) \cap M)$. Then, recall that $N \subset M$. Thus, $H(r_g) \cap N' = H(r_g) \cap N$, which is not empty by (5).

In order to complete the proof, we have to show that all the above equations actually hold.

Proof of (1). We recall that, by Proposition 3.17, we already know that $\mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$ is an unfounded set for $\mathcal{P}$ with respect to $\langle M'|_{B_\mathcal{P}}, B_\mathcal{P}\rangle$. In fact, one may notice that $\mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$ is an unfounded set for $\mathcal{P} \cup M'|_{B_\mathcal{P}}$ with respect to $\langle M'|_{B_\mathcal{P}}, B_\mathcal{P}\rangle$ too, since the rules added to $\mathcal{P}$ are facts corresponding to the atoms in $M'|_{B_\mathcal{P}}$ and $M'|_{B_\mathcal{P}} \cap \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M') = \emptyset$ by Definition 3.14. Thus, since $M \supseteq M'|_{B_\mathcal{P}}$ and M is a stable model of $\mathcal{P} \cup M'|_{B_\mathcal{P}}$, we can apply Theorem 3.11 in order to conclude that $M \cap \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M') = \emptyset$.

Proof of (2). After (1), we can just show that $H(r'_g) \setminus M' \subseteq \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$. In fact, since $N' \subset M'$, we note that $B^+(r'_g) \subseteq N'$ implies $B^+(r'_g) \subset M'$. Thus, $H(r'_g) \setminus M' \subseteq \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$ follows by Definition 3.14 and the form of rule r'_g.

Proof of (3). After (1), we can just show that $B^-(r'_g) \subseteq \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$. Actually, we show that the IDB atoms in $B^-(r'_g)$ belong to $\mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$, as EDB atoms in $B^-(r'_g)$ clearly belong to $\mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$ because $B^-(r'_g) \cap M' = \emptyset$ by assumption. To this end, consider a modified rule $r' \in \mathtt{DMS}(\mathcal{Q}, \mathcal{P})$ such that $r'_g = r'\vartheta$ for some substitution ϑ:

$$
\begin{aligned}
r' :\ & p(\bar{t}') \vee p_1(\bar{t}'_1) \vee \cdots \vee p_n(\bar{t}'_n) \mathrel{:-} \mathit{magic}(p^{\alpha}(\bar{t}')), \mathit{magic}(p_1^{\alpha_1}(\bar{t}'_1)), \ldots, \\
& \quad \mathit{magic}(p_n^{\alpha_n}(\bar{t}'_n)), q_1(\bar{s}'_1), \ldots, q_j(\bar{s}'_j), \mathit{not}\ q_{j+1}(\bar{s}'_{j+1}),\ \ldots,\ \mathit{not}\ q_m(\bar{s}'_m).
\end{aligned}
$$

During the *Generation* step preceding the production of r', a magic rule r^*_i with $H(r^*_i) = \{\mathit{magic}(q_i^{\beta_i}(\bar{s}'_i))\}$ and where $B^+(r^*_i) \subseteq B^+(r')$ has been produced for each $j+1 \leq i \leq m$ such that q_i is an IDB predicate. Hence, since the variables of r^*_i are a subset of the variables of r', the substitution ϑ can be used to map r^*_i to a ground rule $r^*_{i,g} = r^*_i\vartheta$ with $H(r^*_{i,g}) = \{\mathit{magic}(q_i^{\beta_i}(\bar{s}_i))\}$ and $B^+(r^*_{i,g}) \subseteq B^+(r'_g)$. Now, since $B^+(r'_g) \subseteq N' \subset M'$, we can conclude that $B^+(r^*_{i,g})$ is in turn contained in M'. Thus, the head of $r^*_{i,g}$ must be true with respect to M' (we recall that magic rules have empty negative bodies). That is, $\mathit{magic}(q_i^{\beta_i}(\bar{s}_i)) \in M'$ holds, for each $j+1 \leq i \leq m$ such that q_i is an IDB predicate. Moreover, $B^-(r'_g) \cap M' = \emptyset$ implies that $q_i^{\beta_i}(\bar{s}_i) \in B_\mathcal{P} \setminus M'$, as $q_i^{\beta_i}(\bar{s}_i) \in B^-(r'_g)$. Thus, by Definition 3.14, $q_i^{\beta_i}(\bar{s}_i) \in \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$.

Proof of (4). The property immediately follows from (2) and the fact that $H(r'_g) \subseteq B_\mathcal{P}$ and $M \supseteq M'|_{B_\mathcal{P}}$.

Proof of (5). Note that $B^-(r_g) = B^-(r'_g)$, and so (3) implies that there is a rule in $\mathit{Ground}(\mathcal{P})^M$ obtained from r_g by removing the atoms in $B^-(r_g)$. Note also that $B^+(r_g) = B^+(r'_g) \cap B_\mathcal{P} \subseteq N' \cap B_\mathcal{P}$ (since $B^+(r'_g) \subseteq N'$). Thus, by the definition of N', $B^+(r_g) \subseteq N$ (more specifically, $B^+(r_g) \subseteq$

$N \cap M'|_{B_{\mathcal{P}}}$). Moreover, since N is a model of $Ground(\mathcal{P})^M$, the latter entails that $H(r_g) \cap N \neq \emptyset$. □

Theorem 3.19 *Let $\mathcal{Q}$ be a query for a Datalog$^{\vee,\neg_s}$ program $\mathcal{P}$. Then, for each stable model M' of* $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$*, there is a stable model M of $\mathcal{P}$ such that $M'|_{\mathcal{Q}} = M|_{\mathcal{Q}}$.*

Proof. Because of Lemma 3.18, for each stable model M' of $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$, there is a stable model M of $\mathcal{P}$ such that $M \supseteq M'|_{B_{\mathcal{P}}}$. Thus, we trivially have that $M|_{\mathcal{Q}} \supseteq M'|_{\mathcal{Q}}$ holds. We now show that the inclusion cannot be proper.

In fact, by the definition of $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$, the magic seed is associated to any ground instance of $\mathcal{Q}$. Then $B_{\mathcal{P}}|_{\mathcal{Q}} \setminus M' \subseteq killed^{M'}_{\mathcal{Q},\mathcal{P}}(M')$ by Definition 3.14 (we recall that $B_{\mathcal{P}}|_{\mathcal{Q}}$ denotes the ground instances of $\mathcal{Q}$). By Proposition 3.17, $killed^{M'}_{\mathcal{Q},\mathcal{P}}(M')$ is an unfounded set for $\mathcal{P}$ with respect to $\langle M'|_{B_{\mathcal{P}}}, B_{\mathcal{P}}\rangle$. Hence, by Theorem 3.11, we have that $M \cap killed^{M'}_{\mathcal{Q},\mathcal{P}}(M') = \emptyset$. It follows that $M \cap (B_{\mathcal{P}}|_{\mathcal{Q}} \setminus M') = \emptyset$. Thus, $M|_{\mathcal{Q}} \setminus M'|_{\mathcal{Q}} = \emptyset$, which combined with $M|_{\mathcal{Q}} \supseteq M'|_{\mathcal{Q}}$ implies $M|_{\mathcal{Q}} = M'|_{\mathcal{Q}}$. □

3.3.2 Completeness of the Magic Set Method

For the second part of the proof, we construct an interpretation for $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ based on one for $\mathcal{P}$.

Definition 3.20 (Magic Variant) *Let I be an interpretation for $\mathcal{P}$. We define an interpretation $variant^{\infty}_{\mathcal{Q},\mathcal{P}}(I)$ for* $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$*, called the magic variant of I with respect to $\mathcal{Q}$ and $\mathcal{P}$, as the limit of the following sequence:*

$$\begin{aligned}
&variant^{0}_{\mathcal{Q},\mathcal{P}}(I) = EDB(\mathcal{P}); \text{ and}\\
&variant^{i+1}_{\mathcal{Q},\mathcal{P}}(I) = variant^{i}_{\mathcal{Q},\mathcal{P}}(I) \cup\\
&\qquad \{p(\bar{t}) \in I \mid \text{there is a binding } \alpha \text{ such that}\\
&\qquad\qquad\qquad magic(p^{\alpha}(\bar{t})) \in variant^{i}_{\mathcal{Q},\mathcal{P}}(I)\} \cup\\
&\qquad \{magic(p^{\alpha}(\bar{t})) \mid \exists\, r^*_g \in Ground(\mathtt{DMS}(\mathcal{Q},\mathcal{P})) \text{ such that}\\
&\qquad\qquad magic(p^{\alpha}(\bar{t})) \in H(r^*_g) \text{ and } B^+(r^*_g) \subseteq variant^{i}_{\mathcal{Q},\mathcal{P}}(I)\}, \quad \forall i \geq 0.
\end{aligned}$$

Example 3.21 Consider the program $\mathtt{DMS}(\mathcal{Q}_{sc},\mathcal{P}_{sc})$ presented in Section 3.2, the EDB $\{\mathtt{produced\_by}(\mathtt{p},\mathtt{c},\mathtt{c_1})\}$ and the interpretation $M_{sc} = \{\mathtt{produced\_by}(\mathtt{p},\mathtt{c},\mathtt{c_1}), \mathtt{sc}(\mathtt{c})\}$. We next compute the magic variant $variant^{\infty}_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M_{sc})$ of M_{sc} with respect to $\mathcal{Q}_{sc}$ and $\mathcal{P}_{sc}$. We start the sequence with the original EDB: $variant^{0}_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M_{sc}) = \{\mathtt{produced\_by}(\mathtt{p},\mathtt{c},\mathtt{c_1})\}$. For $variant^{1}_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M_{sc})$, we add $\mathtt{magic\_sc^b}(\mathtt{c})$ (the query seed), while for $variant^{2}_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M_{sc})$, we add $\mathtt{sc}(\mathtt{c})$ (because $\mathtt{sc}(\mathtt{c}) \in M_{sc}$ and

$\mathtt{magic\_sc^b(c)} \in \mathit{variant}^0_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M_{sc})$), and $\mathtt{magic\_sc^b(c_1)}$ (because $\mathtt{magic\_sc^b(c_1)}$:− $\mathtt{magic\_sc^b(c)}$. is a rule of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q}_{sc},\mathcal{P}_{sc}))$ and $\mathtt{magic\_sc^b(c)} \in \mathit{variant}^0_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M_{sc})$). Any other element of the sequence coincides with $\mathit{variant}^2_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M_{sc})$, and so also $\mathit{variant}^\infty_{\mathcal{Q}_{sc},\mathcal{P}_{sc}}(M_{sc})$. □

By definition, for a magic variant $\mathit{variant}^\infty_{\mathcal{Q},\mathcal{P}}(I)$ of an interpretation I with respect to $\mathcal{Q}$ and $\mathcal{P}$, $\mathit{variant}^\infty_{\mathcal{Q},\mathcal{P}}(I)|_{B_\mathcal{P}} \subseteq I$ holds. More interestingly, the magic variant of a stable model for $\mathcal{P}$ is in turn a stable model for $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$.

Example 3.22 The magic variant of M_{sc} with respect to $\mathcal{Q}_{sc}$ and $\mathcal{P}_{sc}$ (see Example 3.21) coincides with the interpretation M'_{sc} introduced in Example 3.15. From previous examples, we know that M_{sc} is a stable model of $\mathcal{P}_{sc}$, and M'_{sc} is a stable model of $\mathtt{DMS}(\mathcal{Q}_{sc},\mathcal{P}_{sc})$. □

The following two lemmas formalize the intuition above, with the latter being the counterpart of Lemma 3.18.

Lemma 3.23 *For each stable model M of $\mathcal{P}$, the magic variant $M' = \mathit{variant}^\infty_{\mathcal{Q},\mathcal{P}}(M)$ of M is a model of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$ with $M \supseteq M'|_{B_\mathcal{P}}$.*

Proof. As M' is the magic variant of the stable model M, we trivially have that $M \supseteq M'|_{B_\mathcal{P}}$ holds. We next show that M' is a model of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$. To this end, consider a rule in $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$ having the body true, that is, a rule obtained by removing the negative body literals from a rule $r'_g \in \mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))$ such that $B^-(r'_g) \cap M' = \emptyset$ and $B^+(r'_g) \subseteq M'$ hold. We have to show that $H(r'_g) \cap M' \neq \emptyset$.

In the case where r'_g is a magic rule, then $B^+(r'_g) \subseteq M'$ implies that the (only) atom in $H(r'_g)$ belongs to M' (by Definition 3.20). The only remaining (slightly more involved) case to be analyzed is where r'_g is a modified rule of the form

$$\begin{aligned} r'_g:\ & p(\bar{t}) \vee p_1(\bar{t}_1) \vee \cdots \vee p_n(\bar{t}_n) \text{ :− } \mathit{magic}(p^\alpha(\bar{t})), \mathit{magic}(p_1^{\alpha_1}(\bar{t}_1)), \ldots, \\ & \quad \mathit{magic}(p_n^{\alpha_n}(\bar{t}_n)), q_1(\bar{s}_1), \ldots, q_j(\bar{s}_j), \mathit{not}\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ \mathit{not}\ q_m(\bar{s}_m). \end{aligned}$$

In this case, we first apply as usual Lemma 3.13 in order to conclude the existence of a rule $r_g \in \mathit{Ground}(\mathcal{P})$ of the form

$$\begin{aligned} r_g:\ & p(\bar{t}) \vee p_1(\bar{t}_1) \vee \cdots \vee p_n(\bar{t}_n) \text{ :− } q_1(\bar{s}_1),\ \ldots,\ q_j(\bar{s}_j), \\ & \qquad \mathit{not}\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ \mathit{not}\ q_m(\bar{s}_m). \end{aligned}$$

Then, we claim that the following two properties hold:

- $B^-(r_g) \cap M = \emptyset;$ (6)

- $B^+(r_g) \subseteq M$. (7)

These properties are in fact what we just need to establish the result. Indeed, since M is a model of $Ground(\mathcal{P})^M$, (6) and (7) imply $H(r_g) \cap M \neq \emptyset$. So, we can recall that $H(r_g) = H(r'_g)$, and hence let $p_i(\bar{t}_i)$ be an atom in $H(r_g) \cap M = H(r'_g) \cap M$ and $magic(p_i^{\alpha_i}(\bar{t}_i))$ be its corresponding magic atom in $B^+(r'_g)$ ($i \in \{\epsilon, 1, \ldots, n\}$, where ϵ is the empty string). Since $B^+(r'_g) \subseteq M'$ (by hypothesis) and since $p_i(\bar{t}_i) \in M$, we can then conclude that $p_i(\bar{t}_i)$ is in M' as well by Definition 3.20. That is, $H(r'_g) \cap M' \neq \emptyset$.

