Dynamic Magic Sets for Super-Consistent Answer Set Programs[⋆]

Mario Alviano and Wolfgang Faber

Department of Mathematics, University of Calabria, 87030 Rende (CS), Italy
{alviano,faber}@mat.unical.it

Abstract. For many practical applications of ASP, for instance data integration or planning, query answering is important, and therefore query optimization techniques for ASP are of great interest. Magic Sets are one of these techniques, originally defined for Datalog queries (ASP without disjunction and negation). Dynamic Magic Sets (DMS) are an extension of this technique, which has been proved to be sound and complete for query answering over ASP programs with stratified negation.
A distinguishing feature of DMS is that the optimization can be exploited also during the nondeterministic phase of ASP engines. 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, which may result in exponential performance gains.
In this paper, the correctness of DMS is formally established and proved for brave and cautious reasoning over the class of super-consistent ASP programs (ASP^{sc} programs). ASP^{sc} programs guarantee consistency (i.e., have answer sets) when an arbitrary set of facts is added to them. This result generalizes the applicability of DMS, since the class of ASP^{sc} programs is richer than ASP programs with stratified negation, and in particular includes all odd-cycle-free programs. DMS has been implemented as an extension of DLV, and the effectiveness of DMS for ASP^{sc} programs is empirically confirmed by experimental results with this system.

1 Introduction

Answer Set Programming (ASP) is a powerful formalism for knowledge representation and common sense reasoning [1]. Allowing disjunction in rule heads and nonmonotonic negation in bodies, ASP can express every query belonging to the complexity class Σ_2^P ($\mathrm{NP}^{\mathrm{NP}}$). For this reason, it is not surprising that ASP has found several practical applications, also encouraged by the availability of efficient inference engines, such as DLV [2], GnT [3], Cmodels [4], or ClaspD [5]. As a matter of fact, these systems are continuously enhanced to support novel

[⋆] This research has been partly supported by Regione Calabria and EU under POR Calabria FESR 2007-2013 within the PIA project of DLVSYSTEM s.r.l., and by MIUR under the PRIN project LoDeN.

optimization strategies, enabling them to be effective over increasingly larger application domains. Magic Sets are one of these techniques [6–8].

The goal of the original Magic Set method (defined in the field of Deductive Databases for Datalog programs, i.e., disjunction-free positive ASP programs) is to exploit the presence of constants in a query for restricting the possible search space by considering only a subset of a hypothetic program instantiation, which is sufficient to answer the query in question. Magic sets are extensions of predicates that make this restriction explicit. Extending these ideas to ASP faces a major challenge: While Datalog programs are deterministic, ASP programs are in general nondeterministic.

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 answer set. The second possibility is to allow *dynamic* magic sets, which also allow for non-deterministic definitions of magic sets. This means that the extension of the magic set predicates may differ in various answer sets, and thus can be viewed as being specialized for different answer sets. This also mimics the architecture of current ASP systems, which are divided into a deterministic (grounding) and a non-deterministic (model search) phase.

In [9] the first Dynamic Magic Set (`DMS`) method has been proposed and proved correct for ASP with stratified negation. In this work, we show that this technique can be easily extended and shown to be correct for a broader class of programs, which we call super-consistent ASP programs (ASP^{sc} programs), which includes all stratified and odd-cycle-free programs. In more detail, the contributions are:

- We formally establish the correctness of `DMS` for ASP^{sc} programs. 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 have implemented a `DMS` optimization module inside the DLV system [2]. 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 implementation is available at `http://www.dlvsystem.com/magic/`.
- We have conducted experiments on a synthetic domain that highlight the potential of `DMS`. We have compared the performance of the DLV system without magic set optimization and with `DMS`. The results show that `DMS` can yield drastically better performance than the non optimized evaluation.

Organization. In Section 2, syntax and semantics of ASP are introduced. In this section, we also define ASP^{sc} programs. In Section 3, we show how to apply `DMS` to ASP^{sc} programs and formally prove its correctness. In Section 4, we discuss the implementation and integration of the Magic Set method within the DLV system. Experimental results are reported in Section 5. Finally, in Section 6 we draw our conclusions.

2 Preliminaries

In this section, we recall the basics of ASP and introduce the class of super-consistent ASP programs (ASP$^{\text{sc}}$ programs).

2.1 ASP Syntax and Semantics

A *term* is either a *variable* or a *constant*. If $\mathtt{p}$ is a *predicate* of arity $k \geq 0$, and $\mathtt{t_1}, \ldots, \mathtt{t_k}$ are terms, then $\mathtt{p(t_1, \ldots, t_k)}$ is an *atom*[1]. A *literal* is either an atom $\mathtt{p(\bar{t})}$ (a positive literal), or an atom preceded by the *negation as failure* symbol $\mathtt{not\ p(\bar{t})}$ (a negative literal). A *rule* r is of the form

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

where $\mathtt{p_1(\bar{t}_1)},\ \ldots,\ \mathtt{p_n(\bar{t}_n)},\ \mathtt{q_1(\bar{s}_1)},\ \ldots,\ \mathtt{q_m(\bar{s}_m)}$ are atoms and $n \geq 1$, $m \geq j \geq 0$. The disjunction $\mathtt{p_1(\bar{t}_1)\ v\ \cdots\ v\ p_n(\bar{t}_n)}$ is the *head* of r, while the conjunction $\mathtt{q_1(\bar{s}_1),\ \ldots,\ q_j(\bar{s}_j),\ not\ q_{j+1}(\bar{s}_{j+1}),\ \ldots,\ not\ q_m(\bar{s}_m)}$ 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 set of atoms appearing in positive and negative body literals, respectively, and $\mathit{Atoms}(r)$ for the set $H(r) \cup B^+(r) \cup B^-(r)$. A rule r is normal (or disjunction-free) if $|H(r)| = 1$, positive (or negation-free) if $B^-(r) = \emptyset$, a *fact* if both $B(r) = \emptyset$, $|H(r)| = 1$ and no variable appears in $H(r)$.

A *program* $\mathcal{P}$ is a finite set of rules; if all rules in it are positive (resp. normal), then $\mathcal{P}$ is a positive (resp. normal) program. Odd-cycle-free and stratified programs constitute other two interesting classes of programs. A predicate $\mathtt{p}$ appearing in the head of a rule r *depends* on each predicate $\mathtt{q}$ such that an atom $\mathtt{q(\bar{s})}$ belongs to $B(r)$; if $\mathtt{q(\bar{s})}$ belongs to $B^+(r)$, $\mathtt{p}$ depends on $\mathtt{q}$ positively, otherwise negatively. A program is *odd-cycle-free* if there is no cycle of dependencies involving an odd number of negative dependencies, while it is *stratified* if each cycle of dependencies involves only positive dependencies.

Given a predicate $\mathtt{p}$, a *defining rule* for $\mathtt{p}$ is a rule r such that some atom $\mathtt{p(\bar{t})}$ belongs to $H(r)$. If all defining rules of a predicate $\mathtt{p}$ are facts, then $\mathtt{p}$ is an *EDB predicate*; otherwise $\mathtt{p}$ is an *IDB predicate*[2]. Given a program $\mathcal{P}$, the set of rules having some IDB predicate in head is denoted by $\mathit{IDB}(\mathcal{P})$, while $\mathit{EDB}(\mathcal{P})$ denotes the remaining rules, that is, $\mathit{EDB}(\mathcal{P}) = \mathcal{P} \setminus \mathit{IDB}(\mathcal{P})$.

