TLP **10** (4–6): 497–512, 2010.
doi:10.1017/S1471068410000244

*Disjunctive ASP with functions: Decidable queries and effective computation**

MARIO ALVIANO, WOLFGANG FABER and NICOLA LEONE

Department of Mathematics, University of Calabria
87036 Rende (CS), Italy
(*e-mail:* {alviano,faber,leone}@mat.unical.it)

submitted 8 February 2010; revised 1 May 2010; accepted 16 May 2010

Abstract

Querying over disjunctive ASP with functions is a highly undecidable task in general. In this paper we focus on disjunctive logic programs with stratified negation and functions under the stable model semantics (ASP^{fs}). We show that query answering in this setting is decidable, if the query is finitely recursive (ASP^{fs}_{fr}). Our proof yields also an effective method for query evaluation. It is done by extending the magic set technique to ASP^{fs}_{fr}. We show that the magic-set rewritten program is query equivalent to the original one (under both brave and cautious reasoning). Moreover, we prove that the rewritten program is also finitely ground, implying that it is decidable. Importantly, finitely ground programs are evaluable using existing ASP solvers, making the class of ASP^{fs}_{fr} queries usable in practice.

KEYWORDS: answer set programming, decidability, magic sets, disjunctive logic programs

1 Introduction

Answer Set Programming (ASP), Logic Programming (LP) under the answer set or stable model semantics, has established itself as a convenient and effective method for declarative knowledge representation and reasoning over the course of the last 20 years (Baral 2003; Gelfond and Lifschitz 1991). A major reason for the success of ASP has been the availability of implemented and efficient systems, which allowed for the paradigm to be usable in practice.

This work is about ASP with stratified negation and functions under the stable model semantics (ASP^{fs}). Dealing with the introduction of function symbols in the language of ASP has been the topic of several works in the literature (Bonatti 2002; Bonatti 2004; Baselice *et al.* 2009; Calimeri *et al.* 2009; Syrjänen 2001; Gebser *et al.* 2007; Calimeri *et al.* 2008a; Lierler and Lifschitz 2009; Simkus and Eiter 2007; Eiter and Simkus 2009; Lin and Wang 2008; Cabalar 2008). They have been motivated by overcoming the major limitation of ASP systems with respect to traditional LP systems, which is the possibility of representing only a finite set of individuals by means of constant symbols. Most of the approaches treat function symbols in the

* 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.

traditional logic programming way, that is by considering the Herbrand universe. A few other works treat function symbols in a way which is closer to classical logic (see, e.g., (Cabalar 2008)). The fundamental problem with admitting function symbols in ASP is that the common inference tasks become undecidable. The identification of expressive decidable classes of ASP programs with functions is therefore an important task, and has been addressed in several works (see Section 6).

Here, we follow the traditional logic programming approach, and study the rich language of finitely recursive $\text{ASP}^{\text{fs}}(\text{ASP}^{\text{fs}}_{\text{fr}})$, showing that it is still decidable. In fact, our work links two relevant classes of ASP with functions: finitely recursive and finitely ground programs. We extend a magic set method for programs with disjunctions and stratified negation to deal with functions and specialize it for finitely recursive queries. We show that the transformed program is query equivalent to the original one and that it belongs to the class of finitely ground programs. Finitely ground programs have been shown to be decidable and therefore it follows that $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries are decidable, too. Importantly, by DLV-Complex (Calimeri *et al.* 2008b) there is a system which supports query answering on finitely ground programs, so the magic set method serves also as a means for effectively evaluating $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries. We also show that $\text{ASP}^{\text{fs}}_{\text{fr}}$ programs are maximally expressive, in the sense that each computable function can be represented. In total, $\text{ASP}^{\text{fs}}_{\text{fr}}$ programs and queries are an appealing formalism, since they are decidable, a computational system exists, they provide a rich knowledge-modeling language, including disjunction and stratified negation, and they can express any computable function.