Let now finalize the proof, by showing that the above properties actually hold.

Proof of (6). Consider a modified rule $r' \in \texttt{DMS}(\mathcal{Q}, \mathcal{P})$ such that $r'_g = r'\vartheta$ for a substitution ϑ:

$$r': \ p(\bar{t}') \lor p_1(\bar{t}'_1) \lor \cdots \lor p_n(\bar{t}'_n) \text{ :- } magic(p^{\alpha}(\bar{t}')), magic(p_1^{\alpha_1}(\bar{t}'_1)), \ldots,$$
$$magic(p_n^{\alpha_n}(\bar{t}'_n)), q_1(\bar{s}'_1), \ldots, q_j(\bar{s}'_j), not\ q_{j+1}(\bar{s}'_{j+1}), \ \ldots, \ not\ q_m(\bar{s}'_m).$$

and the rule $r \in \mathcal{P}$ from which r' is produced (such that $r_g = r\vartheta$):

$$r: \ p(\bar{t}') \lor p_1(\bar{t}'_1) \lor \cdots \lor p_n(\bar{t}'_n) \text{ :- } q_1(\bar{s}'_1), \ \ldots, \ q_j(\bar{s}'_j),$$
$$not\ q_{j+1}(\bar{s}'_{j+1}), \ \ldots, \ not\ q_m(\bar{s}'_m).$$

During the *Generation* step preceding the production of r', a magic rule r_i^* such that $H(r_i^*) = \{magic(q_i^{\beta_i}(\bar{s}'_i))\}$ has been produced for each $j+1 \leq i \leq m$ such that q_i is an IDB predicate. Hence, since the variables of r_i^* are a subset of the variables of r', the substitution ϑ can be used to map r_i^* to a ground rule $r_{i,g}^* = r_i^*\vartheta$ such that $H(r_{i,g}^*) = \{magic(q_i^{\beta_i}(\bar{t}_i))\}$ and $B^+(r_{i,g}^*) \subseteq B^+(r'_g)$ (we recall that magic rules have empty negative body). Now, since $B^+(r'_g) \subseteq M'$, we can conclude that $B^+(r_{i,g}^*)$ is in turn contained in M'. Thus, by the construction of M', the head of $r_{i,g}^*$ must be true with respect to M', that is, $magic(q_i^{\beta_i}(\bar{t}_i)) \in M'$ holds for each $j+1 \leq i \leq m$ such that q_i is an IDB predicate. So, if some (IDB) atom $q_i(\bar{s}_i) \in B^-(r_g)$ belongs to M, by Definition 3.20 we can conclude that $q_i(\bar{s}_i) \in M'$, which contradicts the assumption that $B^-(r'_g) \cap M' = \emptyset$ (we recall that $B^-(r_g) = B^-(r'_g)$). This proves that IDB predicates in $B^-(r_g)$ do not occur in M. The same trivially holds for EDB predicates too, since $B^-(r_g) \cap M' = B^-(r'_g) \cap M' = \emptyset$ and $M' \supseteq EDB(\mathcal{P})$ (by the definition of magic variant).

Proof of (7). The equation straightforwardly follows from the fact that $B^+(r_g) = B^+(r'_g)|_{B_{\mathcal{P}}}$, and since $M \supseteq M'|_{B_{\mathcal{P}}}$ and $B^+(r'_g) \subseteq M'$ hold by the construction of M' and by the initial hypothesis on the choice of r'_g, respectively. $\square$

Lemma 3.24 *For each stable model M of $\mathcal{P}$, there is a stable model M' of* $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ *(which is the magic variant of M) such that $M \supseteq M'|_{B_\mathcal{P}}$.*

Proof. After Lemma 3.23, we can show that $M' = \mathit{variant}^{\infty}_{\mathcal{Q},\mathcal{P}}(M)$ is also minimal over all the models of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$. Let $N' \subseteq M'$ be a minimal model of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$. We prove by induction on the definition of the magic variant that M' is in turn contained in N'. The base case (i.e., $\mathit{variant}^{0}_{\mathcal{Q},\mathcal{P}}(M) \subseteq N'$) is clearly true, since $\mathit{variant}^{0}_{\mathcal{Q},\mathcal{P}}(M)$ contains only EDB facts. Suppose $\mathit{variant}^{i}_{\mathcal{Q},\mathcal{P}}(M) \subseteq N'$ in order to prove that $\mathit{variant}^{i+1}_{\mathcal{Q},\mathcal{P}}(M) \subseteq N'$ holds as well.

While considering an atom in $\mathit{variant}^{i+1}_{\mathcal{Q},\mathcal{P}}(M) \setminus \mathit{variant}^{i}_{\mathcal{Q},\mathcal{P}}(M)$, we distinguish two cases:

(a) For a magic atom $\mathit{magic}(p^\alpha(\bar{t}))$ in $\mathit{variant}^{i+1}_{\mathcal{Q},\mathcal{P}}(M) \setminus \mathit{variant}^{i}_{\mathcal{Q},\mathcal{P}}(M)$, by Definition 3.20 there must be a rule $r^*_g \in \mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))$ having $H(r^*_g) = \{\mathit{magic}(p^\alpha(\bar{t}))\}$ and $B^+(r^*_g) \subseteq \mathit{variant}^{i}_{\mathcal{Q},\mathcal{P}}(M)$ (we recall that magic rules have empty negative body and so $r^*_g \in \mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$ holds). We can then conclude that $B^+(r^*_g) \subseteq N'$ holds by the induction hypothesis and so $\mathit{magic}(p^\alpha(\bar{t})) \in N'$ (because N' is a model of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$).

(b) For a standard atom $p(\bar{t})$ in $\mathit{variant}^{i+1}_{\mathcal{Q},\mathcal{P}}(M) \setminus \mathit{variant}^{i}_{\mathcal{Q},\mathcal{P}}(M)$, by Definition 3.20 there is a binding α such that $\mathit{magic}(p^\alpha(\bar{t})) \in \mathit{variant}^{i}_{\mathcal{Q},\mathcal{P}}(M)$ and the atom $p(\bar{t})$ belongs to M. Assume for the sake of contradiction that $p(\bar{t}) \notin N'$. Since M' is a model of $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ and N' is a model of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$, we can compute the set $\mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ as introduced in Section 3.3.1 and note, in particular, that $p(\bar{t}) \in \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ holds (by definition). Moreover, by Proposition 3.17, $\mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ is an unfounded set for $\mathcal{P}$ with respect to $\langle M'|_{B_\mathcal{P}}, B_\mathcal{P}\rangle$. In addition, $M \supseteq M'|_{B_\mathcal{P}}$ holds by Definition 3.20. Thus, M is a stable model for $\mathcal{P}$ such that $M \supseteq M'|_{B_\mathcal{P}}$, and we can hence apply Theorem 3.11 in order to conclude that $M \cap \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(N') = \emptyset$. The latter is in contradiction with $p(\bar{t}) \in \mathit{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(N')$ and $p(\bar{t}) \in M$. Hence, $p(\bar{t}) \in N'$. □

We can then prove the correspondence of stable models with respect to queries.

Theorem 3.25 *Let $\mathcal{Q}$ be a query for a Datalog$^{\vee,\neg_s}$ program $\mathcal{P}$. Then, for each stable model M of $\mathcal{P}$, there is a stable model M' of* $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ *(which is the magic variant of M) such that $M'|_\mathcal{Q} = M|_\mathcal{Q}$.*

Proof. Let M be a stable model of $\mathcal{P}$ and $M' = \mathit{variant}^{\infty}_{\mathcal{Q},\mathcal{P}}(M)$ its magic variant. Because of Lemma 3.24, M' is a stable model of $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ such that $M \supseteq M'|_{B_\mathcal{P}}$. Thus, we trivially have that $M|_\mathcal{Q} \supseteq M'|_\mathcal{Q}$ holds. We now show the reverse inclusion.

Since M' is a stable model of $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$, we can determine the set $killed^{M'}_{\mathcal{Q},\mathcal{P}}(M')$ as defined in Section 3.3.1. Hence, by Definition 3.14 we can conclude that (a) $B_{\mathcal{P}}|_{\mathcal{Q}} \setminus M' \subseteq killed^{M'}_{\mathcal{Q},\mathcal{P}}(M')$ because M' contains the magic seed by construction (we recall that $B_{\mathcal{P}}|_{\mathcal{Q}}$ denotes the ground instances of $\mathcal{Q}$). Moreover, since M is a stable model of $\mathcal{P}$ with $M \supseteq M'|_{B_{\mathcal{P}}}$ and $killed^{M'}_{\mathcal{Q},\mathcal{P}}(M')$ is an unfounded set for $\mathcal{P}$ with respect to $\langle M'|_{B_{\mathcal{P}}}, B_{\mathcal{P}}\rangle$ by Proposition 3.17, we can conclude that (b) $M \cap killed^{M'}_{\mathcal{Q},\mathcal{P}}(M') = \emptyset$ by Theorem 3.11. Thus, by combining (a) and (b) we obtain that $(B_{\mathcal{P}}|_{\mathcal{Q}} \setminus M') \cap M = \emptyset$, which is equivalent to $M|_{\mathcal{Q}} \subseteq M'|_{\mathcal{Q}}$. □

Finally, we show the correctness of the Magic Set method with respect to query answering, that is, we prove that the original and rewritten programs provide the same answers for the input query on all possible EDBs.

Theorem 3.26 *Let $\mathcal{P}$ be a Datalog$^{\vee,\neg_s}$ program, and let $\mathcal{Q}$ be a query. Then* $\mathtt{DMS}(\mathcal{Q},\mathcal{P}) \equiv^b_{\mathcal{Q}} \mathcal{P}$ *and* $\mathtt{DMS}(\mathcal{Q},\mathcal{P}) \equiv^c_{\mathcal{Q}} \mathcal{P}$ *hold.*

Proof. We want to show that, for any set of facts $\mathcal{F}$ defined over the EDB predicates of $\mathcal{P}$ (and $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$), $Ans_b(\mathcal{Q}, \mathtt{DMS}(\mathcal{Q},\mathcal{P}) \cup \mathcal{F}) = Ans_b(\mathcal{Q}, \mathcal{P} \cup \mathcal{F})$ and $Ans_c(\mathcal{Q}, \mathtt{DMS}(\mathcal{Q},\mathcal{P}) \cup \mathcal{F}) = Ans_c(\mathcal{Q}, \mathcal{P} \cup \mathcal{F})$ hold. We first observe that the Magic Set rewriting does not depend on EDB facts; thus, $\mathtt{DMS}(\mathcal{Q},\mathcal{P}) \cup \mathcal{F} = \mathtt{DMS}(\mathcal{Q},\mathcal{P} \cup \mathcal{F})$ holds. Moreover, note that Datalog$^{\vee,\neg_s}$ programs always have stable models. Therefore, as a direct consequence of Theorem 3.19 and Theorem 3.25, we can conclude $Ans_b(\mathcal{Q}, \mathtt{DMS}(\mathcal{Q},\mathcal{P} \cup \mathcal{F})) = Ans_b(\mathcal{Q}, \mathcal{P} \cup \mathcal{F})$ and $Ans_c(\mathcal{Q}, \mathtt{DMS}(\mathcal{Q},\mathcal{P} \cup \mathcal{F})) = Ans_c(\mathcal{Q}, \mathcal{P} \cup \mathcal{F})$. □

3.4 Magic Sets for Stratified Datalog Programs without Disjunction

Stratified Datalog programs without disjunction have exactly one stable model [29]. However, the Magic Set transformation can introduce new dependencies between predicates, possibly resulting in unstratified programs (we refer to the analysis in [38]). Clearly, original and rewritten programs agree on the query, as proved in the previous section, but the question whether the rewritten program admits a unique stable model is also important. In fact, for programs having the unique stable model property, brave and cautious reasoning coincide and a solver can immediately answer the query after the first (and unique) stable model is found. The following theorem states that the rewritten program of a stratified program indeed has a unique stable model.

Theorem 3.27 *Let $\mathcal{P}$ be a disjunction-free Datalog program with stratified negation and $\mathcal{Q}$ a query. Then* $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ *has a unique stable model.*

Proof. Let M be the unique stable model of $\mathcal{P}$, and $M' = variant^{\infty}_{\mathcal{Q},\mathcal{P}}(M)$ its magic variant as presented in Definition 3.20. By Lemma 3.24 we already know that M' is a stable model of $\texttt{DMS}(\mathcal{Q},\mathcal{P})$. We now show that any stable model N' of $\texttt{DMS}(\mathcal{Q},\mathcal{P})$ contains M' by induction on the structure of M'. The base case ($variant^0_{\mathcal{Q},\mathcal{P}}(M) \subseteq N'$) is clearly true, since $variant^0_{\mathcal{Q},\mathcal{P}}(M)$ contains only EDB facts. Suppose $variant^i_{\mathcal{Q},\mathcal{P}}(M) \subseteq N'$ in order to prove that $variant^{i+1}_{\mathcal{Q},\mathcal{P}}(M) \subseteq N'$ holds as well. Thus, while considering an atom in $variant^{i+1}_{\mathcal{Q},\mathcal{P}}(M) \setminus variant^i_{\mathcal{Q},\mathcal{P}}(M)$, two cases are possible:

(1) For a magic atom $magic(p^\alpha(\bar{t}))$ in $variant^{i+1}_{\mathcal{Q},\mathcal{P}}(M) \setminus variant^i_{\mathcal{Q},\mathcal{P}}(M)$, by Definition 3.20 there must be a rule $r^*_g \in Ground(\texttt{DMS}(\mathcal{Q},\mathcal{P}))$ having $H(r^*_g) = \{magic(p^\alpha(\bar{t}))\}$ and $B^+(r^*_g) \subseteq variant^i_{\mathcal{Q},\mathcal{P}}(M)$ (we recall that magic rules have empty negative bodies and so $r^*_g \in Ground(\texttt{DMS}(\mathcal{Q},\mathcal{P}))^{N'}$ holds). We can then conclude that $B^+(r^*_g) \subseteq N'$ holds by the induction hypothesis and so $magic(p^\alpha(\bar{t})) \in N'$ (because N' is a model of $Ground(\texttt{DMS}(\mathcal{Q},\mathcal{P}))^{N'}$).

