Technical Communications of the International Conference on Logic Programming, 2010 (Edinburgh), pp. 226–235
http://www.floc-conference.org/ICLP-home.html

DYNAMIC MAGIC SETS FOR DISJUNCTIVE DATALOG PROGRAMS

MARIO ALVIANO

Department of Mathematics, University of Calabria — 87036 Rende (CS), Italy
E-mail address: alviano@mat.unical.it

Abstract. Answer set programming (ASP) is a powerful formalism for knowledge representation and common sense reasoning that allows disjunction in rule heads and nonmonotonic negation in bodies. Magic Sets are a technique for optimizing query answering over logic programs and have been originally defined for standard Datalog, that is, ASP without disjunction and negation. Essentially, the input program is rewritten in order to identify a subset of the program instantiation which is sufficient for answering the query.

Dynamic Magic Sets (DMS) are an extension of this technique to ASP. The optimization provided by DMS can be exploited also during the nondeterministic phase of ASP systems. In particular, after some assumptions have been made during the computation, parts of the program may become irrelevant to a query (because of these assumptions). This allows for dynamic pruning of the search space, which may result in exponential performance gains.

DMS has been implemented in the DLV system and experimental results confirm the effectiveness of the technique.

Introduction

Answer set programming (ASP) is a powerful formalism for knowledge representation and common sense reasoning [Bar03]. Allowing disjunction in rule heads and nonmonotonic negation in bodies, ASP can express every query belonging to the complexity class Σ_2^{P} ($\mathrm{NP}^{\mathrm{NP}}$); the same expressive power is preserved even if negation is restricted to be used in a stratified way [Eit94].

Magic Sets are a technique for optimizing query answering over logic programs. ASP computations are typically characterized by two phases, namely *program instantiation* and *answer set search*. Program instantiation is deterministic and transforms the input program into an equivalent one with no variables. Answer set search is nondeterministic in general and works on the instantiated program.

Magic Sets have been originally defined for standard Datalog, that is, ASP without disjunction and negation. Essentially, the input program is rewritten in order to identify a subset of the program instantiation which is sufficient for answering the query. The

1998 ACM Subject Classification: Logic and constraint programming.

Key words and phrases: answer set programming, decidability, magic sets, disjunctive logic programs.

Thanks: The author is grateful to Wolfgang Faber, Gianluigi Greco and Nicola Leone for the fundamental contribution in achieving the results summarized in this article and reported in [Alv09]. 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.

Technical Communications of the 26th International Conference on Logic Programming, Edinburgh, July, 2010
Editors: Manuel Hermenegildo, Torsten Schaub
LIPIcs - Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.ICLP.2010.226

restriction of the instantiation is obtained by means of additional "magic" predicates, whose extensions represent relevant atoms w.r.t. the query.

An attempt to extend the method to (disjunctive) ASP has been done in [Gre03]. Magic set predicates of [Gre03] have a deterministic definition and, consequently, have the same extension in each answer set. Actually, this extension can always be computed during program instantiation, and so we call the technique of [Gre03] Static Magic Sets (SMS).

In the context of (disjunctive) ASP, there is no reason for having a deterministic definition of magic predicates. Indeed, while Datalog programs admit exactly one answer set, ASP programs can have several answer sets, each one representing a different, plausible scenario. Since atoms relevant in one scenario could be irrelevant in another (or also in each other), one expects that Magic Sets should capture this aspect and provide a *dynamic* optimization to the answer set search.

Our principal contributions concerning Magic Sets for ASP are stated below.

- We have defined Dynamic Magic Sets (DMS). With DMS, ASP computations can exploit the information provided by magic set predicates also during the nondeterministic answer set search, allowing for potentially exponential performance gains w.r.t. SMS. Indeed, the definition of our magic set predicates depends on the assumptions made during the computation, identifying the atoms that are relevant in the current (partial) scenario.
- We have established the correctness of DMS by proving that the transformed program is query-equivalent to the original program and we have highlighted a strong relationship between magic sets and unfounded sets: The atoms that are relevant w.r.t. an answer set are either true or form an unfounded set.
- We have implemented DMS in the DLV system and compared the performance of DLV with no magic sets, with SMS, and with DMS. The experimental results show that in many cases DMS yields a significant performance benefit. The system is available at `http://www.dlvsystem.com/magic/`.