Summarizing, the main contributions of the paper are the following:

- ► We prove that $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries are decidable under both brave and cautious reasoning.
- ► We show that the restrictions which guarantee the decidability of $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries do not limit their expressiveness. Indeed, we demonstrate that any computable function can be expressed by an $\text{ASP}^{\text{fs}}_{\text{fr}}$ program.
- ► We provide an effective implementation method for $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries, making reasoning over $\text{ASP}^{\text{fs}}_{\text{fr}}$ programs feasible in practice. In particular,
  - We design a magic-set rewriting technique for $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries. The technique is based on a particular *sideways information passing strategy* (SIPS) which exploits the structure of $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries, and guarantees that the rewritten program has a specific shape.
  - We show that the magic-set rewritten program is query equivalent to the original one (under both brave and cautious reasoning).
  - We prove that the rewritten program is finitely ground, implying that it is computable (Calimeri *et al.* 2008a). Importantly, finitely ground programs are evaluable using the existing ASP solver DLV-Complex (Calimeri *et al.* 2008b), making $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries usable in practice.

2 Preliminaries

In this section, we recall the basics of ASP with function symbols, and the decidable classes of finitely ground (Calimeri *et al.* 2008a) and finitely recursive programs (Baselice *et al.* 2009).

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

2.1 ASP syntax and semantics

A *term* is either a *variable* or a *functional term*. A functional term is of the form $f(\mathtt{t}_1,\ldots,\mathtt{t}_k)$, where f is a function symbol (*functor*) of arity $k \geqslant 0$, and $\mathtt{t}_1,\ldots,\mathtt{t}_k$ are terms[1]. A functional term with arity 0 is a *constant*. If $\mathtt{p}$ is a *predicate* of arity $k \geqslant 0$, and $\mathtt{t}_1,\ldots,\mathtt{t}_k$ are terms, then $\mathtt{p}(\mathtt{t}_1,\ldots,\mathtt{t}_k)$ is an *atom*[2]. A *literal* is either an atom $\mathtt{p}(\bar{\mathtt{t}})$ (a positive literal), or an atom preceded by the *negation as failure* symbol $\mathtt{not}\ \mathtt{p}(\bar{\mathtt{t}})$ (a negative literal). A *rule* r is of the form

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

where $\mathtt{p}_1(\bar{\mathtt{t}}_1),\ \ldots,\ \mathtt{p}_n(\bar{\mathtt{t}}_n),\ \mathtt{q}_1(\bar{\mathtt{s}}_1),\ \ldots,\ \mathtt{q}_m(\bar{\mathtt{s}}_m)$ are atoms and $n \geqslant 1$, $m \geqslant j \geqslant 0$. 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 $\mathtt{q}_1(\bar{\mathtt{s}}_1),\ \ldots,\ q_j(\bar{\mathtt{s}}_j),\ \mathtt{not}\ q_{j+1}(\bar{\mathtt{s}}_{j+1}),\ \ldots,\ \mathtt{not}\ q_m(\bar{\mathtt{s}}_m)$ 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 $H(r) \cup B^+(r) \cup B^-(r)$. 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* $\mathscr{P}$ is a finite set of rules; if all the rules in it are positive (resp. normal), then $\mathscr{P}$ is a positive (resp. normal) program. In addition, $\mathscr{P}$ is function-free if each functional term appearing in $\mathscr{P}$ is a constant. 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 there is no cycle of dependencies involving a negative dependency. In this paper we focus on the class of stratified programs.

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*.[3] Given a program $\mathscr{P}$, the set of rules having some IDB predicate in head is denoted by $IDB(\mathscr{P})$, while $EDB(\mathscr{P})$ denotes the remaining rules, that is, $EDB(\mathscr{P}) = \mathscr{P} \setminus IDB(\mathscr{P})$. In addition, the set of all facts of $\mathscr{P}$ is denoted by $Facts(\mathscr{P})$.

The set of terms constructible by combining functors appearing in a program $\mathscr{P}$ is the *universe* of $\mathscr{P}$ and is denoted by $U_{\mathscr{P}}$, while the set of ground atoms constructible from predicates in $\mathscr{P}$ with elements of $U_{\mathscr{P}}$ is the *base* of $\mathscr{P}$, denoted by $B_{\mathscr{P}}$. 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 $U_{\mathscr{P}}$ such that $\mathtt{p}(\bar{\mathtt{t}}) = \mathtt{p}(\bar{\mathtt{t}}')\vartheta$ (resp. $r_g = r\vartheta$). Given a program $\mathscr{P}$, $Ground(\mathscr{P})$ denotes the set of all the instances of the rules in $\mathscr{P}$.