(2) For a standard atom $p(\bar{t})$ in $variant^{i+1}_{\mathcal{Q},\mathcal{P}}(M) \setminus variant^i_{\mathcal{Q},\mathcal{P}}(M)$, by Definition 3.20 there is a binding α such that $magic(p^\alpha(\bar{t})) \in variant^i_{\mathcal{Q},\mathcal{P}}(M)$ and the atom $p(\bar{t})$ belongs to M. Assume for the sake of contradiction that $p(\bar{t}) \notin N'$. Since N' is a stable model of $\texttt{DMS}(\mathcal{Q},\mathcal{P})$, we can compute the set $killed^{N'}_{\mathcal{Q},\mathcal{P}}(N')$ as introduced in Section 3.3.1 and note, in particular, that $p(\bar{t}) \in killed^{N'}_{\mathcal{Q},\mathcal{P}}(N')$ holds, by definition. Moreover, by Proposition 3.17, $killed^{N'}_{\mathcal{Q},\mathcal{P}}(N')$ is an unfounded set for $\mathcal{P}$ with respect to $\langle N'|_{B_\mathcal{P}}, B_\mathcal{P}\rangle$. In addition, by Lemma 3.25 there is a stable model N of $\mathcal{P}$ such that $N \supseteq N'|_{B_\mathcal{P}}$, which would mean that $p(\bar{t}) \notin N$ holds. Hence, we can conclude that N and M are two different stable models of $\mathcal{P}$, obtaining a contradiction, as $\mathcal{P}$ has a unique stable model.

Since stable models are incomparable with respect to containment, $M' \subseteq N'$ implies $M' = N'$. Hence, M' is the unique stable model of $\texttt{DMS}(\mathcal{Q},\mathcal{P})$. $\Box$

4 Implementation

The Dynamic Magic Set method (DMS) has been implemented and integrated into the core of the DLV [43] system. In this section, we shall first briefly describe the architecture of the system and its usage. We then briefly present an optimization for eliminating redundant rules, which are sometimes introduced during the Magic Set rewriting.

4.1 System Architecture and Usage

We have created a prototype system by implementing the Magic Set technique described in Section 3 inside DLV, as shown in the architecture reported in Figure 6. DLV supports both brave and cautious reasoning, and for a completely ground query it can be also used for computing all stable models in which the query is true. DLV performs brave reasoning if invoked with the command-line option `-FB`, while `-FC` indicates cautious reasoning.

In our prototype, the `DMS` algorithm is applied automatically by default when the user invokes DLV with `-FB` or `-FC` together with a (partially) bound query. Magic Sets are not applied by default if the query does not contain any constant. The user can modify this default behavior by specifying the command-line options `-ODMS` (for applying Magic Sets) or `-ODMS-` (for disabling Magic Sets).

If a completely bound query is specified, DLV can print the magic variant of the stable model (not displaying magic predicates), which witnesses the truth (for brave reasoning) or the falsity (for cautious reasoning) of the query, by specifying the command-line option `--print-model`.

Within DLV, `DMS` is applied immediately after parsing the program and the query by the *Magic Set Rewriter* module. The rewritten (and optimized as described in Section 4.2) program is then processed by the *Intelligent Grounding* module and the *Model Generator* module using the implementation of DLV. The only other modification is for the output and its filtering: For ground queries, the witnessing stable model is no longer printed by default, but only if `--print-model` is specified, in which case the magic predicates are omitted from the output.

The SIPS schema[7] implemented in the prototype is as follows: For a rule r, head atom $p(\bar{t})$ and binding α, $\prec_r^{p^\alpha(\bar{t})}$ satisfies the conditions of Definition 3.3, in particular $p(\bar{t}) \prec_r^{p^\alpha(\bar{t})} q(\bar{s})$ holds for all $q(\bar{s}) \neq p(\bar{t})$ in r, and $q(\bar{s}) \not\prec_r^{p^\alpha(\bar{t})} b(\bar{z})$ holds for all head or negative body atoms $q(s) \neq p(\bar{t})$ and any atom $b(\bar{z})$ in r. Moreover, all the positive body literals of r form a chain in $\prec_r^{p^\alpha(\bar{t})}$. This chain is constructed by iteratively inserting those atoms containing most bound arguments (considering α and also the partially formed chain and $f_r^{p^\alpha(\bar{t})}$) into the chain. Among the atoms with most bindings an arbitrary processing order (usually the order appearing in the original rule body) is used. Furthermore, $f_r^{p^\alpha(\bar{t})}(q(\bar{s})) = X$ holds if and only if $q(\bar{s})$ belongs to the positive body of r, has at least one bound argument and X occurs in $\bar{s}$.

[7] Since technically a SIPS has a definition for every single rule, implementations use a schema for creating the SIPS for a given rule.

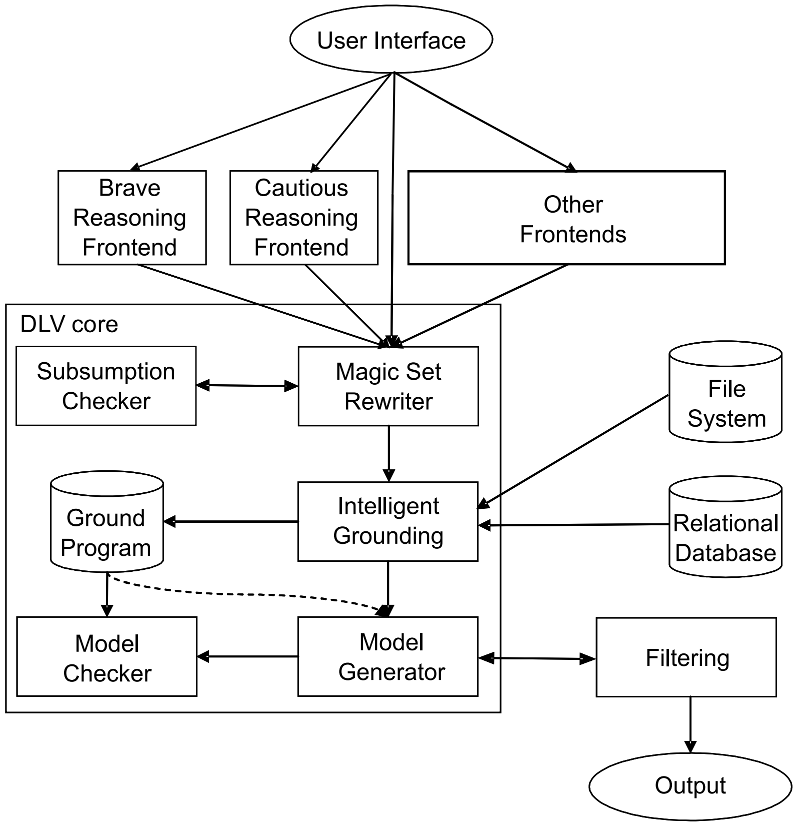


Fig. 6. Prototype system architecture

This means that apart from the head atom via which the rule is adorned, only positive body atoms can yield variable bindings and only if at least one of their arguments is bound, but both atoms with EDB and IDB predicates can do so. Moreover, atoms with more bound arguments will be processed before those with fewer bound arguments.

Note that in this work we did not study the impact of trying different SIPS schemas, as we wanted to focus on showing the impact that our technique can have, rather than fine-tuning its parameters. While we believe that the SIPS schema employed is well-motivated, there probably is quite a bit of room for improvement, which we leave for future work.

An executable of the DLV system supporting the Magic Set optimization is available at `http://www.dlvsystem.com/magic/`.

4.2 Dealing with Redundant Rules

Even though our rewriting algorithm keeps the amount of generated rules low, it might happen that some redundant rules are generated when adorning disjunctive rules, thereby somewhat deteriorating the optimization effort. For instance, in Example 3.6 the first two modified rules are semantically equivalent, and this might happen even if the two head predicates differ. In general not only duplicated rules might be created, but also rules which are logically subsumed by other rules in the program. Let us first give the definition of subsumption for Datalog$^{\vee,\neg_s}$ rules.

Definition 4.1 *Let $\mathcal{P}$ be a Datalog$^{\vee,\neg}$ program, and let r and r' be two rules of $\mathcal{P}$. Then, r is* subsumed *by r' (denoted by $r \sqsubseteq r'$) if there exists a substitution ϑ for the variables of r', such that $H(r')\vartheta \subseteq H(r)$ and $B(r')\vartheta \subseteq B(r)$. A rule r is* redundant *if there exists a rule r' such that $r \sqsubseteq r'$.*

Ideally, a Magic Set rewriting algorithm should be capable of identifying all the possible redundant rules and removing them from the output. Unfortunately, this approach is unlikely to be feasible in polynomial time, given that subsumption checking on first-order expressions is NP-complete (problem [LO18] in [27]).

Thus, in order to identify whether a rule r produced during the Magic Set transformation is redundant, we pragmatically apply a greedy subsumption algorithm in our implementation, for checking whether $r \sqsubseteq r'$ holds for some rule r'. In particular, the employed heuristics aims at building the substitution ϑ (as in Definition 4.1) by iteratively choosing an atom $p(\bar{t})$ (which is not yet processed) from r' and by matching it (if possible) with some atom of r. The greedy approach prefers those atoms of r' with the maximum number of variables not yet matched.

To turn on subsumption checking (applied once after the Magic Set rewriting), DLV has to be invoked with the command-line option `-ODMS+`.

5 Experiments on Standard Benchmarks

We performed several experiments for assessing the effectiveness of the proposed technique. In this section we present the results obtained on various standard benchmarks, most of which have been directly adopted from the literature. Further experiments on an application scenario using real-world data will be discussed in detail in Section 6. We also refer to [45,54] that contain performance evaluations involving `DMS`; in [45] *DLV* with `DMS` was tested on Semantic Web reasoning tasks and confronted with a heterogeneous set of systems, in [54] the system KAON2, which includes a version of `DMS`, is confronted against other ontology systems. In both publications the impact of magic sets is stated explicitly.

5.1 Compared Methods, Benchmark Problems and Data

In order to evaluate the impact of the proposed method, we have compared DMS (using the SIPS defined outlined in Section 4) both with the traditional DLV evaluation without *Magic Sets* and with the SMS method proposed in [33]. Concerning SMS, we were not able to obtain an implementation, and have therefore performed the rewriting manually. As a consequence, the runtime measures obtained for SMS do not contain the time needed for rewriting, while it is included for DMS.

For the comparison, we consider the following benchmark problems. The first three of them had been already used to assess SMS in [33], to which we refer for details:

- *Simple Path:* Given a directed graph G and two nodes a and b, does there exist a unique path connecting a to b in G? The instances are encoded by facts $\texttt{edge}(\texttt{v}_1, \texttt{v}_2)$ for each arc (v_1, v_2) in G, while the problem itself is encoded by the program [8]

$$
\begin{aligned}
&\texttt{sp(X,X)} \ \vee \ \texttt{not\_sp(X,X)} \ \texttt{:-} \ \texttt{edge(X,Y)}.\\
&\texttt{sp(X,Y)} \ \vee \ \texttt{not\_sp(X,Y)} \ \texttt{:-} \ \texttt{sp(X,Z)}, \ \texttt{edge(Z,Y)}.\\
&\texttt{path(X,Y)} \ \texttt{:-} \ \texttt{sp(X,Y)}.\\
&\texttt{path(X,Y)} \ \texttt{:-} \ \texttt{not\_sp(X,Y)}.\\
&\texttt{not\_sp(X,Z)} \ \texttt{:-} \ \texttt{path}(\texttt{X},\texttt{Y}_1), \ \texttt{path}(\texttt{X},\texttt{Y}_2), \ \texttt{Y}_1 <> \texttt{Y}_2,\\
&\qquad\qquad\qquad \texttt{edge}(\texttt{Y}_1,\texttt{Z}), \texttt{edge}(\texttt{Y}_2,\texttt{Z}).
\end{aligned}
$$

  with the query $\texttt{sp(a,b)}$. The structure of the graph, which is the same as the one reported in [33], consists of a square matrix of nodes connected as shown in Figure 7, and the instances have been generated by varying of the number of nodes.
- *Related:* Given a genealogy graph storing information about relationships (father/brother) among people and given two people p_1 and p_2, is p_1 an ancestor of p_2? The instances are encoded by facts $\texttt{related}(\texttt{p}_1, \texttt{p}_2)$ when p_1 is known to be related to p_2, that is, when p_1 is the father or a brother of

[8] The first rule of the program models that for each node X of G, a unique path connecting X with itself can either exist or not.

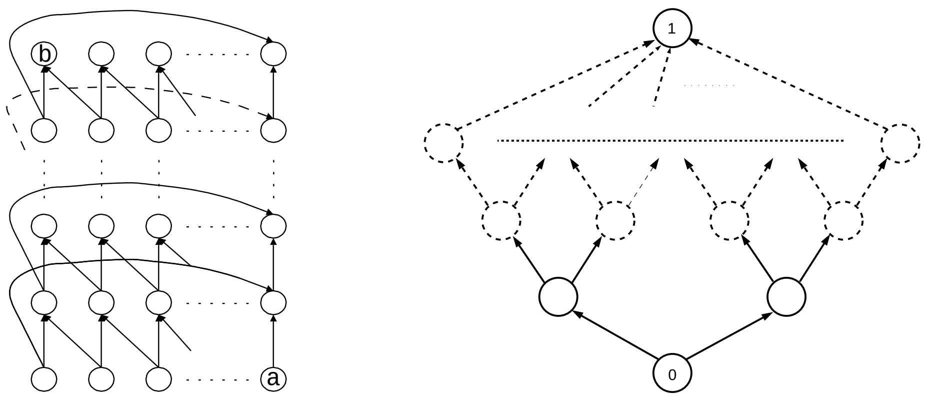


Fig. 7. Instances structure of *Simple Path* and *Related* (left) and of *Conformant Plan Checking* (right)

p_2. The problem can be encoded by the program

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

and the query is `ancestor(p1,p2)`. The structure of the "genealogy" graph is the same as the one presented in [33] and coincides with the one used for testing *Simple Path*. Also in this case, the instances are generated by varying the number of nodes (thus the number of persons in the genealogy) of the graph.