The remainder of the paper is structured as follows. In Section 1, syntax and semantics of ASP are briefly mentioned. Dynamic Magic Sets for stratified ASP programs are introduced in Section 2. In Section 3, the implemented prototype system is briefly presented, while experimental result are discussed in Section 4. Finally, in Section 5, we draw our conclusion and discuss about future work we intend to address.

1. Answer Set Programming

In this section, we recall syntax and semantics of disjunctive ASP with stratified negation, the language for which we will introduce Dynamic Magic Sets in Section 2.2.

1.1. Syntax

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 $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_k)$ is an *atom*[1]. A *literal* is either an atom $\mathtt{p}(\bar{\mathtt{t}})$ (a positive literal), or an atom preceded by the *negation as failure* symbol `not` $\mathtt{p}(\bar{\mathtt{t}})$ (a negative literal). A *rule* r is of the form

$$\mathtt{p}_1(\bar{\mathtt{t}}_1) \ \mathtt{v} \ \cdots \ \mathtt{v} \ \mathtt{p}_n(\bar{\mathtt{t}}_n) \ \mathtt{:-} \ \mathtt{q}_1(\bar{\mathtt{s}}_1), \ \ldots, \ \mathtt{q}_j(\bar{\mathtt{s}}_j), \ \mathtt{not} \ \mathtt{q}_{j+1}(\bar{\mathtt{s}}_{j+1}), \ \ldots, \ \mathtt{not} \ \mathtt{q}_m(\bar{\mathtt{s}}_m).$$

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

where $\mathtt{p_1}(\bar{\mathtt{t}}_1)$, ..., $\mathtt{p_n}(\bar{\mathtt{t}}_n)$, $\mathtt{q_1}(\bar{\mathtt{s}}_1)$, ..., $\mathtt{q_m}(\bar{\mathtt{s}}_m)$ are atoms and $n \geq 1$, $m \geq j \geq 0$. The disjunction $\mathtt{p_1}(\bar{\mathtt{t}}_1)$ $\mathtt{v}$ $\cdots$ $\mathtt{v}$ $\mathtt{p_n}(\bar{\mathtt{t}}_n)$ is the *head* of r, while the conjunction $\mathtt{q_1}(\bar{\mathtt{s}}_1)$, ..., $\mathtt{q_j}(\bar{\mathtt{s}}_j)$, $\mathtt{not}\ q_{j+1}(\bar{s}_{j+1})$, ..., $\mathtt{not}\ q_m(\bar{s}_m)$ 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 $Atoms(r)$ for the set $H(r) \cup B^+(r) \cup B^-(r)$. Rules are assumed to be safe, that is, each variable appearing in a rule r also appears in $B^+(r)$. A rule r is positive (or negation-free) if $B^-(r) = \emptyset$, a *fact* if both $B(r) = \emptyset$ and $|H(r)| = 1$.

A *program* $\mathcal{P}$ is a finite set of rules; if all rules in it are positive, then $\mathcal{P}$ is a positive program. Stratified programs constitute another interesting class of programs. A predicate $\mathtt{p}$ appearing in the head of a rule r *depends* on each predicate $\mathtt{q}$ such that an atom $\mathtt{q}(\bar{\mathtt{s}})$ belongs to $B(r)$; if $\mathtt{q}(\bar{\mathtt{s}})$ belongs to $B^+(r)$, $\mathtt{p}$ depends on $\mathtt{q}$ positively, otherwise negatively. A program is *stratified* if each cycle of dependencies involves only positive dependencies.