An *interpretation* I for a program $\mathscr{P}$ is a subset of $B_{\mathscr{P}}$. 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

[1] We also use Prolog-like square-bracketed list notation as in (Calimeri *et al.* 2008a).
[2] We use the notation $\bar{\mathtt{t}}$ for a sequence of terms, for referring to atoms as $\mathtt{p}(\bar{\mathtt{t}})$.
[3] *EDB* and *IDB* stand for Extensional Database and Intensional Database, respectively.

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

ground literal $\texttt{not p}(\bar{\texttt{t}})$ is true w.r.t. I if and only if $\texttt{p}(\bar{\texttt{t}})$ 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 $r_g \in \mathit{Ground}(\mathscr{P})$ 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 $\mathscr{P}$ if I satisfies all the rules in $\mathit{Ground}(\mathscr{P})$.

Given an interpretation I for a program $\mathscr{P}$, the reduct of $\mathscr{P}$ w.r.t. I, denoted $\mathit{Ground}(\mathscr{P})^I$, is obtained by deleting from $\mathit{Ground}(\mathscr{P})$ 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 $\mathscr{P}$ is then given by the set $\mathscr{SM}(\mathscr{P})$ of the stable models of $\mathscr{P}$, where an interpretation M is a stable model for $\mathscr{P}$ if and only if M is a subset-minimal model of $\mathit{Ground}(\mathscr{P})^M$.

Given a program $\mathscr{P}$ and a query $\mathscr{Q} = \texttt{g}(\bar{\texttt{t}})?$ (a ground atom),[4] $\mathscr{P}$ *cautiously* (resp. *bravely*) entails $\mathscr{Q}$, denoted $\mathscr{P} \models_c \mathscr{Q}$ (resp. $\mathscr{P} \models_b \mathscr{Q}$) if and only if $\texttt{g}(\bar{\texttt{t}}) \in M$ for all (resp. some) $M \in \mathscr{SM}(\mathscr{P})$. Two programs $\mathscr{P}$ and $\mathscr{P}'$ are *cautious-equivalent* (resp. *brave-equivalent*) w.r.t. a query $\mathscr{Q}$, denoted by $\mathscr{P} \equiv^c_{\mathscr{Q}} \mathscr{P}'$ (resp. $\mathscr{P} \equiv^b_{\mathscr{Q}} \mathscr{P}'$), whenever $\mathscr{P} \models_c \mathscr{Q}$ iff $\mathscr{P}' \models_c \mathscr{Q}$ (resp. $\mathscr{P} \models_b \mathscr{Q}$ iff $\mathscr{P}' \models_b \mathscr{Q}$).

2.2 *Finitely ground programs*

The class of finitely ground ($\mathscr{FG}$) programs (Calimeri *et al.* 2008a) constitutes a natural formalization of programs which can be finitely evaluated bottom-up. We recall the key concepts, and refer to (Calimeri *et al.* 2008a) for details and examples.

The dependency graph $\mathscr{G}(\mathscr{P})$ of a program $\mathscr{P}$ is a directed graph having a node for each IDB predicate of $\mathscr{P}$, and an edge $\texttt{q} \rightarrow \texttt{p}$ if there is a rule $r \in \mathscr{P}$ such that p occurs in $H(r)$ and q occurs in $B^+(r)$.[5] A *component* C of $\mathscr{P}$ is then a set of predicates which are strongly connected in $\mathscr{G}(\mathscr{P})$.

The component graph of $\mathscr{P}$, denoted $\mathscr{G}^{\mathscr{C}}(\mathscr{P})$, is a labelled directed graph having (*i*) a node for each component of $\mathscr{G}(\mathscr{P})$, (*ii*) an edge $C' \rightarrow^+ C$ if there is a rule $r \in \mathscr{P}$ such that a predicate $\texttt{p} \in C$ occurs in $H(r)$ and a predicate $\texttt{q} \in C'$ occurs in $B^+(r)$, and (*iii*) an edge $C' \rightarrow^- C$ if (a) $C' \rightarrow^+ C$ is not an edge of $\mathscr{G}^{\mathscr{C}}(\mathscr{P})$, and (b) there is a rule $r \in \mathscr{P}$ such that a predicate $\texttt{p} \in C$ occurs in $H(r)$ and a predicate $\texttt{q} \in C'$ occurs in $B^-(r)$. A path in a component graph $\mathscr{G}^{\mathscr{C}}(\mathscr{P})$ is *weak* if at least one of its edges is labelled with "−", otherwise it is *strong*.