- *Strategic Companies:* This is a slight variant of the problem domain used in the running example. The description here is of the problem as posed in the Third ASP Competition. We consider a collection C of companies, where each company produces some goods in a set G and each company $c_i \in C$ is controlled by a set of owner companies $O_i \subseteq C$. A subset of the companies $C' \subset C$ is a *strategic set* if it is a minimal set of companies producing all the goods in G, such that if $O_i \subseteq C'$ for some $i = 1, \ldots, m$ then $c_i \in C'$ must hold. As in the Second Answer Set Competition,[9] we assume that each product is produced by at most four companies, and that each company is controlled by at most four companies (the complexity of the problem under these restrictions is as hard as without them). Given two distinct companies $c_i, c_j \in C$, is there a strategic set of C which contains both c_i and c_j? The instances are encoded by facts `produced_by(p,c1,c2,c3,c4)` when product p is produced by companies c_1, c_2, c_3, and c_4; if p is produced by fewer than four companies (but at least one), then c_1, c_2, c_3, c_4 contains repetitions of companies. Moreover, facts `controlled_by(c,c1,c2,c3,c4)` represent that company c is controlled by companies c_1, c_2, c_3, and c_4; again, if c is controlled by fewer than four companies, then c_1, c_2, c_3, c_4 contains repetitions.

[9] `http://www.cs.kuleuven.be/~dtai/events/ASP-competition/index.shtml`

The problem can be encoded by the program

$$\texttt{st}(\texttt{C}_1) \lor \texttt{st}(\texttt{C}_2) \lor \texttt{st}(\texttt{C}_3) \lor \texttt{st}(\texttt{C}_4) :- \texttt{produced\_by}(\texttt{P}, \texttt{C}_1, \texttt{C}_2, \texttt{C}_3, \texttt{C}_4).$$

$$\texttt{st}(\texttt{C}) :- \texttt{controlled\_by}(\texttt{C}, \texttt{C}_1, \texttt{C}_2, \texttt{C}_3, \texttt{C}_4), \texttt{st}(\texttt{C}_1), \texttt{st}(\texttt{C}_2), \texttt{st}(\texttt{C}_3), \texttt{st}(\texttt{C}_4).$$

with the query $\texttt{st}(\texttt{c}_\texttt{i}),\ \texttt{st}(\texttt{c}_\texttt{j})$. While the language presented in the previous sections allowed only for one atom in a query for simplicity, the implementation in DLV allows for a conjunction in a query; it is easy to see that a conjunctive query can be emulated by a rule with the conjunction in the body and an atom with a new predicate in the head, which contains all body arguments, and finally replacing the query conjunction with this atom. In this case this would mean adding a rule $\texttt{q}(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j}) :- \texttt{st}(\texttt{c}_\texttt{i}),\ \texttt{st}(\texttt{c}_\texttt{j})$ and replacing the query by $\texttt{q}(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j})$. For this benchmark we used the instances submitted for the Second Answer Set Competition.

- *Conformant Plan Checking:* In addition, we have included a benchmark problem, which highlights the fact that our Magic Set technique can yield improvements not only for the grounding, but also for the model generation phase, as discussed in Section 7. This problem is inspired by a setting in planning, in particular testing whether a given plan is conformant with respect to a state transition diagram [30]. Such a diagram is essentially a directed graph formed of nodes representing states, and in which arcs are labeled by actions, meaning that executing the action in the source state will lead to the target state. In the considered setting nondeterminism is allowed, that is, executing an action in one state might lead nondeterministically to one of several successor states. A plan is a sequence of actions, and it is conformant with respect to a given initial state and a goal state if each possible execution of the action sequence leads to the goal state.

  In our benchmark, we assume that the action selection process has already been done, thus having reduced the state transition diagram to those transitions that actually occur when executing the given plan. Furthermore we assume that there are exactly two possible non-goal successor states for any given state. This can also be viewed as whether all outgoing paths of a node in a directed graph reach a particular confluence node. We encoded instances by facts $\texttt{ptrans}(\texttt{s}_0, \texttt{s}_1, \texttt{s}_2)$ meaning that one of states $\texttt{s}_1$ and $\texttt{s}_2$ will be reached in the plan execution starting from $\texttt{s}_0$. The problem is encoded using

$$\texttt{trans}(\texttt{X}, \texttt{Y}) \lor \texttt{trans}(\texttt{X}, \texttt{Z}) :- \texttt{ptrans}(\texttt{X}, \texttt{Y}, \texttt{Z}).$$

$$\texttt{reach}(\texttt{X}, \texttt{Y}) :- \texttt{trans}(\texttt{X}, \texttt{Y}).$$

$$\texttt{reach}(\texttt{X}, \texttt{Y}) :- \texttt{reach}(\texttt{X}, \texttt{Z}),\ \texttt{trans}(\texttt{Z}, \texttt{Y}).$$

and the query $\texttt{reach}(0, 1)$, where 0 is the initial state and 1 the goal state. If the query is cautiously true, the plan is conformant. The transition graphs

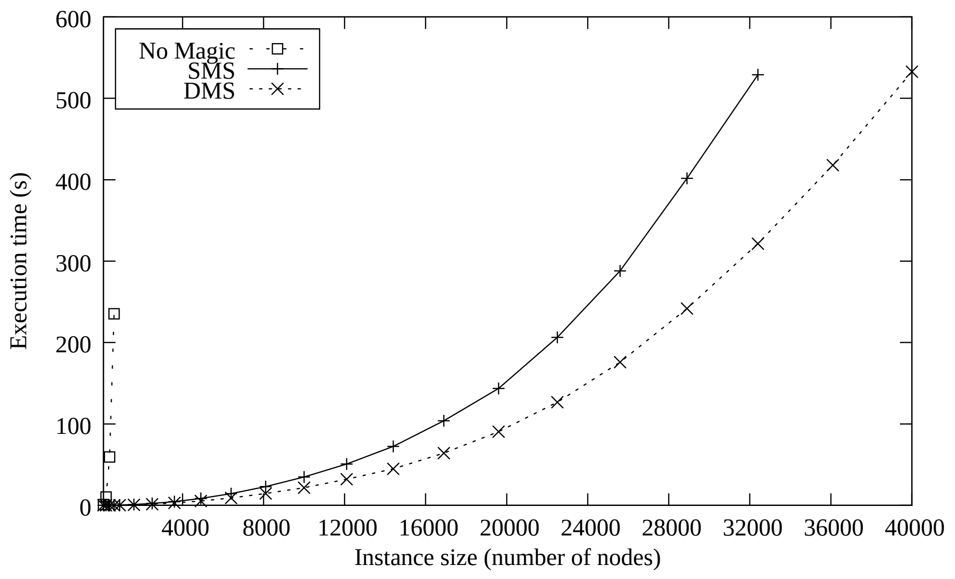


Fig. 8. *Simple Path:* Average execution time

in our experiments have the shape of a binary tree rooted in state 0, and from each leaf there is an arc to state 1, as depicted in Figure 7.

In addition, we have performed further experiments on an application scenario modeled from real-world data for answering user queries in a data integration setting. These latter experiments will be discussed in more detail in Section 6.

5.2 Results and Discussion

The experiments have been performed on a 3GHz Intel® Xeon® processor system with 4GB RAM under the Debian 4.0 operating system with a GNU/Linux 2.6.23 kernel. The DLV prototype used has been compiled using GCC 4.3.3. For each instance, we have allowed a maximum running time of 600 seconds (10 minutes) and a maximum memory usage of 3GB.

On all considered problems, `DMS` outperformed `SMS`, even if `SMS` does not include the rewriting time, as discussed in Section 5.1. Let us analyze the results for each problem in more detail.

The results for *Simple Path* are reported in Figure 8. DLV without Magic Sets solves only the smallest instances, with a very steep increase in execution time. `SMS` does better than DLV, but scales much worse than `DMS`. The difference between `SMS` and `DMS` is mostly due to the grounding of the additional predicates that `SMS` introduces.

Figure 9 reports the results for *Related*. Compared to Simple Path, DLV without Magic Sets exhibits an even steeper increase in runtime, while in contrast

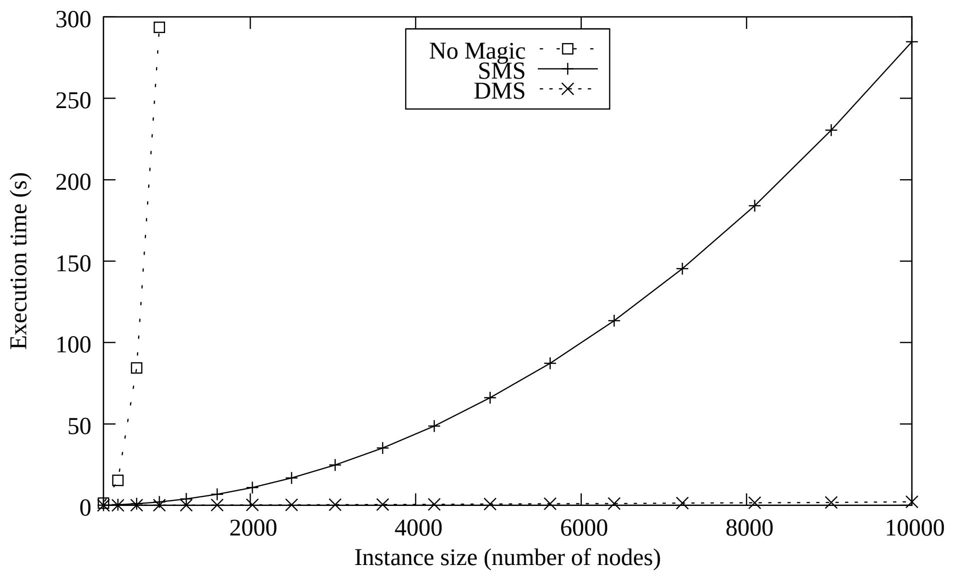


Fig. 9. *Related:* Average execution time

both SMS and DMS scale better than on Simple Path. Comparing SMS and DMS, we note that DMS appears to have an exponential speedup over SMS. In this case, the computational gain of DMS over SMS is due to the dynamic optimization of the model search phase resulting from our Magic Sets definition. This aspect is better highlighted by the *Conformant Plan Checking* benchmark, and will be discussed later in this section.

For *Strategic Companies*, we report the results in Figure 10 as a bar diagram, because the instances do not have a uniform structure. The instances are, however, ordered by size. Also here, DLV without Magic Sets is clearly the least efficient of the tested systems, resolving only the smallest two instances in the allotted time (600 seconds). Concerning the other systems, SMS and DMS essentially show equal performance. In fact, the situation here is quite different to Simple Path and Related, because grounding the program produced by the Magic Set rewriting takes only a negligible amount of time for SMS and DMS. For this benchmark the important feature is reducing the ground program to the part which is relevant for the query, and we could verify that the ground programs produced by SMS and DMS are precisely the same.

Finally, the results for *Conformant Plan Checking* are shown in Figure 11. While DLV shows a similar behavior as for Simple Path and Related, here also SMS does not scale well at all, and in fact DMS appears to have an exponential speedup over SMS. There is a precise reason for this: While the Magic Set rewriting of SMS always creates a deterministic program defining the magic predicates, this is not true for DMS. As a consequence, all magic predicates are completely evaluated during the grounding phase of DLV for SMS, while for DMS this is not the case. At the first glance, this may seem like a disadvantage of DMS, as one might believe that the ground program becomes larger.

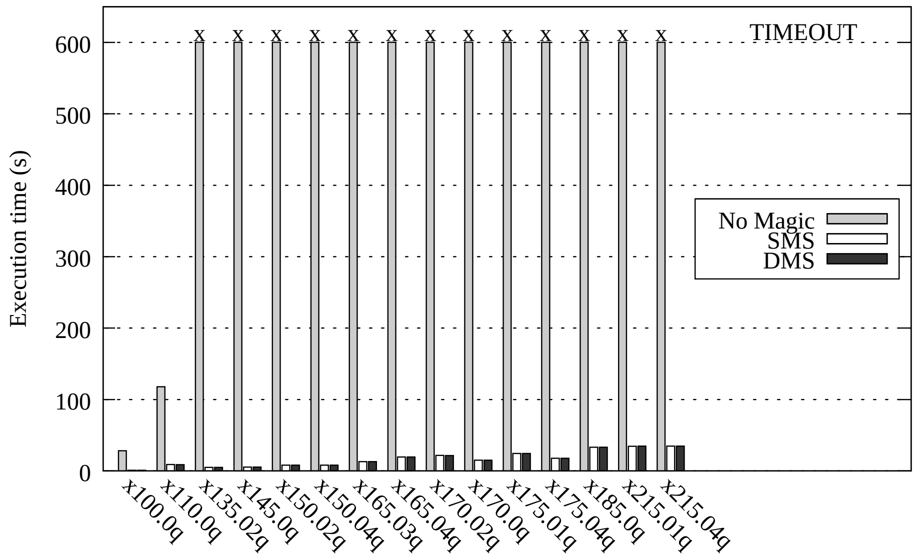


Fig. 10. *Strategic Companies:* Average execution time

However, it is actually a big advantage of DMS, because it offers a more precise identification of the relevant part of the program. Roughly speaking, whatever SMS identifies as relevant for the query will also be identified as relevant in DMS, but DMS can also include nondeterministic relevance information, which SMS cannot. This means that in DMS Magic Sets can be exploited also during the nondeterministic search phase of DLV, dynamically disabling parts of the ground program. In particular, after having made some choices, parts of the program may no longer be relevant to the query, but only because of these choices, and the magic atoms present in the ground program can render these parts satisfied, which means that they will no longer be considered in this part of the search. SMS cannot induce any behavior like this and its effect is limited to the grounding phase of DLV, which can make a huge difference, as evidenced by *Conformant Plan Checking*.