1.2. Semantics

Given a predicate $\mathtt{p}$, a *defining rule* for $\mathtt{p}$ is a rule r such that some atom $\mathtt{p}(\bar{\mathtt{t}})$ 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 $IDB(\mathcal{P})$, while $EDB(\mathcal{P})$ denotes the remaining rules, that is, $EDB(\mathcal{P}) = \mathcal{P} \setminus IDB(\mathcal{P})$.

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 $\mathtt{p}(\bar{\mathtt{t}})$ (resp. a ground rule r_g) is an instance of an atom $\mathtt{p}(\bar{\mathtt{t}}')$ (resp. of a rule r) if there is a substitution ϑ from the variables in $\mathtt{p}(\bar{\mathtt{t}}')$ (resp. in r) to $U_\mathcal{P}$ such that $\mathtt{p}(\bar{\mathtt{t}}) = \mathtt{p}(\bar{\mathtt{t}}')\vartheta$ (resp. $r_g = r\vartheta$). Given a program $\mathcal{P}$, $Ground(\mathcal{P})$ 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 $r_g \in Ground(\mathcal{P})$ 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 $Ground(\mathcal{P})$.

Given an interpretation I for a program $\mathcal{P}$, the reduct of $\mathcal{P}$ w.r.t. I, denoted $Ground(\mathcal{P})^I$, is obtained by deleting from $Ground(\mathcal{P})$ 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 $Ground(\mathcal{P})^M$.

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) $M \in$

[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 $U_\mathcal{P}$.

$\mathcal{AS}(\mathcal{P})$. Given a *query* $\mathcal{Q} = \mathtt{g}(\bar{\mathtt{t}})$? (an atom)[4], $Ans_c(\mathcal{Q},\mathcal{P})$ (resp. $Ans_b(\mathcal{Q},\mathcal{P})$) 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 $Ans_c(\mathcal{Q},\mathcal{P}\cup\mathcal{F}) = Ans_c(\mathcal{Q},\mathcal{P}'\cup\mathcal{F})$ (resp. $Ans_b(\mathcal{Q},\mathcal{P}\cup\mathcal{F}) = Ans_b(\mathcal{Q},\mathcal{P}'\cup\mathcal{F})$) is guaranteed for each set of facts $\mathcal{F}$ defined over the EDB predicates of $\mathcal{P}$ and $\mathcal{P}'$.

2. Magic Sets Techniques

In this section, we first briefly discuss about Magic Sets in the literature; we then introduce Dynamic Magic Sets, our proposal for extending the standard technique to ASP.

2.1. Overview of the Existing Literature

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.

The key idea of Magic Sets 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`).

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 [Bee91, Gre03].

The first attempt to extend Magic Sets to disjunctive Datalog programs is due to [Gre03]. Magic predicates of [Gre03] identify a sizeable superset of all the atoms relevant to answer the given query. An important observation is that this set is defined in a deterministic way, which means that assumptions during the computation cannot be exploited for restricting 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. For this reason, we refer to the method of [Gre03] as *Static Magic Sets* (SMS).

Intuitively, it would be beneficial to also have a form of conditional relevance, exploiting also relevance for assumptions.[5] In the following, we propose a novel Magic Set method that guarantees semantic equivalence and also allows for the exploitation of conditional or dynamic relevance, overcoming a major drawback of SMS.

[4]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 $\mathcal{Q}$.

[5]Experimental evidence for this intuition is provided in Section 4.

2.2. Dynamic Magic Sets

Our proposal to extend Magic Sets to (disjunctive) ASP relies on the observation that atoms relevant in one answer set could be irrelevant in another (or also in each other). DMS capture this aspect, providing a *dynamic* optimization to the answer set search.

In order to properly describe the proposed Magic Set method, we need some additional definition and notation. First, we can materialize the binding information for IDB predicates by means of adorned atoms.