The set of constants appearing in a program $\mathcal{P}$ is the *universe* of $\mathcal{P}$ and is denoted by $U_{\mathcal{P}}$[3], while the set of ground atoms constructible from predicates in $\mathcal{P}$ with elements of $U_{\mathcal{P}}$ is the *base* of $\mathcal{P}$, denoted by $B_{\mathcal{P}}$. We call a term (atom, rule, or program) *ground* if it does not contain any variable. A ground atom $\mathtt{p(\bar{t})}$ (resp. a ground rule r_g) is an instance of an atom $\mathtt{p(\bar{t}')}$ (resp. of a rule r) if there is a substitution ϑ from the variables in $\mathtt{p(\bar{t}')}$ (resp. in r) to $U_{\mathcal{P}}$ such that

[1] We use the notation $\mathtt{\bar{t}}$ for a sequence of terms, for referring to atoms as $\mathtt{p(\bar{t})}$.

[2] *EDB* and *IDB* stand for Extensional Database and Intensional Database, respectively.

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

$\mathtt{p}(\bar{\mathtt{t}}) = \mathtt{p}(\bar{\mathtt{t}}')\vartheta$ (resp. $r_g = r\vartheta$). Given a program $\mathcal{P}$, $\mathit{Ground}(\mathcal{P})$ denotes the set of all instances of the rules in $\mathcal{P}$.

An *interpretation* I for a program $\mathcal{P}$ is a subset of $B_{\mathcal{P}}$. A positive ground literal $\mathtt{p}(\bar{\mathtt{t}})$ is true w.r.t. an interpretation I if $\mathtt{p}(\bar{\mathtt{t}}) \in I$; otherwise, it is false. A negative ground literal $\mathtt{not}\ \mathtt{p}(\bar{\mathtt{t}})$ is true w.r.t. I if and only if $\mathtt{p}(\bar{\mathtt{t}})$ is false w.r.t. I. The body of a ground rule r_g is true w.r.t. I if and only if all the body literals of r_g are true w.r.t. I, that is, if and only if $B^+(r_g) \subseteq I$ and $B^-(r_g) \cap I = \emptyset$. An interpretation I *satisfies* a ground rule $r_g \in \mathit{Ground}(\mathcal{P})$ if at least one atom in $H(r_g)$ is true w.r.t. I whenever the body of r_g is true w.r.t. I. An interpretation I is a *model* of a program $\mathcal{P}$ if I satisfies all the rules in $\mathit{Ground}(\mathcal{P})$.

Given an interpretation I for a program $\mathcal{P}$, the reduct of $\mathcal{P}$ w.r.t. I, denoted $\mathit{Ground}(\mathcal{P})^I$, is obtained by deleting from $\mathit{Ground}(\mathcal{P})$ all the rules r_g with $B^-(r_g) \cap I = \emptyset$, and then by removing all the negative literals from the remaining rules. The semantics of a program $\mathcal{P}$ is then given by the set $\mathcal{AS}(\mathcal{P})$ of the answer sets of $\mathcal{P}$, where an interpretation M is an answer set for $\mathcal{P}$ if and only if M is a subset-minimal model of $\mathit{Ground}(\mathcal{P})^M$.

Given a ground atom $\mathtt{p}(\bar{\mathtt{t}})$ and a program $\mathcal{P}$, $\mathtt{p}(\bar{\mathtt{t}})$ is a cautious (resp. brave) consequence of $\mathcal{P}$, denoted by $\mathcal{P} \models_c \mathtt{p}(\bar{\mathtt{t}})$ (resp. $\mathcal{P} \models_b \mathtt{p}(\bar{\mathtt{t}})$), if $\mathtt{p}(\bar{\mathtt{t}}) \in M$ for each (resp. some) $M \in \mathcal{AS}(\mathcal{P})$. Given a *query*[4] $\mathcal{Q} = \mathtt{g}(\bar{\mathtt{t}})?$, $\mathit{Ans}_c(\mathcal{Q}, \mathcal{P})$ (resp. $\mathit{Ans}_b(\mathcal{Q}, \mathcal{P})$) denotes the set of all the substitutions ϑ for the variables of $\mathtt{g}(\bar{\mathtt{t}})$ such that $\mathcal{P} \models_c \mathtt{g}(\bar{\mathtt{t}})\vartheta$ (resp. $\mathcal{P} \models_b \mathtt{g}(\bar{\mathtt{t}})\vartheta$). Two programs $\mathcal{P}$ and $\mathcal{P}'$ are cautious-equivalent (resp. brave-equivalent) w.r.t. a query $\mathcal{Q}$, denoted by $\mathcal{P} \equiv^c_{\mathcal{Q}} \mathcal{P}'$ (resp. $\mathcal{P} \equiv^b_{\mathcal{Q}} \mathcal{P}'$), if $\mathit{Ans}_c(\mathcal{Q}, \mathcal{P} \cup \mathcal{F}) = \mathit{Ans}_c(\mathcal{Q}, \mathcal{P}' \cup \mathcal{F})$ (resp. $\mathit{Ans}_b(\mathcal{Q}, \mathcal{P} \cup \mathcal{F}) = \mathit{Ans}_b(\mathcal{Q}, \mathcal{P}' \cup \mathcal{F})$) is guaranteed for each set of facts $\mathcal{F}$ defined over the EDB predicates of $\mathcal{P}$ and $\mathcal{P}'$.

2.2 Super-Consistent ASP Programs

We now introduce super-consistent ASP programs (ASP$^{\mathrm{sc}}$ programs), the main class of programs studied in this paper.

Definition 1 (ASP$^{\mathrm{sc}}$ **programs).** *A program* $\mathcal{P}$ *is* super-consistent *if, for every set of facts* $\mathcal{F}$*, the program* $\mathcal{P} \cup \mathcal{F}$ *is consistent, that is,* $\mathcal{AS}(\mathcal{P} \cup \mathcal{F}) \neq \emptyset$*. Let* ASP$^{\mathrm{sc}}$ *denote the set of all super-consistent programs.*

Deciding whether a program $\mathcal{P}$ is ASP$^{\mathrm{sc}}$ is computable. Indeed, if $\mathcal{P}$ is not ASP$^{\mathrm{sc}}$, then there is a set of facts $\mathcal{F}$ such that $\mathcal{P} \cup \mathcal{F}$ is inconsistent. Such an $\mathcal{F}$ can be chosen among all possible sets of ground atoms constructible by combining predicates of $\mathcal{P}$ with constants in $U_{\mathcal{P}} \cup \{\xi_X \mid X \text{ is a variable of } \mathcal{P}\}$ (assuming different rules have different variable names and ξ_X does not belong to $U_{\mathcal{P}}$): If the inconsistency is not due (only) to atoms in $B_{\mathcal{P}}$ but new constant symbols are required, then the choice of these symbols is negligible and the possibility

[4] The queries considered here allow only atoms for simplicity; 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 $\mathtt{p}(\bar{\mathtt{t}})$ such that $\mathtt{p}$ is a predicate not occurring in $\mathcal{P}$ and $\bar{\mathtt{t}}$ are the arguments of $\mathcal{Q}$.

to instantiate each variable with a different constant is sufficient to trigger the inconsistency.