5.3 Experimenting DMS with other Disjunctive Datalog Systems

In order to assess the effectiveness of DMS on other systems than DLV, we tested the grounder Gringo [28] with the following solvers: ClaspD [21], Cmodels [46], GnT1 and GnT2 [37]. ClaspD is based on advanced Boolean constraint solving techniques, featuring backjumping and conflict-driven learning. Cmodels is based on the definition of program completion and loop formula for disjunctive programs [40,47], and uses a SAT solver for generating candidate solutions and testing them. GnT1 is based on Smodels [61], a system handling Datalog programs with unstratified negation (normal programs): A disjunctive program is translated into a normal program, the stable models of which are computed by Smodels and represent stable model candidates of the orig-

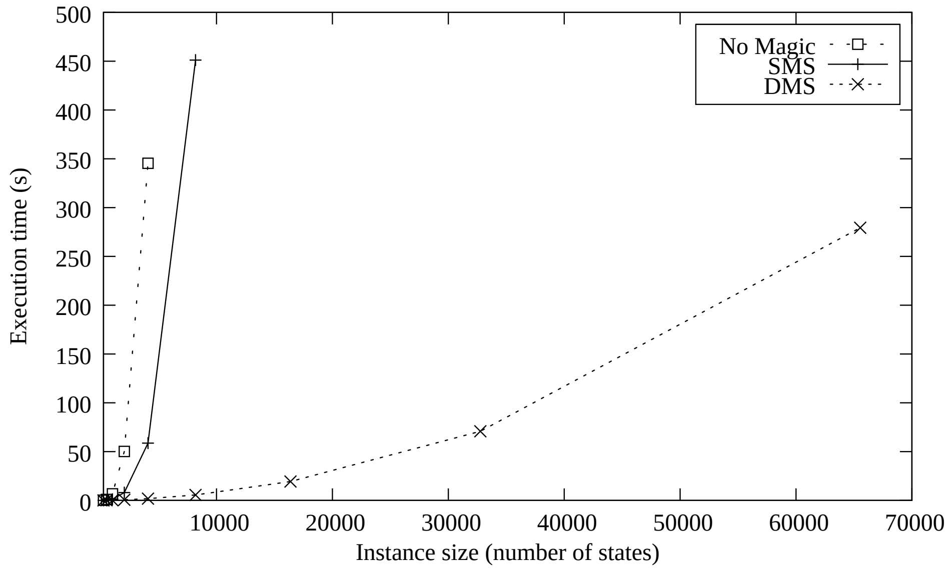


Fig. 11. *Conformant Plan Checking:* Average execution time

inal program. Each of these candidates is then checked to be a stable model of the original program by invoking Smodels on a second normal program. GnT2 is a variant of GnT1 in which the number of candidates produced by the first normal program is reduced by means of additional rules that discard unsupported models, i.e., models containing some atom a for which there is no rule r such that $B(r)$ is true and a is the only true atom in $H(r)$.

All of the benchmarks presented in the previous section were tested on these systems. Since DMS is not implemented in these systems, rewritten programs were produced by DLV during the preparation of the experiment. We recall that DMS does not depend on EDB relations and point out that DLV computes rewritten programs for the considered encodings in 1-2 hundredths of a second. The results of our experiment are reported in Figures 12–16. In general, we tried use a consistent scales in the graphs in order to ease comparability. However, for some graphs we chose a different scale in order to keep them readable for the main purpose (comparing performances with and without DMS), and we mention this explicitly in the accompanying text.

Concerning *Simple Path*, the advantages of DMS over the unoptimized encoding are evident on all tested systems. In fact, as shown in Figure 12, without DMS all tested systems did not answered in the allotted time (600 seconds) on instances with more than 400 nodes (900 for Cmodels). On the other hand, all of the instances considered in the benchmark (up to 40 thousands of nodes) were solved by all tested solvers with the DMS encoding. We also observe that with DMS the tested systems are faster than DLV in this benchmark, which is a clear indication of the optimization potential that can be provided to these systems by our Magic Set technique.

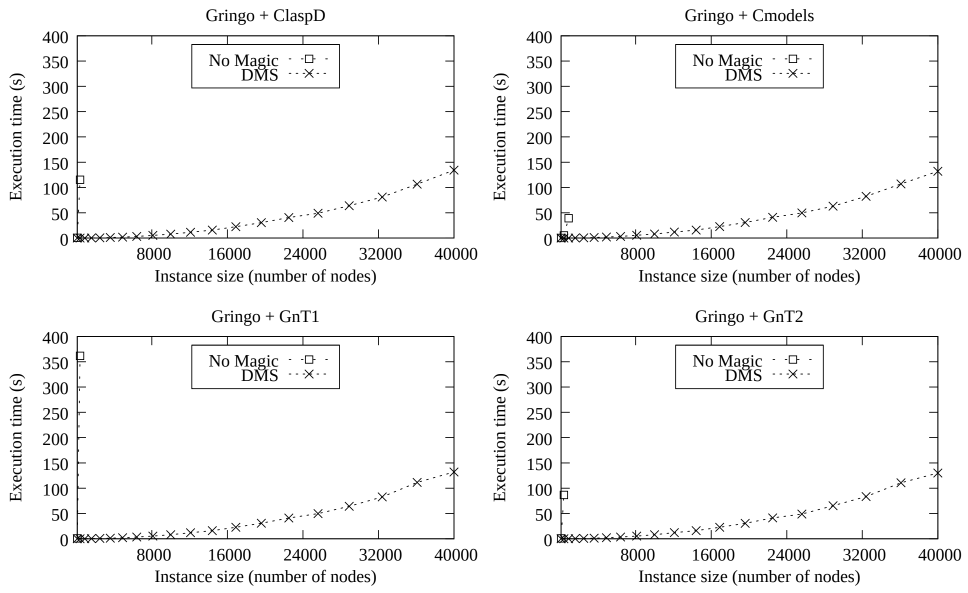


Fig. 12. *Simple Path:* Average execution time on other systems

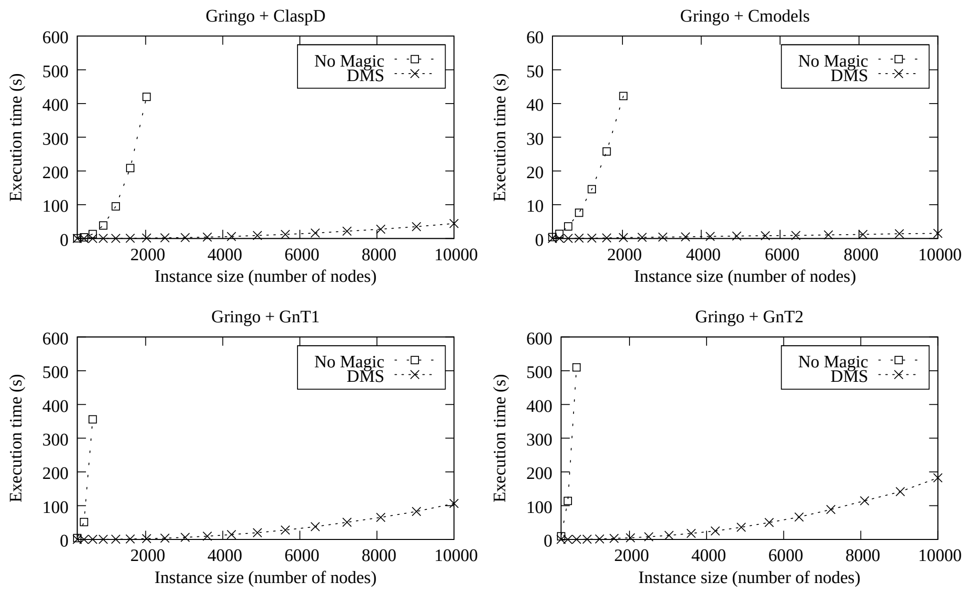
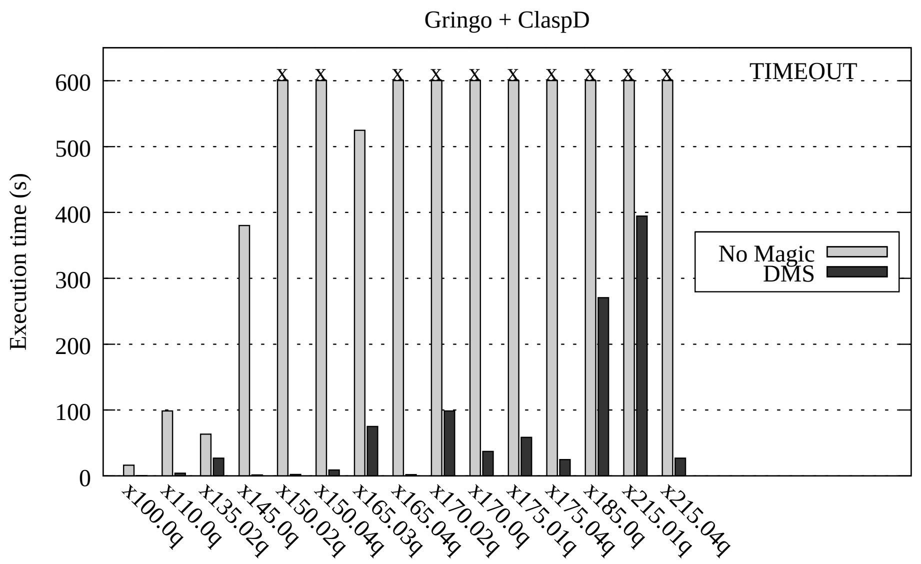


Fig. 13. *Related:* Average execution time on other systems

For *Related* we obtained a similar result, reported in Figure 13 (we used a different scale for the y-axis for Cmodels for readability). Without `DMS` only the smallest instances were solved in the allotted time (up to 2025 nodes for ClaspD and Cmodels, up to 625 nodes for GnT1 and GnT2). With `DMS`, instead, all tested systems solved the biggest instances of the benchmark (up to 10 thousands of nodes). In particular, with `DMS` Cmodels is as performant as DLV in this benchmark.

The effectiveness of `DMS` is also evident in the *Strategic Companies* benchmark (Figures 14–15). In fact, we observed sensible performance gains of all systems on all tested instances. GnT1, which is already faster than the other tested systems in this benchmark, draws particular advantage from `DMS`, solving all



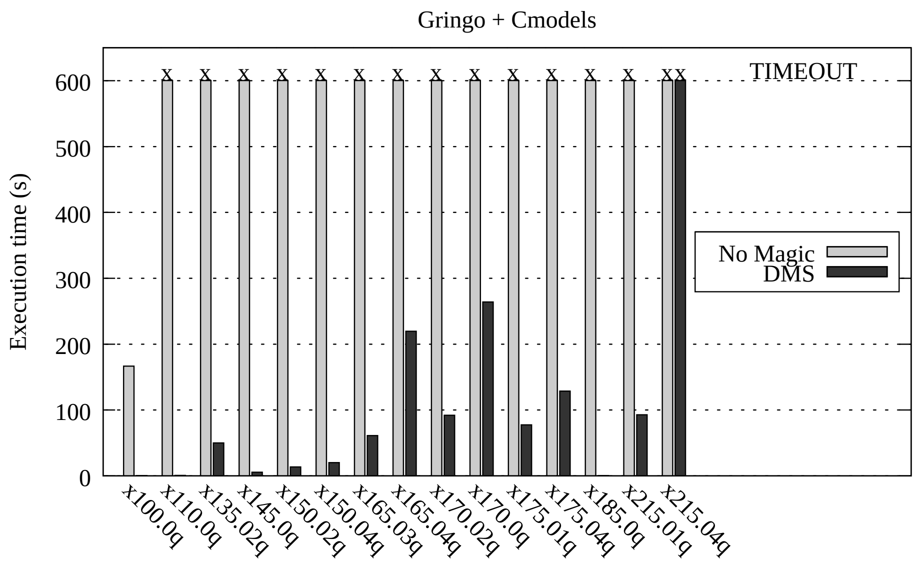
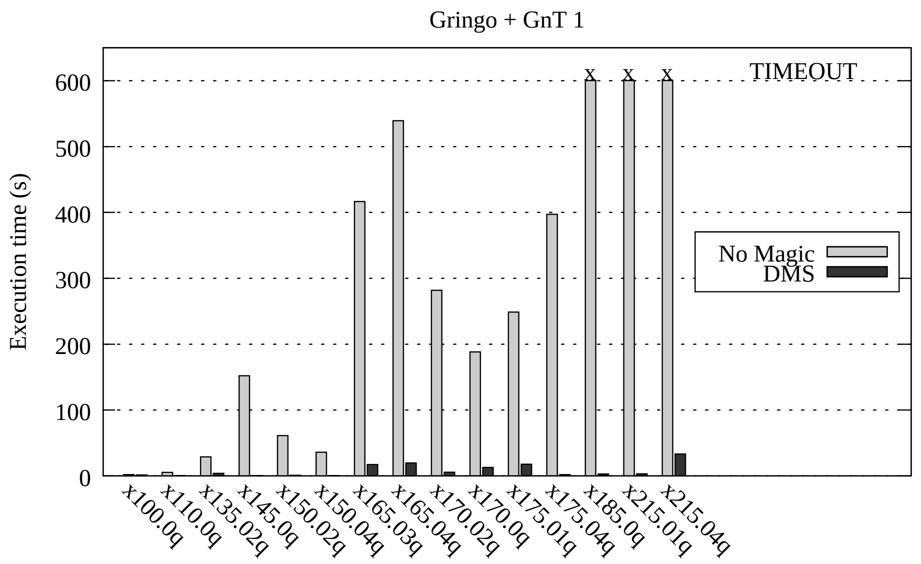


Fig. 14. *Strategic Companies:* Average execution time on other systems (part 1)

instances in few seconds. We give another evidence of the optimization potential provided by `DMS` to these systems by comparing the number of solved instances: Of a total of 60 tests, we counted 37 timeouts on the unoptimized encoding (10 on ClaspD, 14 on Cmodels, 3 on GnT1 and 10 on GnT2), while just one on the encoding obtained by applying `DMS`. We point out that the timeout on the rewritten program was obtained by the Cmodels system, which alone collected 14 timeouts on the unoptimized encoding and is thus the least performant on this benchmark.