Definition 2.1 (Adorned atom)**.** Let $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_k)$ be an atom and $\alpha = \alpha_1 \cdots \alpha_k$ a string of the alphabet $\{\mathtt{b}, \mathtt{f}\}$. Then $\mathtt{p}^\alpha(\mathtt{t}_1, \ldots, \mathtt{t}_k)$ is the adorned version of $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_k)$ in which $\mathtt{t}_i$ is considered either *bound* if α_i is $\mathtt{b}$, or *free* if α_i is $\mathtt{f}$.

Adorned atoms are then associated with magic atoms, which will be used for identifying those atoms that are relevant for answering the input query.

Definition 2.2 (Magic atom)**.** For an adorned atom $\mathtt{p}^\alpha(\bar{\mathtt{t}})$, let $\mathtt{magic}(\mathtt{p}^\alpha(\bar{\mathtt{t}}))$ be its *magic version* defined as the atom $\mathtt{magic\_p}^\alpha(\bar{\mathtt{t}}')$, where $\bar{\mathtt{t}}'$ is obtained from $\bar{\mathtt{t}}$ by eliminating all arguments corresponding to an $\mathtt{f}$ label in α, and where $\mathtt{magic\_p}^\alpha$ is a new predicate symbol (for simplicity denoted by attaching the prefix "$\mathtt{magic\_}$" to the predicate symbol $\mathtt{p}^\alpha$).

Finally, we formally define SIPS for (disjunctive) ASP rules.

Definition 2.3 (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 $Atoms(r)$, such that:
  (a) $\mathtt{p}(\bar{\mathtt{t}}) \prec_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})} \mathtt{q}(\bar{\mathtt{s}})$, for all atoms $\mathtt{q}(\bar{\mathtt{s}}) \in Atoms(r)$ 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 $\mathtt{b}(\bar{\mathtt{z}}) \in Atoms(r)$, $\mathtt{q}(\bar{\mathtt{s}}) \prec_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})} \mathtt{b}(\bar{\mathtt{z}})$ does not hold; and,

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

The Dynamic Magic Set method 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 $\mathrm{DMS}(\mathcal{Q}, \mathcal{P})$ 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.

The computation starts by initializing S and $modifiedRules_{\mathcal{Q},\mathcal{P}}$ to the empty set (step *1*). Then, the function ***BuildQuerySeed***$(\mathcal{Q}, \mathcal{P}, S)$ is used for storing the query seed $\mathtt{magic}(\mathtt{g}^\alpha(\bar{\mathtt{t}}))$ in $magicRules_{\mathcal{Q},\mathcal{P}}$, where α is a string having a $\mathtt{b}$ in position i if $\mathtt{t}_i$ is a constant, or an $\mathtt{f}$ if $\mathtt{t}_i$ is a variable. In addition, ***BuildQuerySeed***$(\mathcal{Q}, \mathcal{P}, S)$ adds the adorned predicate $\mathtt{magic\_g}^\alpha$ into the set 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 (step *3*), and its binding is propagated in each rule of the form

$$r: \ \mathtt{p}(\bar{\mathtt{t}}) \ \mathtt{v} \ \mathtt{p}_1(\bar{\mathtt{t}}_1) \ \mathtt{v} \cdots \mathtt{v} \ \mathtt{p}_n(\bar{\mathtt{t}}_n) \coloneq \mathtt{q}_1(\bar{\mathtt{s}}_1), \ldots, \ \mathtt{q}_j(\bar{\mathtt{s}}_j), \ \mathtt{not} \ \mathtt{q}_{j+1}(\bar{\mathtt{s}}_{j+1}), \ldots, \ \mathtt{not} \ \mathtt{q}_m(\bar{\mathtt{s}}_m).$$

(with $n \geq 0$) 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 $\mathtt{p}$ occur; steps *4–8*).