A component ordering $\gamma = \langle C_1, \ldots, C_n \rangle$ is a total ordering of all the components of $\mathscr{P}$ such that, for any C_i, C_j with $i < j$, both (*a*) there is no strong path from C_j to C_i in $\mathscr{G}^{\mathscr{C}}(\mathscr{P})$, and (*b*) if there is a weak path from C_j to C_i, then there must be a weak path also from C_i to C_j. A *module* $P(C_i)$ of a program $\mathscr{P}$ is the set of rules defining predicates in C_i, excluding those that define also some other predicate belonging to a lower component in γ, that is, a component C_j with $j < i$.

Given a rule r and a set A of ground atoms, an instance r_g of r is an *A-restricted* instance of r if $B^+(r_g) \subseteq A$. The set of all A-restricted instances of all

[4] More complex queries can still be expressed using appropriate rules. We assume that each functor appearing in $\mathscr{Q}$ also appears in $\mathscr{P}$; if this is not the case, then we can add to $\mathscr{P}$ a fact $\texttt{p}(\bar{\texttt{t}})$ (where p is a predicate that occurs neither in $\mathscr{P}$ nor $\mathscr{Q}$) and $\bar{\texttt{t}}$ are the arguments of $\mathscr{Q}$.

[5] In literature, $\mathscr{G}(\mathscr{P})$ is also referred as *positive dependencies graph.*

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

the rules of a program $\mathscr{P}$ is denoted by $Inst_{\mathscr{P}}(A)$. Note that, for any $A \subseteq B_{\mathscr{P}}$, $Inst_{\mathscr{P}}(A) \subseteq Ground(\mathscr{P})$. Intuitively, this identifies those ground instances that may be *supported* by a given set A.

Let $\mathscr{P}$ be a program, C_i a component in a component ordering $\langle C_1, \ldots, C_n \rangle$, T a set of ground rules to be simplified w.r.t. another set R of ground rules. Then the *simplification* $Simpl(T, R)$ of T w.r.t. R is obtained from T by: (*a*) *deleting* each rule r_g such that $H(r_g) \cup B^-(r_g)$ contains some atom $\mathtt{p}(\bar{\mathtt{t}}) \in Facts(R)$; (*b*) *eliminating* from each remaining rule r_g the atoms in $B^+(r_g) \cap Facts(R)$, and each atom $\mathtt{p}(\bar{\mathtt{t}}) \in B^-(r_g)$ such that $\mathtt{p} \in C_j$, with $j < i$, and there is no rule in R with $\mathtt{p}(\bar{\mathtt{t}})$ in its head. Assuming that R contains all ground instances obtained from the modules preceding C_i, $Simpl(T, R)$ deletes from T the rules whose head is certainly already true w.r.t. R or whose body is certainly false w.r.t. R, and simplifies the remaining rules by removing from the bodies all literals true w.r.t. R. We define now the operator Φ, combining $Inst$ and $Simpl$.

Let $\mathscr{P}$ be a program, C_i a component in a component ordering $\langle C_1, \ldots, C_n \rangle$, R and S two sets of ground rules. Then $\Phi_{P(C_i),R}(S) = Simpl(Inst_{P(C_i)}(A), R)$, where A is the set of atoms belonging to the head of some rule in $R \cup S$. The operator Φ always admit a least fixpoint $\Phi^{\infty}_{P(C_i),R}(\emptyset)$. We can then define the *intelligent instantiation* $\mathscr{P}^{\gamma}$ of a program $\mathscr{P}$ for a component ordering $\gamma = \langle C_1, \ldots, C_n \rangle$ as the last element $\mathscr{P}^{\gamma}_n$ of the sequence $\mathscr{P}^{\gamma}_0 = EDB(\mathscr{P})$, $\mathscr{P}^{\gamma}_i = \mathscr{P}^{\gamma}_{i-1} \cup \Phi^{\infty}_{P(C_i),\mathscr{P}^{\gamma}_{i-1}}(\emptyset)$. $\mathscr{P}$ is *finitely ground* ($\mathscr{FG}$) if $\mathscr{P}^{\gamma}$ is finite for every component ordering γ for $\mathscr{P}$. The main result for this class of programs is that reasoning is effectively computable.