ASP^{sc} programs constitute an interesting class of programs, properly including odd-cycle-free programs (hence also stratified programs). Indeed, every odd-cycle-free program admits at least one answer set and remains odd-cycle-free even if an arbitrary set of facts is added to its rules. On the other hand, there are programs having odd-cycles that are ASP^{sc}.

Example 1. Consider the following program:

```
a v b.     a :- not a, not b.
```

Even if an odd-cycle involving `a` is present in the dependency graph, the program above is ASP^{sc}. Indeed, the first rule assures that the body of the second rule is false in every model, then annihilating the odd-cycle. □

3 Magic-Set Techniques

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. Dynamic Magic Sets (`DMS`) are an extension of this technique, which has been proved to be sound and complete for query answering over ASP programs with stratified negation.

In this section, we first recall the `DMS` algorithm, as presented in [9]. We then show how to apply `DMS` to ASP^{sc} programs and formally prove the correctness of query answering for this class.

3.1 Dynamic Magic Sets

The method of [9][5] is structured in three main phases.

(1) Adornment. The key idea is to materialize the binding information for IDB predicates that would be propagated during a top-down computation, like for instance the one adopted by Prolog. According to this kind of evaluation, all the rules r such that $\mathtt{g}(\bar{\mathtt{t}}') \in H(r)$ (where $\mathtt{g}(\bar{\mathtt{t}}')\vartheta = \mathcal{Q}$ for some substitution ϑ) are considered in a first step. Then the atoms in $Atoms(r\vartheta)$ different from $\mathcal{Q}$ are considered as new queries and the procedure is iterated.

Note that during this process the information about *bound* (i.e. non-variable) arguments in the query is "passed" to the other atoms in the rule. Moreover, it is assumed that the rule is processed in a certain sequence, and processing an atom may bind some of its arguments for subsequently considered atoms, thus "generating" and "passing" bindings. Therefore, whenever an atom is processed, each of its argument is considered to be either *bound* (`b`) or *free* (`f`).

[5] For a detailed description of the standard magic set technique we refer to [6].

```
Input: An ASP$^{sc}$ program $\mathcal{P}$, and a query $\mathcal{Q} = \mathtt{g}(\bar{\mathtt{t}})$?
Output: The optimized program $\mathtt{DMS}(\mathcal{Q}, \mathcal{P})$.
var $S$: set of adorned predicates; $modifiedRules_{\mathcal{Q},\mathcal{P}}$, $magicRules_{\mathcal{Q},\mathcal{P}}$: set of rules;
begin
  1. $S := \emptyset$; $modifiedRules_{\mathcal{Q},\mathcal{P}} := \emptyset$; $magicRules_{\mathcal{Q},\mathcal{P}} := \{\textbf{\textit{BuildQuerySeed}}(\mathcal{Q}, \mathcal{P}, S)\}$;
  2. while $S \neq \emptyset$ do
  3.    $\mathtt{p}^{\alpha}$ := an element of $S$;   $S := S \setminus \{\mathtt{p}^{\alpha}\}$;
  4.    for each rule $r \in \mathcal{P}$ and for each atom $\mathtt{p}(\bar{\mathtt{t}}) \in H(r)$ do
  5.       $r^{a} := \textbf{\textit{Adorn}}(r, \mathtt{p}^{\alpha}, S)$;
  6.       $magicRules_{\mathcal{Q},\mathcal{P}} := magicRules_{\mathcal{Q},\mathcal{P}} \bigcup \textbf{\textit{Generate}}(r^{a})$;
  7.       $modifiedRules_{\mathcal{Q},\mathcal{P}} := modifiedRules_{\mathcal{Q},\mathcal{P}} \bigcup \{ \textbf{\textit{Modify}}(r^{a}) \}$;
  8.    end for
  9. end while
 10. $\mathtt{DMS}(\mathcal{Q}, \mathcal{P}) := magicRules_{\mathcal{Q},\mathcal{P}} \cup modifiedRules_{\mathcal{Q},\mathcal{P}} \cup EDB(\mathcal{P})$;
 11. return $\mathtt{DMS}(\mathcal{Q}, \mathcal{P})$;
end.
```

Fig. 1. Dynamic Magic Set algorithm (DMS) for ASP^{sc} programs.

The specific propagation strategy adopted in a 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 originate and propagate [8, 10]. Thus, in this phase, adornments are first created for the query predicate. Then each adorned predicate is used to propagate its information to the other atoms of the rules defining it according to a SIPS, thereby simulating a top-down evaluation. 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.
(2) Generation. The adorned rules are then used to generate *magic rules* defining *magic predicates*, which represent the atoms relevant for answering the input query. The bodies of magic rules contain the atoms required for binding the arguments of some atom, following the adopted SIPS.
(3) Modification. Subsequently, magic atoms are added to the bodies of the adorned rules in order to limit the range of the head variables, thus avoiding the inference of facts which are irrelevant for the query. The resulting rules are called *modified rules.*

The complete rewritten program consists of the magic and modified rules (together with the original EDB). Given a stratified program $\mathcal{P}$, a query $\mathcal{Q}$, and the rewritten program $\mathcal{P}'$, $\mathcal{P}$ and $\mathcal{P}'$ are equivalent w.r.t. $\mathcal{Q}$, i.e., $\mathcal{P} \equiv^{b}_{\mathcal{Q}} \mathcal{P}'$ and $\mathcal{P} \equiv^{c}_{\mathcal{Q}} \mathcal{P}'$ hold [9].

3.2 Applying DMS to ASP^{sc} Programs

The algorithm DMS implementing the Magic-Set technique described in the previous section is reported in Figure 1. The algorithm exploits a set S for storing all the adorned predicates to be used for propagating the binding of the query and, after all the adorned predicates are processed, 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 $modifiedRules_{\mathcal{Q},\mathcal{P}}$ and $magicRules_{\mathcal{Q},\mathcal{P}}$, respectively.

We note that, even if the DMS method is presented for stratified ASP programs, this restriction is not required by the algorithm. Indeed, in [9], stratification is only used to prove query equivalence of the rewritten program with the original program. Here we claim that DMS can be correctly applied to a larger class of programs, precisely that of ASP^{sc} programs.

We now describe the applicability of DMS to ASP^{sc} programs, and in the next section we will prove its correctness for this class of programs. For illustrating the technique we will use the following running example.