Finally, consider the results for *Conformant Plan Checking* reported in Figure 16 (we used a different scale on the y-axis for ClaspD for readability; note also that ClaspD and GnT2 only solved the smallest instances of this



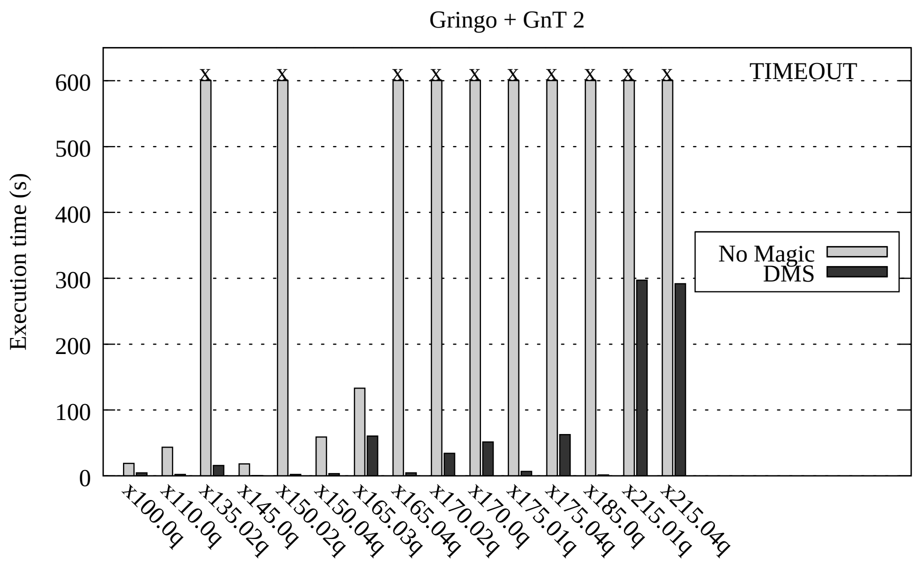


Fig. 15. *Strategic Companies:* Average execution time on other systems (part 2)

benchmark, and we thus used a different scale for their x-axes). The performance of ClaspD is poor in this benchmark, nonetheless we observed a slight improvement in execution time if `DMS` is applied on the encoding reported in Section 5.1. Cmodels performs better than ClaspD in this case and the optimization potential of `DMS` emerges with an exponential improvement in performance. A similar result was observed for GnT1, while GnT2 on this benchmark is the only outlier of the experiment: Its performance deteriorates if the original program is processed by `DMS`. However, in this benchmark GnT2 performs worse that GnT1 also with the original encoding. In fact, while GnT1 solved the biggest instance (more than 65 thousands of states) in 209.74 seconds (12.28 seconds with the `DMS` encoding), the execution of GnT2 did not terminate in the allotted time (600 seconds) on instances containing more than

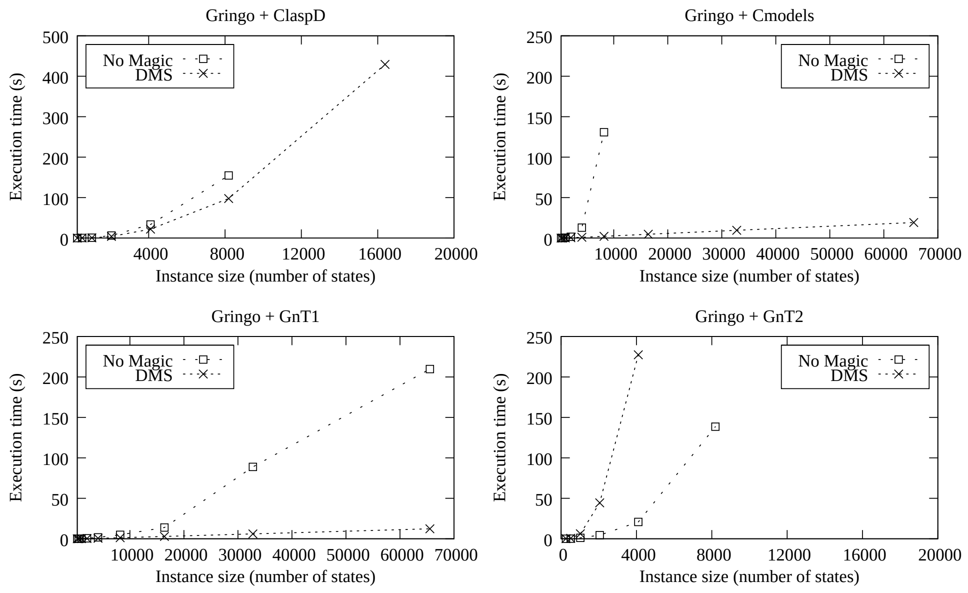


Fig. 16. *Conformant Plan Checking:* Average execution time on other systems

10 thousands of states. We finally note that with DMS GnT1 and Cmodels are faster than DLV in this benchmark. In fact, for the biggest instance in the benchmark, GnT1 and Cmodels required 12.28 and 19.13 seconds, respectively, while DLV terminated in 279.41 seconds. The significant performance gain of GnT1 and Cmodels due to DMS is a further confirmation of the potential of our optimization technique.

6 Application to Data Integration

In this section we give a brief account of a case study that evidences the impact of the Magic Set method when used on programs that realize data integration systems. We first give an overview of data integration systems, show how they can be implemented using Datalog$^{\vee,\neg_s}$, and finally assess the impact of Magic Sets on a data integration system involving real-world data.

6.1 Data Integration Systems in a Nutshell

The main goal of data integration systems is to offer transparent access to heterogeneous sources by providing users with a *global schema*, which users can query without having to know from what sources the data come from. In fact, it is the task of the data integration system to identify and access the data sources which are relevant for finding the answer to a query over the global schema, followed by a combination of the data thus obtained. The data integration system uses a set of *mapping assertions*, which specify the relationship between the data sources and the global schema. Following [41], we formalize a data integration system $\mathcal{I}$ as a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where:

(1) $\mathcal{G}$ is the *global (relational) schema*, that is, a pair $\langle \Psi, \Sigma \rangle$, where Ψ is a finite set of relation symbols, each with an associated positive arity, and Σ is a finite set of *integrity constraints* (ICs) expressed on the symbols in Ψ. ICs are first-order assertions that are intended to be satisfied by database instances.
(2) $\mathcal{S}$ is the *source schema*, constituted by the schemas of the various sources that are part of the data integration system. We assume that $\mathcal{S}$ is a relational schema of the form $\mathcal{S} = \langle \Psi', \emptyset \rangle$, which means that there are no integrity constraints on the sources. This assumption implies that data stored at the sources are locally consistent; this is a common assumption in data integration, because sources are in general external to the integration system, which is not in charge of analyzing or restoring their consistency.
(3) $\mathcal{M}$ is the *mapping* which establishes the relationship between $\mathcal{G}$ and $\mathcal{S}$. In our framework, the mapping follows the GAV approach, that is, each global relation is associated with a *view*—a Datalog$^{\vee,\neg_s}$ query over the sources.

The main semantic issue in data integration systems is that, since integrated sources are originally autonomous, their data, transformed via the mapping assertions, may not satisfy the constraints of the global schema. An approach to remedy to this problem that has lately received a lot of interest in the literature (see, e.g., [3,11,12,14,16–19,25,26]) is based on the notion of *repair* for an inconsistent database as introduced in [4]. Roughly speaking, a repair of a database is a new database that satisfies the constraints in the schema, and minimally differs from the original one. Since an inconsistent database might possess multiple repairs, the standard approach in answering user queries is to return those answers that are true in every possible repair. These are called *consistent answers* in the literature.

6.2 Consistent Query Answering via Datalog$^{\vee,\neg_s}$ Queries

There is an intuitive relation between consistent answers to queries over data integration systems and queries over Datalog$^{\vee,\neg_s}$programs: Indeed, if one could find a translation from data sources, mapping, and the query to a Datalog$^{\vee,\neg_s}$ program, which possesses a stable model for each possible repair, and a query over it, the consistent answers within the data integration system will correspond to cautious consequences of the obtained Datalog$^{\vee,\neg_s}$ setting.

In fact, various authors [5,7,14,16,17,31] considered the idea of encoding the constraints of the global schema $\mathcal{G}$ into various kinds of logic programs, such that the stable models of this program yield the repairs of the database retrieved from the sources. Some of these approaches use logic programs with unstratified negation, [16], whereas disjunctive Datalog programs together with unstratified negation have been considered in [13,51].

It has already been realized earlier that Magic Sets are a crucial optimization technique in this context, and indeed the availability of the transformational approach using stable logic programming as its core language was a main motivation for the research presented in this article, since in this way a Magic Set method for stable logic programs immediately yields an optimization technique for data integration systems. Indeed, the benefits of Magic Sets in the context of optimizing logic programs with unstratified negation (but without disjunction) have been discussed in [24]. The Magic Set technique defined in [24] is quite different from the one defined in this article, as it does not consider disjunctive rules, and works only for programs, which are consistent, that is, have at least one stable model. In [51] our preliminary work reported in [20], which eventually led to the present article, has been expanded in an ad-hoc way to particular kinds of Datalog programs with disjunction and unstratified negation. It is ad-hoc in the sense that it is tailored to programs which are created by the transformation described in [51]. The experimental results reported in [51] show huge computational advantages when using Magic Sets.

We now report an alternative transformation which produces Datalog$^{\vee,\neg_s}$ programs (therefore different to [51], there are no unstratified occurrences of negation). This rewriting has been devised and used within the INFOMIX system on data integration [42].

Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and let $\mathcal{D}$ be a database for $\mathcal{G}$, which is represented as a set of facts over the relational predicates in $\mathcal{G}$. We assume that constraints over the global schema are *key* and *exclusion dependencies*. In particular, we recall that a set of attributes $\bar{x}$ is a key for the relation r if:

$$(r(\bar{x}, \bar{y}) \wedge r(\bar{x}, \bar{z})) \rightarrow \bar{y} = \bar{z}, \qquad \forall \{r(\bar{x}, \bar{y}), r(\bar{x}, \bar{z})\} \subseteq \mathcal{D}$$

and that an exclusion dependency holds between a set of attributes $\bar{x}$ of a relation r and a set of attributes $\bar{w}$ of a relation s if

$$(r(\bar{x}, \bar{y}) \wedge s(\bar{w}, \bar{z})) \rightarrow \bar{y} \neq \bar{z}, \qquad \forall \{r(\bar{x}, \bar{y}), s(\bar{w}, \bar{z})\} \subseteq \mathcal{D}$$

Then, the disjunctive rewriting of a query q with respect to $\mathcal{I}$ is the Datalog$^{\vee,\neg_s}$ program $\Pi(\mathcal{I}) = \Pi_{KD} \cup \Pi_{ED} \cup \Pi_{\mathcal{M}} \cup \Pi_{coll}$ where:

- For each relation r in $\mathcal{G}$ and for each key defined over its set of attributes $\bar{x}$, Π_{KD} contains the rules:

$$r_{out}(\bar{x},\bar{y}) \ \vee \ r_{out}(\bar{x},\bar{z}) \ \text{:-} \ r_{\mathcal{D}}(\bar{x},\bar{y}) \ , \ r_{\mathcal{D}}(\bar{x},\bar{z}), Y_1 \neq Z_1.$$
$$\vdots$$
$$r_{out}(\bar{x},\bar{y}) \ \vee \ r_{out}(\bar{x},\bar{z}) \ \text{:-} \ r_{\mathcal{D}}(\bar{x},\bar{y}) \ , \ r_{\mathcal{D}}(\bar{x},\bar{z}), Y_m \neq Z_m.$$

where $\bar{y} = Y_1, \ldots, Y_m$, and $\bar{z} = Z_1, \ldots, Z_m$.

- For each exclusion dependency between a set of attributes $\bar{x}$ of a relation r and a set of attributes $\bar{w}$ of a relation s, Π_{ED} contains the following rule:

$$r_{out}(\bar{x},\bar{y}) \ \vee \ s_{out}(\bar{w},\bar{z}) \ \text{:-} \ r_{\mathcal{D}}(\bar{x},\bar{y}) \ , \ s_{\mathcal{D}}(\bar{w},\bar{z}), \ X_1 = W_1, \ \ldots, \ X_m = W_m.$$

where $\bar{x} = X_1, \ldots, X_m$, and $\bar{w} = W_1, \ldots, W_m$. In the implementation the following equivalent rule is used:

$$r_{out}(\bar{x},\bar{y}) \ \vee \ s_{out}(\bar{x},\bar{z}) \ \text{:-} \ r_{\mathcal{D}}(\bar{x},\bar{y}), \ s_{\mathcal{D}}(\bar{x},\bar{z}).$$

- For each relation r in $\mathcal{G}$, Π_{coll} contains the rule:

$$r(\bar{w}) \ \text{:-} \ r_{\mathcal{D}}(\bar{w}) \ , \ not \ r_{out}(\bar{w}).$$

- For each Datalog rule r in $\mathcal{M}$ such that:

$$k(\bar{t}) \text{ :- } q_1(\bar{s}_1), \ldots, q_m(\bar{s}_m).$$

where k is a relation in $\mathcal{G}$ and q_i (for $1 \leq i \leq m$) is a relation in $\mathcal{S}$, $\Pi_{\mathcal{M}}$ contains the rule:

$$k_{\mathcal{D}}(\bar{t}) \text{ :- } q_1(\bar{s}_1), \ldots, q_m(\bar{s}_m).$$

It can be shown that for each user query $\mathcal{Q}$ (over $\mathcal{G}$) and for each source database $\mathcal{F}$ (over $\mathcal{S}$), consistent query answers to $\mathcal{Q}$ precisely coincide with the set $Ans_c(\mathcal{Q}, \Pi(\mathcal{I}) \cup \mathcal{F})$. Actually, within the INFOMIX project also *inclusion dependencies* have been considered according to the rewriting discussed in [16], whose details we omit for clarity. Since the rewriting for inclusion dependencies also modifies queries, in the INFOMIX project queries have been limited to conjunctive queries. It is however important to notice that the program $\Pi(\mathcal{I})$ contains only stratified negation and is therefore a Datalog$^{\vee,\neg_s}$ program, making the Magic Set method defined in this article applicable.