Theorem 2.1

Cautious and brave reasoning over $\mathscr{FG}$ programs are decidable.

2.3 Finitely recursive queries

We next provide the definition of finitely recursive queries (Calimeri *et al.* 2009) and programs (Baselice *et al.* 2009).

Let $\mathscr{P}$ be a program and $\mathscr{Q}$ a query. The relevant atoms for $\mathscr{Q}$ are: (*a*) $\mathscr{Q}$ itself, and (*b*) each atom in $Atoms(r_g)$, where $r_g \in Ground(\mathscr{P})$ is such that some atom in $H(r_g)$ is relevant for $\mathscr{Q}$. Then (*i*) $\mathscr{Q}$ is *finitely recursive* on $\mathscr{P}$ if only a finite number of ground atoms is relevant for $\mathscr{Q}$, and (*ii*) $\mathscr{P}$ is *finitely recursive* if every query is finitely recursive on $\mathscr{P}$.

Example 2.2

Consider the query $\mathtt{greaterThan(s(s(0)),0)?}$ for the following program:

$$
\begin{aligned}
r_1 &: \quad \mathtt{lessThan(X, s(X)).} \\
r_2 &: \quad \mathtt{lessThan(X, s(Y))} \text{ :– } \mathtt{lessThan(X, Y).} \\
r_3 &: \quad \mathtt{greaterThan(s(X), Y)} \text{ :– } \mathtt{not\ lessThan(X, Y).}
\end{aligned}
$$

The program cautiously and bravely entails the query. The query is clearly finitely recursive; also the program is finitely recursive. □

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

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. In this section we first recall the magic set technique for disjunctive programs with stratified negation without function symbols, as presented in (Alviano *et al.* 2009), we then lift the technique to $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ queries, and formally prove its correctness.

3.1 Magic sets for function-free programs

The method of (Alviano *et al.* 2009)[6] 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 $\mathrm{g}(\bar{\mathrm{t}}') \in H(r)$ (where $\mathrm{g}(\bar{\mathrm{t}}')\vartheta = \mathcal{Q}$ for some substitution ϑ) are considered in a first step. Then the atoms in $\mathit{Atoms}(r\vartheta)$ 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 arguments is considered to be either *bound* or *free*.

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 originated and propagate (Beeri and Ramakrishnan 1991; Greco 2003). 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. Thus, 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 function-free program $\mathscr{P}$, a query $\mathcal{Q}$, and the rewritten program $\mathscr{P}'$, $\mathscr{P}$ and $\mathscr{P}'$ are equivalent w.r.t. $\mathcal{Q}$, i.e., $\mathscr{P} \equiv^{b}_{\mathcal{Q}} \mathscr{P}'$ and $\mathscr{P} \equiv^{c}_{\mathcal{Q}} \mathscr{P}'$ hold (Alviano *et al.* 2009).

[6] For a detailed description of the standard technique we refer to (Ullman 1989).