Example 2 (Related [10]). A genealogy graph storing information of relationship (father/brother) among people is given, from which a non-deterministic "ancestor" relation can be derived. Assuming the genealogy graph is encoded by facts $\mathtt{rel}(\mathtt{p_1},\mathtt{p_2})$ when $\mathtt{p_1}$ is known to be related to $\mathtt{p_2}$, that is, when $\mathtt{p_1}$ is the father or a brother of $\mathtt{p_2}$, the *ancestor* relation can be derived by the following ASP^{sc} program $\mathcal{P}_{rel}$:

$$\begin{array}{lll}
r_1: & \mathtt{fath(X,Y)} & :- \ \mathtt{rel(X,Y),\ not\ brot(X,Y).}\\
r_2: & \mathtt{brot(X,Y)} & :- \ \mathtt{rel(X,Y),\ not\ fath(X,Y).}\\
r_3: & \mathtt{anc(X,Y)} & :- \ \mathtt{fath(X,Y).}\\
r_4: & \mathtt{anc(X,Y)} & :- \ \mathtt{fath(X,Z),\ anc(Z,Y).}
\end{array}$$

Given two people $\mathtt{p_1}$ and $\mathtt{p_2}$, we consider a query $\mathcal{Q}_{rel} = \mathtt{anc}(\mathtt{p_1},\mathtt{p_2})$? asking whether $\mathtt{p_1}$ is an ancestor of $\mathtt{p_2}$. □

The computation starts in step *1* by initializing S and $\mathit{modifiedRules}_{\mathcal{Q},\mathcal{P}}$ to the empty set. Then the function ***BuildQuerySeed***$(\mathcal{Q},\mathcal{P},S)$ is used for storing the magic seed $\mathtt{magic}(\mathtt{g}^{\alpha}(\bar{\mathtt{t}}))$. in $\mathit{magicRules}_{\mathcal{Q},\mathcal{P}}$, where α is a string having a b in position i if $\mathtt{t_i}$ is a constant, or an f if $\mathtt{t_i}$ is a variable. Intuitively, the magic seed states that atoms matching the input query are relevant. In addition, ***BuildQuerySeed***$(\mathcal{Q},\mathcal{P},S)$ adds the adorned predicate $\mathtt{magic\_g}^{\alpha}$ into the set S.

Example 3. Given the query $\mathcal{Q}_{rel} = \mathtt{anc}(\mathtt{p_1},\mathtt{p_2})$? and the program $\mathcal{P}_{rel}$, ***Build-QuerySeed***$(\mathcal{Q}_{rel},\mathcal{P}_{rel},S)$ creates the fact $\mathtt{magic\_anc}^{\mathtt{bb}}(\mathtt{p_1},\mathtt{p_2})$. and inserts $\mathtt{anc}^{\mathtt{bb}}$ in S. □

The core of the algorithm (steps *2–9*) 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 $\mathtt{p}^{\alpha}$ is removed from S in step *3*, and its binding is propagated in each (disjunctive) rule $r \in \mathcal{P}$ of the form

$$\begin{array}{ll}
r: \ \mathtt{p}(\bar{\mathtt{t}})\ \mathtt{v}\ \mathtt{p_1}(\bar{\mathtt{t}}_1)\ \mathtt{v}\ \cdots\ \mathtt{v}\ \mathtt{p_n}(\bar{\mathtt{t}}_n) :- & \mathtt{q_1}(\bar{\mathtt{s}}_1),\ \ldots,\ \mathtt{q_j}(\bar{\mathtt{s}}_j),\\
& \mathtt{not}\ \mathtt{q_{j+1}}(\bar{\mathtt{s}}_{j+1}),\ \ldots,\ \mathtt{not}\ \mathtt{q_m}(\bar{\mathtt{s}}_m).
\end{array}$$

(with $\mathtt{n} \geq 0$) having an atom $\mathtt{p}(\bar{\mathtt{t}})$ in the head (note that the rule r is processed as often as head atoms with predicate p occur; steps *4–8*).

(1) Adornment. Step *5* implements the adornment of the rule according to a fixed SIPS specifically conceived for disjunctive programs.

Definition 2 (SIPS). *A* SIPS *for a rule r w.r.t. a binding α for an atom* $\mathtt{p}(\bar{\mathtt{t}}) \in H(r)$ *is a pair* $(\prec_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})}, f_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})})$*, where:*

1. $\prec_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})}$ *is a strict partial order over the atoms in* $\mathit{Atoms}(r)$*, such that:*
   (a) $\mathtt{p}(\bar{\mathtt{t}}) \prec_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})} \mathtt{q}(\bar{\mathtt{s}})$*, for all atoms* $\mathtt{q}(\bar{\mathtt{s}}) \in \mathit{Atoms}(r)$ *different from* $\mathtt{p}(\bar{\mathtt{t}})$*;*
   (b) *for each pair of atoms* $\mathtt{q}(\bar{\mathtt{s}}) \in (H(r) \setminus \{\mathtt{p}(\bar{\mathtt{t}})\}) \cup B^-(r)$ *and* $\mathtt{b}(\bar{\mathtt{z}}) \in \mathit{Atoms}(r)$*,* $\mathtt{q}(\bar{\mathtt{s}}) \prec_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})} \mathtt{b}(\bar{\mathtt{z}})$ *does not hold; and,*
2. $f_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})}$ *is a function assigning to each atom* $\mathtt{q}(\bar{\mathtt{s}}) \in \mathit{Atoms}(r)$ *a subset of the variables in* $\bar{\mathtt{s}}$*—intuitively, those made bound when processing* $\mathtt{q}(\bar{\mathtt{s}})$*.*

The adornments for a rule r w.r.t. an (adorned) head atom $\mathtt{p}^\alpha(\bar{\mathtt{t}})$ are precisely dictated by $(\prec_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})}, f_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})})$; in particular, a variable $\mathtt{X}$ of an atom $\mathtt{q}(\bar{\mathtt{s}})$ in r is bound if and only if either:

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

The function $\boldsymbol{Adorn}(r, \mathtt{p}^\alpha, S)$ produces an adorned disjunctive rule r^a from an adorned predicate $\mathtt{p}^\alpha$ and a suitable unadorned rule r, by inserting all newly adorned predicates in S. Hence, in step *5* the rule r^a is of the form

$$\begin{array}{l} r^a :\ \mathtt{p}^\alpha(\bar{\mathtt{t}})\ \mathtt{v}\ \mathtt{p}_1^{\alpha_1}(\bar{\mathtt{t}}_1)\ \mathtt{v}\ \cdots\ \mathtt{v}\ \mathtt{p}_\mathtt{n}^{\alpha_\mathtt{n}}(\bar{\mathtt{t}}_\mathtt{n}) \ \texttt{:-}\ \mathtt{q}_1^{\beta_1}(\bar{\mathtt{s}}_1),\ \ldots,\ \mathtt{q}_\mathtt{j}^{\beta_\mathtt{j}}(\bar{\mathtt{s}}_\mathtt{j}), \\ \qquad\qquad\qquad\qquad\qquad \mathtt{not}\ \mathtt{q}_{\mathtt{j+1}}^{\beta_{\mathtt{j+1}}}(\bar{\mathtt{s}}_{\mathtt{j+1}}),\ \ldots,\ \mathtt{not}\ \mathtt{q}_\mathtt{m}^{\beta_\mathtt{m}}(\bar{\mathtt{s}}_\mathtt{m}). \end{array}$$

where each $\alpha_1, \ldots, \alpha_\mathtt{n}, \beta_1, \ldots, \beta_\mathtt{m}$ is either a string representing the bindings defined in 1. and 2. above (for IDB atoms), or the empty string (for EDB atoms).