6.3 Experimental Results

The effectiveness of the Magic Set method in this crucial application context has then been assessed via a number of experiments carried out 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

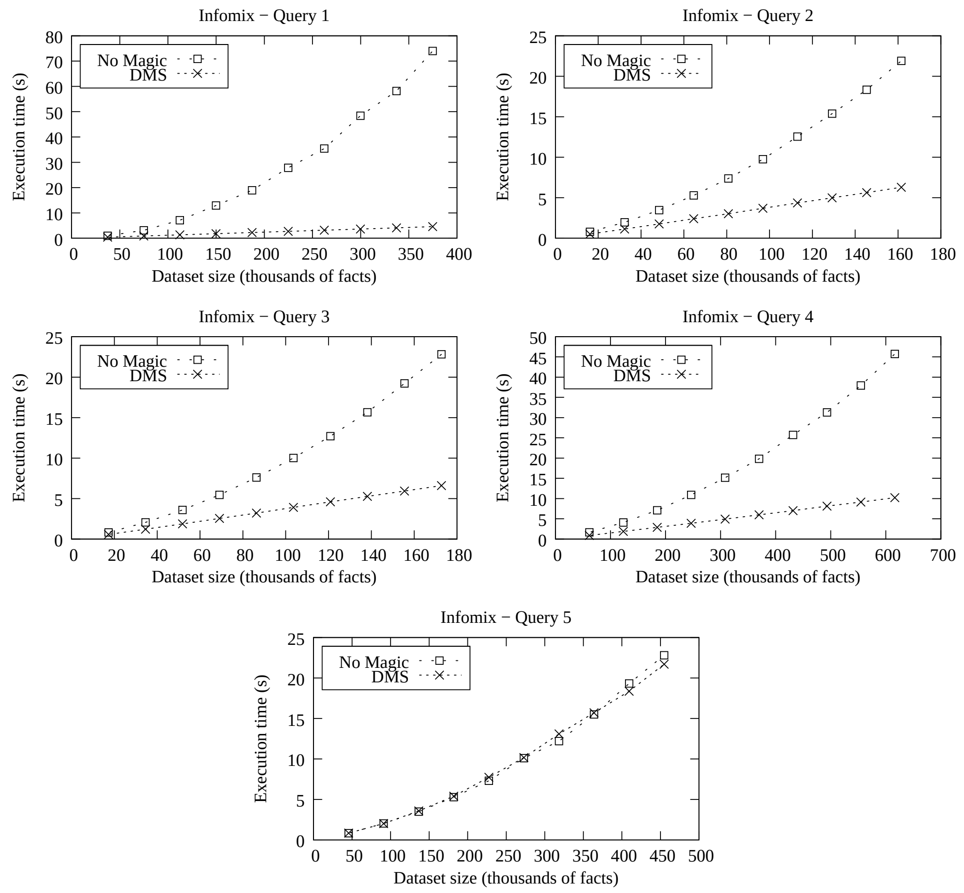


Fig. 17. Average execution time of query evaluation in the INFOMIX Demo Scenario

global relations with 29 constraints, while the data sources include 29 relations of 3 legacy databases and 12 wrappers generating relational data from web pages. This amounts to more than 24MB of data regarding students, professors and exams in several faculties of the university. For a detailed description of the INFOMIX project see `https://www.mat.unical.it/infomix/`.

On this schema, we have tested five typical queries with different characteristics, which model different use cases. For the sake of completeness, the full encodings of the tested queries are reported in the Appendix. In particular, we measured the average execution time of DLV computing $Ans_c(\mathcal{Q}, \Pi(\mathcal{I}) \cup \mathcal{F})$ and $Ans_c(\mathcal{Q}, \mathtt{DMS}(\mathcal{Q}, \Pi(\mathcal{I})) \cup \mathcal{F})$ on datasets of increasing size. The experiments were performed by running the INFOMIX prototype system on a 3GHz Intel$^{®}$ Xeon$^{®}$ processor system with 4GB RAM under the Debian 4.0 operating system with a GNU/Linux 2.6.23 kernel. The DLV prototype used as the computational core of the INFOMIX system had been compiled using GCC 4.3.3. For each instance, we allowed a maximum running time of 10 minutes and a maximum memory usage of 3GB.

The results, reported in Figure 17, confirm that on these typical queries the performance is considerably improved by Magic Sets. On Queries 1 to 4 in Figure 17 the response time scales much better with Magic Sets than without, appearing essentially linear on the tested instance sizes, while without Magic Sets the behavior has a decidedly non-linear appearance. We also observe that there is basically no improvement on Query 5. We have analyzed this query and for this use case all data seems to be relevant to the query, which means that Magic Sets cannot have any positive effect. It is however important to observe that the Magic Set rewriting does not incur any significant overhead.

7 Related Work

In this section we first discuss the main body of work which is related to `DMS`, the technique developed in this paper for query answering optimization. In particular, we discuss Magic Set techniques for Datalog languages. The discussion is structured in paragraphs grouping techniques which cover the same language. After that, we discuss some applications for which `DMS` have already been exploited. All these applications refer to the preliminary work published in [20].

Magic Sets for Datalog. In order to optimize query evaluation in bottom-up systems, like deductive database systems, several works have proposed the simulation of top-down strategies by means of suitable transformations introducing new predicates and rewriting clauses. Among them, Magic Sets for Datalog queries are one of the best known logical optimization techniques for database systems. The method, first developed in [6], has been analyzed and refined by many authors; see, for instance, [9,55,62,63]. These works form the foundations of `DMS`.

Magic Sets for Datalog$^{\neg_s}$. Many authors have addressed the issue of extending the Magic Set technique in order to deal with Datalog queries involving stratified negation. The main problem related to the extension of the technique to Datalog$^{\neg_s}$ programs is how to assign a semantics to the rewritten programs. Indeed, while Datalog$^{\neg_s}$ programs have a natural and accepted semantics, namely the perfect model semantics [2,64], the application of Magic Sets can introduce unstratified negation in the rewritten programs. A solution has been presented in [10,38,39,59]. In particular, in [38,59] rewritten programs have been evaluated according to the well-founded semantics, a three-valued semantics for Datalog$^{\neg}$ programs which is two-valued for stratified programs, while in [10,39] ad-hoc semantics have been defined. All of these methods exploit a property of Datalog$^{\neg_s}$ which is not present in disjunctive Datalog, uniqueness of the intended model. This property in turn implies that query answering just consists in establishing the truth value of some atoms in one intended model. Using our terminology, brave and cautious reasoning coincide for these programs. Therefore, all these methods are quite different from `DMS`, the technique developed in this paper.

Magic Sets for Datalog$^\neg$. Extending the Magic Set technique to Datalog$^\neg$ programs must face two major difficulties. First, for a Datalog$^\neg$ program uniqueness of the intended model is no more guaranteed, thus query answering in this setting involves a set of stable models in general. The second difficulty is that parts of a Datalog$^\neg$ program may act as constraints, thus impeding a relevant interpretation to be a stable model. In [24] a Magic Set method for Datalog$^\neg$ programs has been defined and proved to be correct for coherent programs, i.e., programs admitting at least one stable model. This method takes special precautions for relevant parts of the program that act as constraints, called *dangerous rules* in [24]. We observe that dangerous rules cannot occur in Datalog$^{\vee,\neg_s}$ programs, which allows for the simpler `DMS` algorithm to work correctly for this class of programs.

Magic Sets for Datalog$^\vee$. The first extension of the Magic Set technique to disjunctive Datalog is due to [32,33], where the `SMS` method has been presented and proved to be correct for Datalog$^\vee$ programs. We point out that the main drawback of this method is the introduction of *collecting* predicates. Indeed, magic and collecting predicates of `SMS` have deterministic definitions. As a consequence, their extension can be completely computed during program instantiation, which means that no further optimization is provided for the subsequent stable model search. Moreover, while the correctness of `DMS` has been formally established for Datalog$^{\vee,\neg_s}$ programs in general, the applicability of `SMS` to Datalog$^{\vee,\neg_s}$ programs has only been outlined in [32,33].

Applications. Magic Sets have been applied in many contexts. In particular, [13,36,51,53] have profitably exploited the optimization provided by `DMS`. In particular, in [13,51] a data integration system has been presented. The system is based on disjunctive Datalog and exploits `DMS` for fast query answering. In [36,53], instead, an algorithm for answering queries over description logic knowledge bases has been presented. More specifically, the algorithm reduces a $\mathcal{SHIQ}$ knowledge base to a disjunctive Datalog program, so that `DMS` can be exploited for query answering optimization.

8 Conclusion

The Magic Set method is one of the best-known techniques for the optimization of positive recursive Datalog programs due to its efficiency and its generality. Just a few other focused methods such as the supplementary Magic Set and other special techniques for linear and chain queries have gained similar visibility (see, e.g., [34,56,63]). After seminal papers [6,9], the viability of the approach was demonstrated e.g., in [35,55]. Later on, extensions and refinements were proposed, addressing e.g., query constraints in [62], the well-founded semantics in [38], or integration into cost-based query optimization in [60]. The research on variations of the Magic Set method is still going on. For instance, in [24] an extension of the Magic Set method was discussed for the class of unstratified logic programs (without disjunction). In [10] a technique for the class of *soft-stratifiable* programs was given. Finally, in [33] the first variant of the technique for disjunctive programs (`SMS`) was described.

In this paper, we have elaborated on the issues addressed in [32,33]. Our approach is similar to `SMS`, but differs in several respects:

- `DMS` is a dynamic optimization of query answering, in the sense that in addition to the optimization of the grounding process (which is the only optimization performed by `SMS`), `DMS` can drive the model generation phase by dynamically disabling parts of the program that become irrelevant in the considered partial interpretations.
- `DMS` has a strong relationship with unfounded sets, allowing for a clean application to disjunctive Datalog programs also in presence of stratified negation.
- `DMS` can be further improved by performing a subsequent subsumption check.
- `DMS` is integrated into the DLV system [43], profitably exploiting the DLV internal data-structures and the ability of controlling the grounding module.

We have conducted experiments on several benchmarks, many of which taken from the literature. The results of our experimentation evidence that our implementation outperforms `SMS` in general, often by an exponential factor. This is mainly due to the optimization of the model generation phase, which is specific to our Magic Set technique. In addition, we have conducted further experiments on a real application scenario, which show that Magic Sets can play a crucial role in optimizing consistent query answering over inconsistent databases. Importantly, other authors have already recognized the benefits of our optimization strategies with respect to this very important application domain [51], thereby confirming the validity and the robustness of the work discussed in this paper.

We conclude by observing that it has been noted in the literature (e.g., in [38]) that in the non-disjunctive case *memoing* techniques lead to similar computations as evaluations after Magic Set transformations. Also in the disjunctive case such techniques have been proposed (e.g., Hyper Tableaux [8]), for which similar relations might hold. While [38] has already evidenced that an advantage of Magic Sets over such methods is that they may be more easily combined with other optimization techniques, we believe that achieving a deeper comprehension of the relationships among these techniques constitutes an interesting avenue for further research.

Another issue that we leave for future work is to study the impact of changing some parameters of the `DMS` method, in particular the impact of different SIPSes.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a Theory of Declarative Knowledge. In Minker [52], pages 89–148.

[3] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Scalar aggregation in fd-inconsistent databases. In *International Conference on Database Theory (ICDT-2001)*, pages 39–53. Springer Verlag, 2001.

[4] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 68–79, 1999.

[5] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Specifying and querying database repairs using logic programs with exceptions. In *Proc. of the 4th Int. Conf. on Flexible Query Answering Systems (FQAS 2000)*, pages 27–41. Springer, 2000.

[6] François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic Sets and Other Strange Ways to Implement Logic Programs. In *Proc. Int. Symposium on Principles of Database Systems*, pages 1–16, 1986.

[7] Pablo Barceló and Leopoldo Bertossi. Repairing databases with annotated predicate logic. In *Proc. the 10th Int. Workshop on Non-Monotonic Reasoning (NMR 2002)*, pages 160–170, 2002.

[8] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *Proceedings of the European Workshop on Logics in Artificial Intelligence (JELIA'96)*, number 1126 in LNCS, pages 1–17. Springer, 1996.

[9] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. *Journal of Logic Programming*, 10(1–4):255–259, 1991.

[10] Andreas Behrend. Soft stratification for magic set based query evaluation in deductive databases. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 102–110, New York, NY, USA, 2003. ACM.

[11] Leo Bertossi and Jan Chomicki. Query answering in inconsistent databases. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*, chapter 2, pages 43–83. Springer-Verlag, 2003.

[12] Leopoldo Bertossi, Jan Chomicki, Alvaro Cortes, and Claudio Gutierrez. Consistent answers from integrated data sources. In *Proc. of the 6th Int. Conf. on Flexible Query Answering Systems (FQAS 2002)*, pages 71–85, 2002.

[13] Leopoldo E. Bertossi and Loreto Bravo. Consistent query answers in virtual data integration systems. In *Inconsistency Tolerance*, volume 3300 of *LNCS*, pages 42–83. Springer, 2005.

[14] Loreto Bravo and Leopoldo Bertossi. Logic programming for consistently querying data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 10–15, 2003.

[15] Marco Cadoli, Thomas Eiter, and Georg Gottlob. Default Logic as a Query Language. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):448–463, May/June 1997.

[16] Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 16–21, 2003.

[17] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, 2005.

[18] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Computing consistent query answers using conflict hypergraphs. In *Proc. 13th ACM Conference on Information and Knowledge Management (CIKM-2004)*, pages 417–426. ACM Press, 2004.

[19] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Hippo: A System for Computing Consistent Answers to a Class of SQL Queries. In Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari, editors, *9th International Conference on Extending Database Technology (EDBT-2004)*, volume 2992 of *Lecture Notes in Computer Science*, pages 841–844. Springer, 2004.

[20] Chiara Cumbo, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Enhancing the magic-set method for disjunctive datalog programs. In *Proceedings of the the 20th International Conference on Logic Programming – ICLP'04*, volume 3132 of *Lecture Notes in Computer Science*, pages 371–385, 2004.

[21] Christian Drescher, Martin Gebser, Torsten Grote, Benjamin Kaufmann, Arne König, Max Ostrowski, and Torsten Schaub. Conflict-Driven Disjunctive Answer Set Solving. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 422–432, Sydney, Australia, 2008. AAAI Press.

[22] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, September 1997.

[23] Wolfgang Faber. *Enhancing Efficiency and Expressiveness in Answer Set Programming Systems*. PhD thesis, Institut für Informationssysteme, Technische Universität Wien, 2002.

[24] Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic Sets and their Application to Data Integration. *Journal of Computer and System Sciences*, 73(4):584–609, 2007.

[25] Ariel Fuxman, Elham Fazli, and Renée J. Miller. Conquer: Efficient management of inconsistent databases. In *SIGMOD Conference*, 2005.

[26] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. In Thomas Eiter and Leonid Libkin, editors, *Proceedings of the 10th International Conference on Database Theory (ICDT 2005)*, number 3363 in LNCS, pages 337–351. Springer, 2005.

[27] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[28] Martin Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. In Chitta Baral, Gerhard Brewka, and John Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning — 9th International Conference, LPNMR'07*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271, Tempe, Arizona, May 2007. Springer Verlag.