Input: A stratified program $\mathcal{P}$, and a query $\mathcal{Q} = \mathtt{g}(\bar{\mathtt{t}})$?
Output: The optimized program 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. $S := \emptyset$; $modifiedRules_{\mathcal{Q},\mathcal{P}} := \emptyset$; $magicRules_{\mathcal{Q},\mathcal{P}} := \{\textbf{\textit{BuildQuerySeeds}}(\mathcal{Q}, \mathcal{P}, S)\}$;
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. 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.

Figure 1: Dynamic Magic Set algorithm (DMS) for stratified programs.

Adornment. The function $\textbf{\textit{Adorn}}(r, \mathtt{p}^\alpha, S)$ implements the adornment of the rule r w.r.t. an (adorned) head atom $\mathtt{p}^\alpha(\bar{\mathtt{t}})$ according to a fixed SIPS $(\prec_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})}, f_r^{\mathtt{p}^\alpha(\bar{\mathtt{t}})})$ (step *5*). In particular, a variable $\mathtt{X}$ of an IDB atom[6] $\mathtt{q}(\bar{\mathtt{s}})$ 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.

Therefore, $\textbf{\textit{Adorn}}(r, \mathtt{p}^\alpha, S)$ produces an adorned disjunctive rule r^a from an adorned predicate $\mathtt{p}^\alpha$ and a suitable unadorned rule r (according to the bindings defined in (1) and (2) above), by inserting all newly adorned predicates in S. Hence, the rule r^a is of the form

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

Generation. The adorned rules are then used to generate *magic rules* defining *magic predicates*, which represent the atoms relevant for answering the input query (step *6*). The bodies of magic rules contain the atoms required for binding the arguments of some atom, following the adopted SIPS. More specifically, if $\mathtt{q}_\mathtt{i}^{\beta_\mathtt{i}}(\bar{\mathtt{s}}_\mathtt{i})$ is an adorned atom (i.e., β_i is not the empty string) in an adorned rule r^a having $\mathtt{p}^\alpha(\bar{\mathtt{t}})$ in head, $\textbf{\textit{Generate}}(r^a)$ produces a magic rule r^* such that (i) $H(r^*) = \{\mathtt{magic}(\mathtt{q}_\mathtt{i}^{\beta_\mathtt{i}}(\bar{\mathtt{s}}_\mathtt{i}))\}$ and (ii) $B(r^*)$ is the union of $\{\mathtt{magic}(\mathtt{p}^\alpha(\bar{\mathtt{t}}))\}$ and the set of all the atoms $\mathtt{q}_\mathtt{j}^{\beta_\mathtt{j}}(\bar{\mathtt{s}}_\mathtt{j}) \in Atoms(r)$ such that $\mathtt{q}_\mathtt{j}(\bar{\mathtt{s}}_\mathtt{j}) \prec_r^\alpha \mathtt{q}_\mathtt{i}(\bar{\mathtt{s}}_\mathtt{i})$.

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* (step *7*).

A modified rule r' is obtained from an adorned rule r^a by adding to its body a magic atom $\mathtt{magic}(\mathtt{p}^\alpha(\bar{\mathtt{t}}))$ for each atom $\mathtt{p}^\alpha(\bar{\mathtt{t}}) \in H(r^a)$ and by stripping off the adornments of the original atoms. Hence, the function $\textbf{\textit{Modify}}(r^a)$ constructs a rule r' of the form

$$r'\colon\ \mathtt{p}(\bar{\mathtt{t}})\ \mathtt{v}\ \mathtt{p}_1(\bar{\mathtt{t}}_1)\ \mathtt{v}\ \cdots\ \mathtt{v}\ \mathtt{p}_\mathtt{n}(\bar{\mathtt{t}}_\mathtt{n}) \mathtt{:-}\ \mathtt{magic}(\mathtt{p}^\alpha(\bar{\mathtt{t}})),\ \mathtt{magic}(\mathtt{p}_1^{\alpha_1}(\bar{\mathtt{t}}_1)),\ \ldots,\ \mathtt{magic}(\mathtt{p}_\mathtt{n}^{\alpha_\mathtt{n}}(\bar{\mathtt{t}}_\mathtt{n})),\ \mathtt{q}_1(\bar{\mathtt{s}}_1),\ \ldots, \mathtt{q}_\mathtt{j}(\bar{\mathtt{s}}_\mathtt{j}),\ \mathtt{not}\ \mathtt{q}_{\mathtt{j}+1}(\bar{\mathtt{s}}_{\mathtt{j}+1}),\ \ldots,\ \mathtt{not}\ \mathtt{q}_\mathtt{m}(\bar{\mathtt{s}}_\mathtt{m}).$$