Example 4. Let us resume from Example 3. We are supposing the adopted SIPS is passing the bindings whenever possible, in particular

$$\begin{array}{rcl} \mathtt{fath(X,Y)} & \prec_{r_1}^{\mathtt{fath^{bb}(X,Y)}} & \mathtt{rel(X,Y)} \\ \mathtt{fath(X,Y)} & \prec_{r_1}^{\mathtt{fath^{bb}(X,Y)}} & \mathtt{brot(X,Y)} \end{array} \qquad \begin{array}{l} f_{r_1}^{\mathtt{fath^{bb}(X,Y)}}(\mathtt{fath(X,Y)}) = \{\mathtt{X,Y}\} \\ f_{r_1}^{\mathtt{fath^{bb}(X,Y)}}(\mathtt{rel(X,Y)}) = \{\mathtt{X,Y}\} \\ f_{r_1}^{\mathtt{fath^{bb}(X,Y)}}(\mathtt{brot(X,Y)}) = \{\mathtt{X,Y}\} \end{array}$$

$$\begin{array}{rcl} \mathtt{fath(X,Y)} & \prec_{r_1}^{\mathtt{fath^{bf}(X,Y)}} & \mathtt{rel(X,Y)} \\ \mathtt{fath(X,Y)} & \prec_{r_1}^{\mathtt{fath^{bf}(X,Y)}} & \mathtt{brot(X,Y)} \\ \mathtt{rel(X,Y)} & \prec_{r_1}^{\mathtt{fath^{bf}(X,Y)}} & \mathtt{brot(X,Y)} \end{array} \qquad \begin{array}{l} f_{r_1}^{\mathtt{fath^{bf}(X,Y)}}(\mathtt{fath(X,Y)}) = \{\mathtt{X}\} \\ f_{r_1}^{\mathtt{fath^{bf}(X,Y)}}(\mathtt{rel(X,Y)}) = \{\mathtt{X,Y}\} \\ f_{r_1}^{\mathtt{fath^{bf}(X,Y)}}(\mathtt{brot(X,Y)}) = \{\mathtt{X,Y}\} \end{array}$$

$$\begin{array}{rcl} \mathtt{brot(X,Y)} & \prec_{r_2}^{\mathtt{brot^{bb}(X,Y)}} & \mathtt{rel(X,Y)} \\ \mathtt{brot(X,Y)} & \prec_{r_2}^{\mathtt{brot^{bb}(X,Y)}} & \mathtt{fath(X,Y)} \end{array} \qquad \begin{array}{l} f_{r_2}^{\mathtt{brot^{bb}(X,Y)}}(\mathtt{brot(X,Y)}) = \{\mathtt{X,Y}\} \\ f_{r_2}^{\mathtt{brot^{bb}(X,Y)}}(\mathtt{rel(X,Y)}) = \{\mathtt{X,Y}\} \\ f_{r_2}^{\mathtt{brot^{bb}(X,Y)}}(\mathtt{fath(X,Y)}) = \{\mathtt{X,Y}\} \end{array}$$

$$\begin{array}{rcl} \mathtt{anc(X,Y)} & \prec_{r_3}^{\mathtt{anc^{bb}(X,Y)}} & \mathtt{fath(X,Y)} \end{array} \qquad \begin{array}{l} f_{r_3}^{\mathtt{anc^{bb}(X,Y)}}(\mathtt{anc(X,Y)}) = \{\mathtt{X,Y}\} \\ f_{r_3}^{\mathtt{anc^{bb}(X,Y)}}(\mathtt{fath(X,Y)}) = \{\mathtt{X,Y}\} \end{array}$$

$$\begin{array}{rcl} \mathtt{anc(X,Y)} & \prec_{r_4}^{\mathtt{anc^{bb}(X,Y)}} & \mathtt{fath(X,Z)} \\ \mathtt{anc(X,Y)} & \prec_{r_4}^{\mathtt{anc^{bb}(X,Y)}} & \mathtt{anc(Z,Y)} \\ \mathtt{fath(X,Z)} & \prec_{r_4}^{\mathtt{anc^{bb}(X,Y)}} & \mathtt{anc(Z,Y)} \end{array} \qquad \begin{array}{l} f_{r_4}^{\mathtt{anc^{bb}(X,Y)}}(\mathtt{anc(X,Y)}) = \{\mathtt{X,Y}\} \\ f_{r_4}^{\mathtt{anc^{bb}(X,Y)}}(\mathtt{fath(X,Z)}) = \{\mathtt{X,Z}\} \\ f_{r_4}^{\mathtt{anc^{bb}(X,Y)}}(\mathtt{anc(Z,Y)}) = \{\mathtt{Z,Y}\} \end{array}$$

When $\texttt{anc}^{\texttt{bb}}$ is removed from the set S, r_3 and r_4[6] are adorned:

$$\begin{array}{rl} r_3^a : & \texttt{anc}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \ \ {:}{-} \ \ \texttt{fath}^{\texttt{bb}}(\texttt{X},\texttt{Y}). \\ r_4^a : & \texttt{anc}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \ \ {:}{-} \ \ \texttt{fath}^{\texttt{bf}}(\texttt{X},\texttt{Z}),\ \texttt{anc}^{\texttt{bb}}(\texttt{Z},\texttt{Y}). \end{array}$$

The adorned predicates $\texttt{fath}^{\texttt{bb}}$ and $\texttt{fath}^{\texttt{bf}}$ are added to S. Then, $\texttt{fath}^{\texttt{bb}}$ is removed from S and r_1 is adorned:

$$r_{1,1}^a : \ \texttt{fath}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \ \ {:}{-} \ \ \texttt{rel}(\texttt{X},\texttt{Y}),\ \texttt{not}\ \texttt{brot}^{\texttt{bb}}(\texttt{X},\texttt{Y}).$$

Thus, $\texttt{brot}^{\texttt{bb}}$ is added to S. We then remove $\texttt{fath}^{\texttt{bf}}$ from S and adorn r_1:

$$r_{1,2}^a : \ \texttt{fath}^{\texttt{bf}}(\texttt{X},\texttt{Y}) \ \ {:}{-} \ \ \texttt{rel}(\texttt{X},\texttt{Y}),\ \texttt{not}\ \texttt{brot}^{\texttt{bb}}(\texttt{X},\texttt{Y}).$$

In this case nothing is added to S. Finally, $\texttt{brot}^{\texttt{bb}}$ is removed from S and r_2 is adorned:

$$r_2^a : \ \texttt{brot}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \ \ {:}{-} \ \ \texttt{rel}(\texttt{X},\texttt{Y}),\ \texttt{not}\ \texttt{fath}^{\texttt{bb}}(\texttt{X},\texttt{Y}).$$

□

(2) Generation. The algorithm uses the adorned rules for generating and collecting the magic rules in step *6*. For an adorned atom $\texttt{p}^\alpha(\bar{\texttt{t}})$, let $\texttt{magic}(\texttt{p}^\alpha(\bar{\texttt{t}}))$ be its *magic version* defined as the atom $\texttt{magic\_p}^\alpha(\bar{\texttt{t}}')$, where $\bar{\texttt{t}}'$ is obtained from $\bar{\texttt{t}}$ by eliminating all arguments corresponding to an $\texttt{f}$ label in α, and where $\texttt{magic\_p}^\alpha$ is a new predicate symbol (for simplicity denoted by attaching the prefix "$\texttt{magic\_}$" to the predicate symbol $\texttt{p}^\alpha$). Then, if $\texttt{q}_\texttt{i}^{\beta_\texttt{i}}(\bar{\texttt{s}}_\texttt{i})$ is an adorned atom (i.e., β_i is not the empty string) in an adorned rule r^a having $\texttt{p}^\alpha(\bar{\texttt{t}})$ in head, ***Generate***(r^a) produces a magic rule r^* such that (i) $H(r^*) = \{\texttt{magic}(\texttt{q}_\texttt{i}^{\beta_\texttt{i}}(\bar{\texttt{s}}_\texttt{i}))\}$ and (ii) $B(r^*)$ is the union of $\{\texttt{magic}(\texttt{p}^\alpha(\bar{\texttt{t}}))\}$ and the set of all the atoms $\texttt{q}_\texttt{j}^{\beta_\texttt{j}}(\bar{\texttt{s}}_\texttt{j}) \in \mathit{Atoms}(r)$ such that $\texttt{q}_\texttt{j}(\bar{\texttt{s}}_\texttt{j}) \prec_r^\alpha \texttt{q}_\texttt{i}(\bar{\texttt{s}}_\texttt{i})$.