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

Input: A program $\mathscr{P}$, and a query $\mathscr{Q} = \mathtt{g}(\bar{\mathtt{t}})$?
Output: The optimized program $\mathtt{DMS}(\mathscr{Q}, \mathscr{P})$.
var S, D: **set** of predicates; $modifiedRules_{\mathscr{Q},\mathscr{P}}$, $magicRules_{\mathscr{Q},\mathscr{P}}$: **set** of rules;
begin
1. $D := \emptyset$; $modifiedRules_{\mathscr{Q},\mathscr{P}} := \emptyset$; $magicRules_{\mathscr{Q},\mathscr{P}} := \{\mathtt{magic\_g}(\bar{\mathtt{t}}).\}$; $S := \{\mathtt{g}\}$;
2. **while** $S \neq \emptyset$ **do**
3. take an element p from S; remove p from S; add p to D;
4. **for each** rule $r \in \mathscr{P}$ and **for each** atom $\mathtt{p}(\bar{\mathtt{t}}) \in H(r)$ **do**
5. $r' := r$;
6. **for each** atom $\mathtt{q}(\bar{\mathtt{s}}) \in H(r)$ **do** add $\mathtt{magic\_q}(\bar{\mathtt{s}})$ to $B(r')$; **end for**
7. add r' to $modifiedRules_{\mathscr{Q},\mathscr{P}}$;
8. **for each** atom $\mathtt{q}(\bar{\mathtt{s}}) \in Atoms(r) \setminus \{\mathtt{p}(\bar{\mathtt{t}})\}$ such that q is an IDB predicate **do**
9. add $\mathtt{magic\_q}(\bar{\mathtt{s}}) \text{ :- } \mathtt{magic\_p}(\bar{\mathtt{t}})$. to $magicRules_{\mathscr{Q},\mathscr{P}}$; add q to S **if** $\mathtt{q} \notin D$;
10. **end for**
11. **end for**
12. **end while**
13. $\mathtt{DMS}(\mathscr{Q}, \mathscr{P}) := magicRules_{\mathscr{Q},\mathscr{P}} \cup modifiedRules_{\mathscr{Q},\mathscr{P}} \cup EDB(\mathscr{P})$;
14. **return** $\mathtt{DMS}(\mathscr{Q}, \mathscr{P})$;
end.

Fig. 1. Magic Set algorithm (DMS) for $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries.

3.2 A rewriting algorithm for $\text{ASP}^{\text{fs}}_{\text{fr}}$ *programs*

Our rewriting algorithm exploits the peculiarities of $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries, and guarantees that the rewritten program is query equivalent, that it has a particular structure and that it is bottom-up computable. In particular, for a finitely recursive query $\mathscr{Q}$ over an ASP^{fs} program $\mathscr{P}$, the Magic-Set technique can be simplified due to the following observations:

- For each (sub)query $\mathtt{g}(\bar{\mathtt{t}})$ and each rule r having an atom $\mathtt{g}(\bar{\mathtt{t}}') \in H(r)$, all the variables appearing in r appear also in $\mathtt{g}(\bar{\mathtt{t}}')$. Indeed, if this is not the case, then an infinite number of ground atoms would be relevant for $\mathscr{Q}$ (the query would not be finitely recursive)[7]. Therefore, each adorned predicate generated in the **Adornment** phase has all arguments bound.
- Since all variables of a processed rule are bound by the (sub)query, the body of a magic rule produced in the **Generation** phase consists only of the magic version of the (sub)query (by properly limiting the adopted SIPS).

We assume the original program has no predicate symbol that begins with the string "magic_". In the following we will then use magic_p for denoting the magic predicate associated with the predicate p. So the magic atom associated with $\mathtt{p}(\bar{\mathtt{t}})$ will be $\mathtt{magic\_p}(\bar{\mathtt{t}})$, in which, by previous considerations, each argument is bound.

The algorithm DMS implementing the Magic-Set technique for $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries is reported in Figure 1. Given a program $\mathscr{P}$ and a query $\mathscr{Q}$, the algorithm outputs a rewritten and optimized program $\mathtt{DMS}(\mathscr{Q}, \mathscr{P})$, consisting of a set of *modified* and *magic* rules, stored by means of the sets $modifiedRules_{\mathscr{Q},\mathscr{P}}$ and $magicRules_{\mathscr{Q},\mathscr{P}}$, respectively (together with the original EDB). The algorithm exploits a set S for storing all the predicates to be processed, and a set D for storing the predicates already done.

The computation starts by initializing D and $modifiedRules_{\mathscr{Q},\mathscr{P}}$ to the empty set (step *1*). Then the magic seed $\mathtt{magic\_g}(\bar{\mathtt{t}})$. (a fact) is stored in $magicRules_{\mathscr{Q},\mathscr{P}}$ and the

[7] We assume the general case where there is some functor with arity greater than 0.