Finally, after all the adorned predicates have been processed, the algorithm outputs the program DMS$(\mathcal{Q}, \mathcal{P})$ (steps *10–11*).

[6]EDB atoms are always adorned with the empty string.

2.3. Query Equivalence Results

We conclude the presentation of the DMS algorithm by sketching the correctness proof presented in [Alv09], to which we refer for the details. Throughout this section, we use the well established notion of unfounded set for disjunctive programs with negation defined in [Leo97]. 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 2.4 (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 $r_g \in \mathit{Ground}(\mathcal{P})$ 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 [Leo97].

Theorem 2.5. *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 2.6 (Killed atoms)**.** Given a model M for $\mathrm{DMS}(\mathcal{Q},\mathcal{P})$, and a model $N \subseteq M$ of $\mathit{Ground}(\mathrm{DMS}(\mathcal{Q},\mathcal{P}))^M$, the set $\mathtt{killed}^M_{\mathcal{Q},\mathcal{P}}(N)$ of the *killed atoms* w.r.t. M and N is defined as:

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

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

The soundness of the algorithm for stratified programs is proved by the next lemma.

Lemma 2.8. *Let $\mathcal{Q}$ be a query over a stratified program $\mathcal{P}$. Then, for each answer set M' of $\mathrm{DMS}(\mathcal{Q},\mathcal{P})$, 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 stratified, 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 (by following [Alv09]). 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 2.7. ∎

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

Definition 2.9 (Magic variant)**.** Let I be an interpretation for $\mathcal{P}$. We define an interpretation $\mathtt{var}^{\infty}_{\mathcal{Q},\mathcal{P}}(I)$ for 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:

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

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 DMS$(\mathcal{Q},\mathcal{P})$ preserving the truth/falsity of $\mathcal{Q}\vartheta$, for every substitution ϑ.

Lemma 2.10. *For each answer set M of $\mathcal{P}$, there is an answer set M' of* 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 DMS$(\mathcal{Q},\mathcal{P})$. 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 2.7. ∎

From the above lemma, together with Lemma 2.8, the correctness of the Magic Set method with respect to query answering directly follows.

Theorem 2.11. *Let $\mathcal{Q}$ be a query over a stratified program $\mathcal{P}$. Then both* DMS$(\mathcal{Q},\mathcal{P}) \equiv^{b}_{\mathcal{Q}} \mathcal{P}$ *and* DMS$(\mathcal{Q},\mathcal{P}) \equiv^{c}_{\mathcal{Q}} \mathcal{P}$ *hold.*

3. Implementation

DMS has been implemented and integrated into the core of the DLV [Leo04] system. 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 behaviour 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 answer set (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`.

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

4. Experimental Results

In order to evaluate the impact of the proposed method, we have compared DMS both with the traditional DLV evaluation without Magic Sets and with the SMS rewriting. We considered several benchmarks, including 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. Below, we briefly discuss Conformant Plan Checking, a representative benchmark of our experimentation.