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

[30] R. Goldman and M. Boddy. Expressive Planning and Explicit Knowledge. In *Proceedings AIPS-96*, pages 110–117. AAAI Press, 1996.

[31] Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Proc. of the 17th Int. Conf. on Logic Programming (ICLP'01)*, volume 2237 of *Lecture Notes in Artificial Intelligence*, pages 348–364. Springer, 2001.

[32] Sergio Greco. Optimization of Disjunction Queries. In Danny De Schreye, editor, *Proceedings of the 16th International Conference on Logic Programming (ICLP'99)*, pages 441–455, Las Cruces, New Mexico, USA, November 1999. The MIT Press.

[33] Sergio Greco. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):368–385, March/April 2003.

[34] Sergio Greco, Domenico Saccà, and Carlo Zaniolo. The PushDown Method to Optimize Chain Logic Programs (Extended Abstract). In *Proc. Int. Colloquim on Automata, Languages and Programming*, pages 523–534, 1995.

[35] Ashish Gupta and Inderpal Singh Mumick. Magic-sets Transformation in Nonrecursive Systems. In *Proceedings of the Thirteenth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-92)*, pages 354–367, 1992.

[36] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.

[37] Tomi Janhunen, Ilkka Niemelä, Dietmar Seipel, Patrik Simons, and Jia-Huai You. Unfolding Partiality and Disjunctions in Stable Model Semantics. *ACM Transactions on Computational Logic*, 7(1):1–37, January 2006.

[38] David B. Kemp, Divesh Srivastava, and Peter J. Stuckey. Bottom-up evaluation and query optimization of well-founded models. *Theoretical Computer Science*, 146:145–184, July 1995.

[39] Jean-Marc Kerisit and Jean-Marc Pugin. Efficient query answering on stratified databases. In *FGCS*, pages 719–726, 1988.

[40] Joohyung Lee and Vladimir Lifschitz. Loop Formulas for Disjunctive Logic Programs. In *Proceedings of the Nineteenth International Conference on Logic Programming (ICLP-03)*. Springer Verlag, December 2003.

[41] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–246, 2002.

[42] Nicola Leone, Georg Gottlob, Riccardo Rosati, Thomas Eiter, Wolfgang Faber, Michael Fink, Gianluigi Greco, Giovambattista Ianni, Edyta Kałka, Domenico Lembo, Maurizio Lenzerini, Vincenzino Lio, Bartosz Nowicki, Marco Ruzzi, Witold Staniszkis, and Giorgio Terracina. The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In *Proceedings of the 24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, pages 915–917, Baltimore, Maryland, USA, June 2005. ACM Press.

[43] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, July 2006.

[44] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. *Information and Computation*, 135(2):69–112, June 1997.

[45] Senlin Liang, Paul Fodor, Hui Wan, and Michael Kifer. OpenRuleBench: An analysis of the performance of rule engines. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web (WWW 2009)*, pages 601–610. ACM, 2009.

[46] Yuliya Lierler. Disjunctive Answer Set Programming via Satisfiability. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *Logic Programming and Nonmonotonic Reasoning — 8th International Conference, LPNMR'05, Diamante, Italy, September 2005, Proceedings*, volume 3662 of *Lecture Notes in Computer Science*, pages 447–451. Springer Verlag, September 2005.

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

[48] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming.* The MIT Press, Cambridge, Massachusetts, 1992.

[49] Marco Manna, Massimo Ruffolo, Ermelinda Oro, Mario Alviano, and Nicola Leone. The HiLeX system for semantic information extraction. *Transactions on Large-Scale Data- and Knowledge-Centered Systems.* Springer Berlin/Heidelberg, Lecture Notes in Computer Science 7100:91–125, 2012.

[50] Marco Manna, Francesco Scarcello, and Nicola Leone. On the complexity of regular-grammars with integer attributes. *Journal of Computer and System Sciences (JCSS)*, 77(2):393–421, 2011.

[51] Mónica Caniupán Marileo and Leopoldo E. Bertossi. The consistency extractor system: Querying inconsistent databases using answer set programs. In *SUM 2007*, pages 74–88, 2007.

[52] Jack Minker, editor. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, Inc., Washington DC, 1988.

[53] Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases.* PhD thesis, Fakultät für Wirtschaftswissenschaften, Universität Fridericiana zu Karlsruhe, 2006.

[54] Boris Motik and Ulrike Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In Miki Hermann and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference (LPAR 2006)*, volume 4246 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2006.

[55] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic is relevant. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 247–258, 1990.

[56] Raghu Ramakrishnan, Yehoshua Sagiv, Jeffrey D. Ullman, and Moshe Y. Vardi. Logical Query Optimization by Proof-Tree Transformation. *Journal of Computer and System Sciences*, 47(1):222–248, 1993.

[57] Francesco Ricca, Mario Alviano, Antonella Dimasi, Giovanni Grasso, Salvatore Maria Ielpa, Salvatore Iiritano, Marco Manna, and Nicola Leone. A logic-based system for e-tourism. *Fundamenta Informaticae.* IOS Press, 105(1–2):35–55, 2010.

[58] Francesco Ricca, Giovanni Grasso, Mario Alviano, Marco Manna, Vincenzino Lio, Salvatore Iiritano, and Nicola Leone. Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practice of Logic Programming*, 2012. To appear, doi:10.1017/S147106841100007X.

[59] K. A. Ross. Modular Stratification and Magic Sets for Datalog Programs with Negation. *Journal of the ACM*, 41(6):1216–1266, 1994.

[60] Praveen Seshadri, Joseph M. Hellerstein, Hamid Pirahesh, T. Y. Cliff Leung, Raghu Ramakrishnan, Divesh Srivastava, Peter J. Stuckey, and S. Sudarshan. Cost-based optimization for magic: Algebra and implementation. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 435–446. ACM Press, June 1996.

[61] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138:181–234, June 2002.

[62] Peter J. Stuckey and S. Sudarshan. Compiling query constraints. In *Proceedings of the Thirteenth Symposium on Principles of Database Systems (PODS'94)*, pages 56–67. ACM Press, May 1994.

[63] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II.* Computer Science Press, 1989.

[64] A. van Gelder. Negation as Failure Using Tight Derivations for General Logic Programs. In Minker [52], pages 1149–1176.

A Queries on the INFOMIX Demo Scenario

INFOMIX is a project that was funded by the European Commission in its Information Society Technologies track of the Sixth Framework Programme for providing an advanced system for information integration. A detailed description of the project, including references in the literature, can be found at `https://www.mat.unical.it/infomix/`. Five typical queries of the INFOMIX demo scenario have been considered for assessing Dynamic Magic Sets. The full encodings of the tested queries are reported in Figures A.1–A.2. Note that the encodings include the transformation described in Section 6, and that underlined predicates denote source relations.

course$_{\mathcal{D}}$(X$_1$,X$_2$) :− $\underline{\texttt{esame}}$(_,X$_1$,X$_2$,_).

course$_{\mathcal{D}}$(X$_1$,X$_2$) :− $\underline{\texttt{esame\_diploma}}$(X$_1$,X$_2$).

exam_record$_{\mathcal{D}}$(X$_1$,X$_2$,Z,W,X$_4$,X$_5$,Y) :− $\underline{\texttt{affidamenti\_ing\_informatica}}$(X$_2$,X$_3$,Y),

  $\underline{\texttt{dati\_esami}}$(X$_1$,_,X$_2$,X$_5$,X$_4$,_,Y), $\underline{\texttt{dati\_professori}}$(X$_3$,Z,W).

exam_record$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,Y$_5$,Y$_6$,Y$_7$) $\vee$ exam_record$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,Z$_5$,Z$_6$,Z$_7$) :−

  exam_record$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,Y$_5$,Y$_6$,Y$_7$), exam_record$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,Z$_5$,Z$_6$,Z$_7$), Y$_5$$\neq$Z$_5$.

exam_record$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,Y$_5$,Y$_6$,Y$_7$) $\vee$ exam_record$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,Z$_5$,Z$_6$,Z$_7$) :−

  exam_record$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,Y$_5$,Y$_6$,Y$_7$), exam_record$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,Z$_5$,Z$_6$,Z$_7$), Y$_6$$\neq$Z$_6$.

exam_record$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,Y$_5$,Y$_6$,Y$_7$) $\vee$ exam_record$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,Z$_5$,Z$_6$,Z$_7$) :−

  exam_record$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,Y$_5$,Y$_6$,Y$_7$), exam_record$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,Z$_5$,Z$_6$,Z$_7$), Y$_7$$\neq$Z$_7$.

course(X$_1$,X$_2$) :− course$_{\mathcal{D}}$(X$_1$,X$_2$), not course$_{\text{out}}$(X$_1$,X$_2$).

exam_record(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$) :− exam_record$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$),

  not exam_record$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$).

query$_1$(CD) :− course(C,CD), exam_record("09089903",C,_,_,_,_,_).

query$_1$(CD)?

---

student$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$) :− $\underline{\texttt{diploma\_maturita}}$(Y,X$_7$),

  $\underline{\texttt{studente}}$(X$_1$,X$_3$,X$_2$,_,_,_,_,_,_,_,_,_,X$_6$,X$_5$,_,_,X$_4$,_,_,_,_,Y,_).

student(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$) :− student$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$),

  not student$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$).

query$_2$(SFN,SLN,COR,ADD,TEL,HSS) :− student("09089903",SFN,SLN,COR,ADD,TEL,HSS).

query$_2$(SFN,SLN,COR,ADD,TEL,HSS)?

---

student$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$) :− $\underline{\texttt{diploma\_maturita}}$(Y,X$_7$),

  $\underline{\texttt{studente}}$(X$_1$,X$_3$,X$_2$,_,_,_,_,_,_,_,_,_,X$_6$,X$_5$,_,_,X$_4$,_,_,_,_,Y,_).

student_course_plan$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$) :− $\underline{\texttt{orientamento}}$(Y$_1$,X$_3$),

  $\underline{\texttt{piano\_studi}}$(X$_1$,X$_2$,Y$_1$,X$_4$,Y$_2$,_,_,_,_,_), $\underline{\texttt{stato}}$(Y$_2$,X$_5$).

student(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$) :− student$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$),

  not student$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$).

student_course_plan(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$) :− student_course_plan$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$),

  not student_course_plan$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$).

query$_3$(SID,SLN,R) :− student(SID,"ZNEPB",SLN,_,_,_,_),

  student_course_plan(_,SID,_,R,"APPROVATO SENZA MODIFICHE").

query$_3$(SID,SLN,R)?

Fig. A.1. INFOMIX Queries 1–3

student$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$) :− $\underline{\texttt{diploma\_maturita}}$(Y,X$_7$),
        $\underline{\texttt{studente}}$(X$_1$,X$_3$,X$_2$,_,_,_,_,_,_,_,_,_,X$_6$,X$_5$,_,_,X$_4$,_,_,_,_,Y,_).
course$_{\mathcal{D}}$(X$_1$,X$_2$) :− $\underline{\texttt{esame}}$(_,X$_1$,X$_2$,_).
course$_{\mathcal{D}}$(X$_1$,X$_2$) :− $\underline{\texttt{esame\_diploma}}$(X$_1$,X$_2$).
student_course_plan$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$) :− $\underline{\texttt{orientamento}}$(Y$_1$,X$_3$),
        $\underline{\texttt{piano\_studi}}$(X$_1$,X$_2$,Y1,X$_4$,Y$_2$,_,_,_,_,_), $\underline{\texttt{stato}}$(Y$_2$,X$_5$).
plan_data$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$) :− $\underline{\texttt{dati\_piano\_studi}}$(X$_1$,X$_2$,_),
        $\underline{\texttt{esame\_ingegneria}}$(X$_2$,Y$_3$,Y$_2$,_), $\underline{\texttt{tipo\_esame}}$(Y$_2$,X$_3$).
student(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$) :− student$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$),
        not student$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$,X$_6$,X$_7$).
student_course_plan(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$) :− student_course_plan$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$)
        not student_course_plan$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$).
plan_data(X$_1$,X$_2$,X$_3$) :− plan_data$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$), not plan_data$_{\text{out}}$(X$_1$,X$_2$,X$_3$).
course(X$_1$,X$_2$) :− course$_{\mathcal{D}}$(X$_1$,X$_2$), not course$_{\text{out}}$(X$_1$,X$_2$).
query$_4$(F,S) :− course(CID,“RETILOGICHE”), plan_data(SCID,CID,_),
        student(SID,F,S,“ROMA”,_,_,_), student_course_plan(SCID,SID,_,_,_).
query$_4$(F,S)?

---

course$_{\mathcal{D}}$(X$_1$,X$_2$) :− $\underline{\texttt{esame}}$(_,X$_1$,X$_2$,_).
course$_{\mathcal{D}}$(X$_1$,X$_2$) :− $\underline{\texttt{esame\_diploma}}$(X$_1$,X$_2$).
student_course_plan$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$) :− $\underline{\texttt{orientamento}}$(Y$_1$,X$_3$),
        $\underline{\texttt{piano\_studi}}$(X$_1$,X$_2$,Y$_1$,X$_4$,Y$_2$,_,_,_,_,_), $\underline{\texttt{stato}}$(Y$_2$,X$_5$).
plan_data$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$) :− $\underline{\texttt{dati\_piano\_studi}}$(X$_1$,X$_2$,_),
        $\underline{\texttt{esame\_ingegneria}}$(X$_2$,Y$_3$,Y$_2$,_), $\underline{\texttt{tipo\_esame}}$(Y$_2$,X$_3$).
student_course_plan(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$) :− student_course_plan$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$),
        not student_course_plan$_{\text{out}}$(X$_1$,X$_2$,X$_3$,X$_4$,X$_5$).
plan_data(X$_1$,X$_2$,X$_3$) :− plan_data$_{\mathcal{D}}$(X$_1$,X$_2$,X$_3$), not plan_data$_{\text{out}}$(X$_1$,X$_2$,X$_3$).
course(X$_1$,X$_2$) :− course$_{\mathcal{D}}$(X$_1$,X$_2$), not course$_{\text{out}}$(X$_1$,X$_2$).
query$_5$(D) :− course(E,D), plan_data(C,E,_), student_course_plan(C,“09089903”,_,_,_).
query$_5$(D)?

Fig. A.2. INFOMIX Queries 4–5