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

predicate g is inserted in the set S (step *1*). The core of the algorithm (steps *2–12*) is repeated until the set S is empty, i.e., until there is no further predicate to be propagated. In particular, a predicate p is moved from S to D (step *3*), and each rule $r \in \mathscr{P}$ having an atom $\mathtt{p}(\bar{\mathtt{t}})$ in the head is considered (note that one rule r is processed as often as p occurs in its head; steps *4–11*). A modified rule r' is subsequently obtained from r by adding an atom $\mathtt{magic\_q}(\bar{\mathtt{s}})$ (for each atom $\mathtt{q}(\bar{\mathtt{s}})$ in the head of r) to its body (steps *5–7*). In addition, for each atom $\mathtt{q}(\bar{\mathtt{s}})$ in $Atoms(r) \setminus \{\mathtt{p}(\bar{\mathtt{t}})\}$ such that q is an IDB predicate (steps *8–10*), a magic rule $\mathtt{magic\_q}(\bar{\mathtt{s}}) \text{ :- } \mathtt{magic\_p}(\bar{\mathtt{t}})$. is generated (step *9*), and the predicate q is added to the set S if not already processed (i.e., if $\mathtt{q} \notin D$; step *9*). Note that the magic rule $\mathtt{magic\_q}(\bar{\mathtt{s}}) \text{ :- } \mathtt{magic\_p}(\bar{\mathtt{t}})$. is added also if $\mathtt{q}(\bar{\mathtt{s}})$ occurs in the head or in the negative body, since bindings are propagated in a uniform way to all IDB atoms.

Example 3.1
The result of the application of the DMS algorithm to the program and query in Example 2.2 is:

r'_1 : lessThan(X, s(X)) :- magic_lessThan(X, s(X)).
r'_2 : lessThan(X, s(Y)) :- magic_lessThan(X, s(Y)), lessThan(X, Y).
r'_3 : greaterThan(s(X), Y) :- magic_greaterThan(s(X), Y), not lessThan(X, Y).
r^*_2 : magic_lessThan(X, Y) :- magic_lessThan(X, s(Y)).
r^*_3 : magic_lessThan(X, Y) :- magic_greaterThan(s(X), Y).
$r_{\mathscr{Q}}$: magic_greaterThan(s(s(0)), 0). □

3.3 Query equivalence result

We conclude the presentation of the DMS algorithm by formally proving its correctness. This section essentially follows (Alviano *et al.* 2009), to which we refer for the details, while here we highlight the necessary considerations for generalizing the results of (Alviano *et al.* 2009) to $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ queries, exploiting the considerations described in Section 3.2. Throughout this section, we use the well established notion of unfounded set for disjunctive programs with negation defined in (Leone *et al.* 1997). 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 3.2 (*Unfounded sets*)
Let I be an interpretation for a program $\mathscr{P}$, and $X \subseteq B_{\mathscr{P}}$ be a set of ground atoms. Then X is an *unfounded set* for $\mathscr{P}$ w.r.t. I if and only if for each ground rule $r_g \in Ground(\mathscr{P})$ 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 (Leone *et al.* 1997).

Theorem 3.3
Let I be an interpretation for a program $\mathscr{P}$. Then, for any stable model $M \supseteq I$ of $\mathscr{P}$, and for each unfounded set X of $\mathscr{P}$ w.r.t. I, $M \cap X = \emptyset$ holds.

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

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

$$\{\, \mathtt{p}(\bar{\mathtt{t}}) \in B_{\mathscr{P}} \setminus N \mid \text{ either } \mathtt{p} \text{ is an EDB predicate, or } \mathtt{magic\_p}(\bar{\mathtt{t}}) \in N \,\}.$$

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 stable model for $\mathscr{P}$ containing $M \cap B_{\mathscr{P}}$.

Proposition 3.5
Let M be a model for $\mathtt{DMS}(\mathcal{Q},\mathscr{P})$, and $N \subseteq M$ a model of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathscr{P}))^M$. Then $\mathtt{killed}^M_{\mathcal{Q},\mathscr{P}}(N)$ is an unfounded set for $\mathscr{P}$ w.r.t. $M \cap B_{\mathscr{P}}$.

We can now prove the soundness of the algorithm.

Lemma 3.6
Let $\mathcal{Q}$ be an $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ query over $\mathscr{P}$. Then, for each stable model M' of $\mathtt{DMS}(\mathcal{Q},\mathscr{P})$, there is a stable model M of $\mathscr{P}$ such that $\mathcal{Q} \in M$ if and only if $\mathcal{Q} \in M'$.