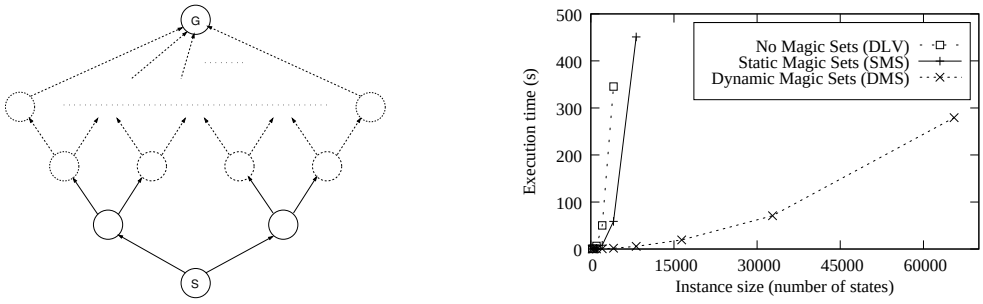

Figure 2: *Conformant Plan Checking*: instance structure and average execution time.

Definition 4.1 (Conformant Plan Checking [Gol96])**.** A state transition diagram and a plan are given. A plan is a sequence of nondeterministic actions, and is conformant if all its possible executions lead from an initial state S to a goal state G.

In our experiments, the shape of transition diagrams is as shown in Figure 2. A successor state is guessed for each state in the input diagram, so that the plan is conformant if G is reachable from S in each answer set of the program

```
trans(X,Y) v trans(X,Z) :- ptrans(X,Y,Z).
reach(X,Y) :- trans(X,Y).
reach(X,Y) :- reach(X,Z), trans(Z,Y).
```

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.

As shown in Figure 2, DMS has an exponential speed-up over both DLV and SMS. In this case, the exponential computational gain of DMS over DLV and SMS is due to the dynamic optimization of the answer set search phase resulting from our magic sets definition. Indeed, DMS include nondeterministic relevance 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.

5. 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 our work, we have extended the technique to (disjunctive) ASP. The main novelty of the proposed Magic Set method is the dynamic optimization of the answer set search. Indeed, with DMS, ASP computations can exploit the information provided by magic set predicates also during the nondeterministic answer set search, allowing for potentially exponential performance gains with respect to unoptimized evaluations.

We have established the correctness of DMS for stratified ASP programs 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. an answer set are either true or form an unfounded set.

DMS has been implemented in the DLV system. Experiments 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.

Our research has been focused on ASP with stratified negation, because the concept of "relevance" can be extended quite easily for this class of programs. Conversely, if recursion over negation is allowed, an inconsistency may arise in some part of the program apparently not related to the query. A first step forward in extending DMS to programs with unstratified negation has been done in [Alv10], in which the technique presented in this paper has been proved to be correct for super-consistent ASP, a large class of programs including odd-cycle-free programs (that is, programs in which no cycle of dependencies involves an odd number of negative dependencies). Analysing the possibility to extend DMS to a larger class of unstratified ASP programs is a challenge we intend to address in the future.

Acknowledgement

The author wishes to thank Wolfgang Faber for his care in checking this work and for the fruitful discussions without which the results herein summarized could not be achieved.

References

[Alv09] Mario Alviano, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic Sets for Disjunctive Datalog Programs. Tech. Rep. 09/2009, Department of Mathematics, University of Calabria, Italy, 2009. `http://www.wfaber.com/research/papers/TRMAT092009.pdf`.

[Alv10] Mario Alviano and Wolfgang Faber. Dynamic Magic Sets for Super-Consistent Answer Set Programs. In *3rd Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP10)*. 2010. To appear.

[Bar03] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[Bee91] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. 10(1–4):255–259, 1991.

[Eit94] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Adding Disjunction to Datalog. In *Proceedings of the Thirteenth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-94)*, pp. 267–278. ACM Press, 1994.

[Gol96] Robert P. Goldman and Mark S. Boddy. Expressive Planning and Explicit Knowledge. In *Proceedings AIPS-96*, pp. 110–117. AAAI Press, 1996.

[Gre03] Sergio Greco. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):368–385, 2003.

[Leo97] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. 135(2):69–112, 1997.

[Leo04] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. 2004. To appear. Available via `http://www.arxiv.org/ps/cs.AI/0211004`.