Example 5. In the program of Example 4, the magic rules produced are

$$\begin{array}{rl} r_3^* & : \texttt{magic\_fath}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \ \ {:}{-} \ \ \texttt{magic\_anc}^{\texttt{bb}}(\texttt{X},\texttt{Y}). \\ r_{4,1}^* & : \texttt{magic\_fath}^{\texttt{bf}}(\texttt{X}) \ \ {:}{-} \ \ \texttt{magic\_anc}^{\texttt{bb}}(\texttt{X},\texttt{Y}). \\ r_{4,2}^* & : \texttt{magic\_anc}^{\texttt{bb}}(\texttt{Z},\texttt{Y}) \ \ {:}{-} \ \ \texttt{magic\_anc}^{\texttt{bb}}(\texttt{X},\texttt{Y}),\ \texttt{fath}(\texttt{X},\texttt{Z}). \\ r_{1,1}^* & : \texttt{magic\_brot}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \ \ {:}{-} \ \ \texttt{magic\_fath}^{\texttt{bb}}(\texttt{X},\texttt{Y}). \\ r_{1,2}^* & : \texttt{magic\_brot}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \ \ {:}{-} \ \ \texttt{magic\_fath}^{\texttt{bf}}(\texttt{X}),\ \texttt{rel}(\texttt{X},\texttt{Y}). \\ r_2^* & : \texttt{magic\_fath}^{\texttt{bb}}(\texttt{X},\texttt{Y}) \ \ {:}{-} \ \ \texttt{magic\_brot}^{\texttt{bb}}(\texttt{X},\texttt{Y}). \end{array}$$

□

(3) Modification. In step *7* the modified rules are generated and collected. A modified rule r' is obtained from an adorned rule r^a by adding to its body a magic atom $\texttt{magic}(\texttt{p}^\alpha(\bar{\texttt{t}}))$ for each atom $\texttt{p}^\alpha(\bar{\texttt{t}}) \in H(r^a)$ and by stripping off the

[6] Note that, according to the SIPS described above, variable $\texttt{Z}$ in $\texttt{anc}(\texttt{Z},\texttt{Y})$ is considered bound because of $\texttt{fath}(\texttt{X},\texttt{Z}) \prec_{r_4}^{\texttt{anc}^{\texttt{bb}}(\texttt{X},\texttt{Y})} \texttt{anc}(\texttt{Z},\texttt{Y})$ and $f_{r_4}^{\texttt{anc}^{\texttt{bb}}(\texttt{X},\texttt{Y})}(\texttt{fath}(\texttt{X},\texttt{Z})) = \{\texttt{X},\texttt{Z}\}$. Choosing a different SIPS would result in a different (still correct) program.

adornments of the original atoms. Hence, the function ***Modify***(r^a) constructs a rule r' of the form

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

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

Example 6. In our running example, we derive the following set of modified rules:

$$\begin{array}{lll} r'_3 & : \mathtt{anc(X,Y)} & \mathrel{:-} \mathtt{magic\_anc^{bb}(X,Y),\ fath(X,Y).} \\ r'_4 & : \mathtt{anc(X,Y)} & \mathrel{:-} \mathtt{magic\_anc^{bb}(X,Y),\ fath(X,Z),\ anc(Z,Y).} \\ r'_{1,1} & : \mathtt{fath(X,Y)} & \mathrel{:-} \mathtt{magic\_fath^{bb}(X,Y),\ rel(X,Y),\ not\ brot(X,Y).} \\ r'_{1,2} & : \mathtt{fath(X,Y)} & \mathrel{:-} \mathtt{magic\_fath^{bf}(X,Y),\ rel(X,Y),\ not\ brot(X,Y).} \\ r'_2 & : \mathtt{brot(X,Y)} & \mathrel{:-} \mathtt{magic\_brot^{bb}(X,Y),\ rel(X,Y),\ not\ fath(X,Y).} \end{array}$$

The optimized program $\mathtt{DMS}(\mathcal{Q}_{rel}, \mathcal{P}_{rel})$ comprises the above modified rules as well as the magic rules in Example 5, and the magic seed $\mathtt{magic\_anc^{bb}(p_1,p_2)}$. (together with the original EDB). □

3.3 Query Equivalence Results

We conclude the presentation of the DMS algorithm by formally proving its correctness. This section essentially follows [9], to which we refer for the details, while here we highlight the necessary considerations for generalizing the results of [9] to ASP^{sc} queries. Throughout this section, we use the well established notion of unfounded set for disjunctive programs with negation defined in [11]. Since we deal with total interpretations, represented as the set of atoms interpreted as true, the definition of unfounded set can be restated as follows.

Definition 3 (Unfounded sets). *Let I be an interpretation for a program $\mathcal{P}$, and $X \subseteq B_{\mathcal{P}}$ be a set of ground atoms. Then X is an* unfounded set *for $\mathcal{P}$ w.r.t. I if and only if for each ground rule $r_g \in Ground(\mathcal{P})$ with $X \cap H(r_g) \neq \emptyset$, either* (1.*a*) *$B^+(r_g) \not\subseteq I$, or* (1.*b*) *$B^-(r_g) \cap I \neq \emptyset$, or* (2) *$B^+(r_g) \cap X \neq \emptyset$, or* (3) *$H(r_g) \cap (I \setminus X) \neq \emptyset$.*

Intuitively, conditions (1.*a*), (1.*b*) and (3) check if the rule is satisfied by I regardless of the atoms in X, while condition (2) assures that the rule can be satisfied by taking the atoms in X as false. Therefore, the next theorem immediately follows from the characterization of unfounded sets in [11].

Theorem 1. *Let I be an interpretation for a program $\mathcal{P}$. Then, for any answer set $M \supseteq I$ of $\mathcal{P}$, and for each unfounded set X of $\mathcal{P}$ w.r.t. I, $M \cap X = \emptyset$ holds.*

We now prove the correctness of the DMS strategy by showing that it is *sound* and *complete*. In both parts of the proof, we exploit the following set of atoms.

Definition 4 (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* $\mathtt{killed}^M_{\mathcal{Q},\mathcal{P}}(N)$ *of the* killed atoms *w.r.t. M and N is defined as:*

$$\{\, \mathtt{p}(\bar{\mathtt{t}}) \in B_{\mathcal{P}} \setminus N \mid \textit{either } \mathtt{p} \textit{ is EDB, or some } \mathtt{magic}(\mathtt{p}^{\alpha}(\bar{\mathtt{t}})) \textit{ belongs to } N \,\}.$$

Thus, killed atoms are either false instances of some EDB predicate, or false atoms which are relevant for $\mathcal{Q}$ (since a magic atom exists in N). Therefore, we expect that these atoms are also false in any answer set for $\mathcal{P}$ containing $M \cap B_{\mathcal{P}}$.

Proposition 1. *Let M be a model for* $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$*, and $N \subseteq M$ a model of* $Ground(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^M$*. Then* $\mathtt{killed}^M_{\mathcal{Q},\mathcal{P}}(N)$ *is an unfounded set for $\mathcal{P}$ w.r.t. $M \cap B_{\mathcal{P}}$.*

Proof. See [9], proof of Proposition 3.15. □

For proving the completeness of the algorithm we provide a construction for passing from an interpretation for $\mathcal{P}$ to one for $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$.

Definition 5 (Magic variant). *Let I be an interpretation for $\mathcal{P}$. We define an interpretation* $\mathtt{var}^{\infty}_{\mathcal{Q},\mathcal{P}}(I)$ *for* $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$*, called the magic variant of I w.r.t. $\mathcal{Q}$ and $\mathcal{P}$, as the fixpoint of the following sequence:*

$$\begin{aligned}
\mathtt{var}^{0}_{\mathcal{Q},\mathcal{P}}(I) &= \mathit{EDB}(\mathcal{P})\\
\mathtt{var}^{i+1}_{\mathcal{Q},\mathcal{P}}(I) &= \mathtt{var}^{i}_{\mathcal{Q},\mathcal{P}}(I) \cup \{\mathtt{p}(\bar{\mathtt{t}}) \in I \mid \textit{some } \mathtt{magic}(\mathtt{p}^{\alpha}(\bar{\mathtt{t}})) \textit{ belongs to } \mathtt{var}^{i}_{\mathcal{Q},\mathcal{P}}(I)\}\\
&\quad \cup \{\mathtt{magic}(\mathtt{p}^{\alpha}(\bar{\mathtt{t}})) \mid \exists\, r^*_g \in Ground(\mathtt{DMS}(\mathcal{Q},\mathcal{P})) \textit{ such that}\\
&\qquad\quad \mathtt{magic}(\mathtt{p}^{\alpha}(\bar{\mathtt{t}})) \in H(r^*_g) \textit{ and } B^{+}(r^*_g) \subseteq \mathtt{var}^{i}_{\mathcal{Q},\mathcal{P}}(I)\}, \quad \forall i \geq 0
\end{aligned}$$

By definition, for a magic variant $\mathtt{var}^{\infty}_{\mathcal{Q},\mathcal{P}}(I)$ of an interpretation I for $\mathcal{P}$, $\mathtt{var}^{\infty}_{\mathcal{Q},\mathcal{P}}(I) \cap B_{\mathcal{P}} \subseteq I$ holds. More interesting, the magic variant of an answer set for $\mathcal{P}$ is in turn an answer set for $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ preserving the truth/falsity of $\mathcal{Q}\vartheta$, for every substitution ϑ.

Lemma 1. *For each answer set M of $\mathcal{P}$, there is an answer set M' of* $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ *(which is the magic variant of M) such that, for every substitution ϑ, $\mathcal{Q}\vartheta \in M$ if and only if $\mathcal{Q}\vartheta \in M'$.*

Proof. We can show that $M' = \mathtt{var}^{\infty}_{\mathcal{Q},\mathcal{P}}(I)$ is an answer set of $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ (see [9], proof of Lemma 3.21). Thus, since $\mathcal{Q}\vartheta$ belongs either to M' or to $\mathtt{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$, for every substitution ϑ, the claim follows by Proposition 1. □

Proving the soundness of the algorithm requires quite more attention. Indeed, if the technique is used for a program which is not ASP$^{\mathrm{sc}}$, the rewritten program might provide some wrong answer.

Example 7. Consider the program

```
edb(a).     q(X) v p(X) :- edb(X).     co(X) :- q(X), not co(X).
```

and the query $\mathtt{q(a)}$?. The program above admits a unique answer set, namely $\{\mathtt{edb(a)}, \mathtt{p(a)}\}$. Applying DMS will result in the following program:

$$\begin{array}{l}\mathtt{edb(a).} \quad \mathtt{magic\_q^b(a).} \quad \mathtt{magic\_p^b(X)} \ \mathtt{:-} \ \mathtt{magic\_q^b(X).} \\ \mathtt{magic\_q^b(X)} \ \mathtt{:-} \ \mathtt{magic\_p^b(X).} \\ \mathtt{q(X)} \ \mathtt{v} \ \mathtt{p(X)} \ \mathtt{:-} \ \mathtt{magic\_q^b(X),} \ \mathtt{magic\_p^b(X),} \ \mathtt{edb(X).}\end{array}$$

The rewritten program has two answer sets, namely $\{\mathtt{magic\_q^b(a)}, \mathtt{magic\_p^b(a)}, \mathtt{edb(a)}, \mathtt{p(a)}\}$ and $\{\mathtt{magic\_q^b(a)}, \mathtt{magic\_p^b(a)}, \mathtt{edb(a)}, \mathtt{q(a)}\}$. Therefore, $\mathtt{q(a)}$ is a brave consequence of the rewritten program but not of the original program. We note that the original program is not ASP^{sc}; indeed, an inconsistent program can be obtained by adding the fact $\mathtt{q(a)}$. □

The soundness of the algorithm for ASP^{sc} programs is proved below.

Lemma 2. *Let $\mathcal{Q}$ be a query over an* ASP^{sc} *program $\mathcal{P}$. Then, for each answer set M' of* $\mathtt{DMS}(\mathcal{Q}, \mathcal{P})$*, there is an answer set M of $\mathcal{P}$ such that, for every substitution ϑ, $\mathcal{Q}\vartheta \in M$ if and only if $\mathcal{Q}\vartheta \in M'$.*

Proof. Consider the program $\mathcal{P} \cup (M' \cap B_{\mathcal{P}})$, that is, the program obtained by adding to $\mathcal{P}$ a fact for each atom in $M' \cap B_{\mathcal{P}}$. Since $\mathcal{P}$ is ASP^{sc}, there is at least an answer set M for $\mathcal{P} \cup (M' \cap B_{\mathcal{P}})$. Clearly $M \supseteq M' \cap B_{\mathcal{P}}$; moreover, we can show that M is an answer set of $\mathcal{P}$ as well (see [9], proof of Lemma 3.16). Thus, since $\mathcal{Q}\vartheta$ belongs either to M' or to $\mathtt{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$, for every substitution ϑ, the claim follows by Proposition 1. □

From the above lemma, together with Lemma 1, the correctness of the Magic Set method with respect to query answering directly follows.

Theorem 2. *Let $\mathcal{P}$ be an* ASP^{sc} *program, and let $\mathcal{Q}$ be a query. Then both* $\mathtt{DMS}(\mathcal{Q}, \mathcal{P}) \equiv^{b}_{\mathcal{Q}} \mathcal{P}$ *and* $\mathtt{DMS}(\mathcal{Q}, \mathcal{P}) \equiv^{c}_{\mathcal{Q}} \mathcal{P}$ *hold.*

4 Implementation

The Dynamic Magic Set method (DMS) has been implemented and integrated into the core of DLV [2], as shown in the architecture reported in Figure 2.

In our prototype, the DMS algorithm is applied automatically by default when the user invokes DLV with `-FB` (brave reasoning) or `-FC` (cautious reasoning) 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).

Within DLV, DMS is applied immediately after parsing the program and the query by the *Magic Set Rewriter* module. The rewritten program is then processed by the *Intelligent Grounding* module and the *Model Generator* module using the standard DLV implementation. The only other modification with respect to standard DLV is for the output and its filtering: For ground queries, the

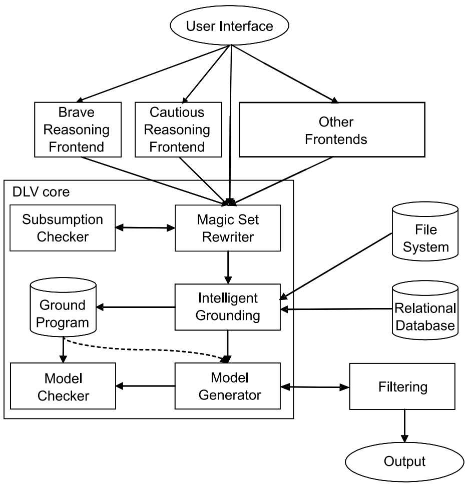


Fig. 2. Prototype system architecture.

witnessing answer set is no longer printed by default, but only if `--print-model` is specified, in which case the magic predicates are omitted from the output.

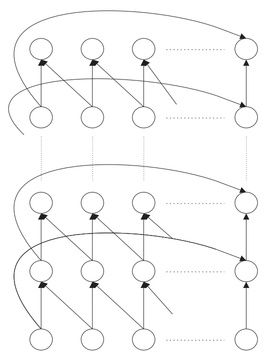
An executable of the DLV system supporting the Magic Set optimization is available at `http://www.dlvsystem.com/magic/`.

5 Experimental Results

In order to evaluate the impact of the proposed method, we have compared `DMS` with the traditional DLV evaluation without *Magic Sets* on several instances of the *Related* problem introduced in Section 3. In our benchmark, the structure of the "genealogy" graph consists of a square matrix of nodes connected as shown in Figure 3, and the instances are generated by varying the number of nodes (thus the number of persons in the genealogy) of the graph. We are interested in deciding whether the top-leftmost person can be an ancestor of the bottom-rightmost person (i.e., the benchmark is designed for brave reasoning). This setting has been used in [10] for a disjunctive, negation-free ASP encoding.

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.

The results for *Related* are reported in Figure 3. Without magic sets, DLV solves only the smallest instances, with a very steep increase in execution time. In this case, the exponential computational gain of `DMS` over DLV with no magic sets is due to the dynamic optimization of the model search phase resulting from our magic sets definition. Indeed, `DMS` include nondeterministic relevance



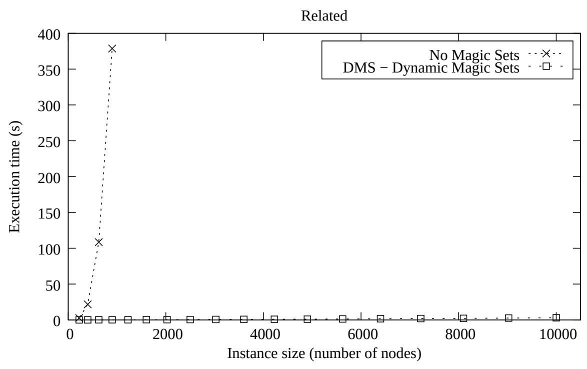


Fig. 3. *Related:* Instance structure (left) and average execution time (right).

information that 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.

6 Conclusion

The Magic Set method is one of the most well-known techniques for the optimization of positive recursive Datalog programs due to its efficiency and its generality. In this paper, we have elaborated on the issues addressed in [9]. In particular, we have showed the applicability of `DMS` for ASP^{sc} programs. With `DMS`, ASP computations can exploit the information provided by magic set predicates also during the nondeterministic stable model search, allowing for potentially exponential performance gains with respect to unoptimized evaluations.

We have established the correctness of `DMS` for ASP^{sc} by proving that the transformed program is query-equivalent to the original program. A strong relationship between magic sets and unfounded sets has been highlighted: The atoms that are relevant w.r.t. a stable model are either true or form an unfounded set.

`DMS` has been implemented in the DLV system. Experimental activities on the implemented prototype system evidenced that our implementation can outperform the standard evaluation in general also by an exponential factor. This is mainly due to the optimization of the model generation phase, which is specific of our Magic Set technique. However, we would like to point out that in general we expect a trade-off between the larger ground program due to the presence of ground magic atoms and its capability of pruning the search space.

As a final point, we would like to point out the relationship of this work to [12]: There, a Magic Set method for disjunction-free programs has been defined and proved to be correct for consistent programs. First, that method will not work for programs containing disjunction. Second, observe that consistent programs are not necessarily in ASP^{sc}; indeed the method of [12] has to take special precautions for relevant parts of the program that act as constraints (called *dangerous rules*) and thus may impede a relevant interpretation to be an answer set. The definition of ASP^{sc} implies that programs in this class cannot contain relevant dangerous rules, which allows for the simpler `DMS` strategy to work correctly.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
2. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic **7**(3) (2006) 499–562
3. Janhunen, T., Niemelä, I., Simons, P., You, J.H.: Partiality and Disjunctions in Stable Model Semantics. In Cohn, A.G., Giunchiglia, F., Selman, B., eds.: Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2000), April 12-15, Breckenridge, Colorado, USA, Morgan Kaufmann Publishers, Inc. (2000) 411–419
4. Lierler, Y.: Disjunctive Answer Set Programming via Satisfiability. In Baral, C., Greco, G., Leone, N., Terracina, G., eds.: Logic Programming and Nonmonotonic Reasoning — 8th International Conference, LPNMR'05, Diamante, Italy, September 2005, Proceedings. Volume 3662 of Lecture Notes in Computer Science., Springer Verlag (2005) 447–451
5. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-Driven Disjunctive Answer Set Solving. In Brewka, G., Lang, J., eds.: Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008), Sydney, Australia, AAAI Press (2008) 422–432
6. Ullman, J.D.: Principles of Database and Knowledge Base Systems. Computer Science Press (1989)
7. Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.D.: Magic Sets and Other Strange Ways to Implement Logic Programs. In: Proc. Int. Symposium on Principles of Database Systems. (1986) 1–16
8. Beeri, C., Ramakrishnan, R.: On the power of magic. Journal of Logic Programming **10**(1–4) (1991) 255–259
9. Alviano, M., Faber, W., Greco, G., Leone, N.: Magic sets for disjunctive datalog programs. Technical Report 09/2009, Dipartimento di Matematica, Università della Calabria, Italy (2009) `http://www.wfaber.com/research/papers/TRMAT092009.pdf`.
10. Greco, S.: Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. IEEE Transactions on Knowledge and Data Engineering **15**(2) (2003) 368–385
11. Leone, N., Rullo, P., Scarcello, F.: Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. Information and Computation **135**(2) (1997) 69–112
12. Faber, W., Greco, G., Leone, N.: Magic Sets and their Application to Data Integration. Journal of Computer and System Sciences **73**(4) (2007) 584–609