Proof
We can show that there is $M \in \mathscr{SM}(\mathscr{P})$ such that $M \supseteq M' \cap B_{\mathscr{P}}$. Since $\mathcal{Q}$ belongs either to M' or to $\mathtt{killed}^{M'}_{\mathcal{Q},\mathscr{P}}(M')$, the claim follows by Proposition 3.5. □

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

Definition 3.7 (*Magic variant*)
Let I be an interpretation for an $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ query $\mathcal{Q}$ over $\mathscr{P}$. We define an interpretation $\mathtt{variant}_{\mathcal{Q},\mathscr{P}}(I)$ for $\mathtt{DMS}(\mathcal{Q},\mathscr{P})$, called the magic variant of I w.r.t. $\mathcal{Q}$ and $\mathscr{P}$, as follows:

$$\mathtt{variant}_{\mathcal{Q},\mathscr{P}}(I) = \mathit{EDB}(\mathscr{P}) \cup M^* \cup \{\mathtt{p}(\bar{\mathtt{t}}) \in I \mid \mathtt{magic\_p}(\bar{\mathtt{t}}) \in M^*\},$$

where M^* is the unique stable model of $\mathit{magicRules}_{\mathcal{Q},\mathscr{P}}$.

In this definition, we exploit the fact that $\mathit{magicRules}_{\mathcal{Q},\mathscr{P}}$ has a unique and finite stable model for $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ queries (see Lemma 4.2 for a detailed proof). By definition, for a magic variant $\mathtt{variant}_{\mathcal{Q},\mathscr{P}}(I)$ of an interpretation I for $\mathscr{P}$, $\mathtt{variant}_{\mathcal{Q},\mathscr{P}}(I) \cap B_{\mathscr{P}} \subseteq I$ holds. More interesting, the magic variant of a stable model for $\mathscr{P}$ is in turn a stable model for $\mathtt{DMS}(\mathcal{Q},\mathscr{P})$ preserving truth/falsity of $\mathcal{Q}$. The following formalizes the intuition above.

Lemma 3.8
If M is a stable model of an $\mathrm{ASP}^{\mathrm{fs}}$ program $\mathscr{P}$ with a finitely recursive query $\mathcal{Q}$, then $M' = \mathtt{variant}_{\mathcal{Q},\mathscr{P}}(M)$ is a stable model of $\mathtt{DMS}(\mathcal{Q},\mathscr{P})$ and $\mathcal{Q} \in M'$ if and only if $\mathcal{Q} \in M$.

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

Proof
Consider a modified rule $r'_g \in \mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))$ having $B^+(r'_g) \subseteq M'$ and $B^-(r'_g) \cap M' = \emptyset$:

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

We can show that

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

belongs to $\mathit{Ground}(\mathcal{P})$. Since $B^+(r'_g) \subseteq M'$ and $B^-(r'_g) \cap M' = \emptyset$, we have $B^+(r_g) \subseteq M$, $B^-(r_g) \cap M = \emptyset$, and $H(r'_g) \cap M' = H(r_g) \cap M$. Thus, $H(r'_g) \cap M' = H(r_g) \cap M \neq \emptyset$ because M is a model of $\mathcal{P}$. Moreover, if there is a model $N' \subset M'$ of $\mathit{Ground}(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))^{M'}$, then $M \setminus (M' \setminus N')$ is a model for $\mathit{Ground}(\mathcal{P})^M$, contradicting the assumption that M is a stable model of $\mathcal{P}$.

Thus, $M' = \mathtt{variant}_{\mathcal{Q},\mathcal{P}}(M)$ is a stable model of $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$. Since $\mathcal{Q}$ belongs either to M' or to $\mathtt{killed}^{M'}_{\mathcal{Q},\mathcal{P}}(M')$, the claim follows by Proposition 3.5. □

From the above lemma, together with Lemma 3.6, the correctness of the Magic Set method with respect to query answering directly follows.

Theorem 3.9
If $\mathcal{Q}$ is an $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ query over $\mathcal{P}$, 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 Decidability result

In this section, we prove that $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ queries are decidable. To this end, we link finitely recursive queries to finitely ground programs. More specifically, we show that the Magic-Set rewriting of a finitely recursive query is a finitely ground program, for which querying is known to be decidable.

We first show some properties of the rewritten program due to the particular restrictions applied to the adopted SIPS.

Lemma 4.1
If $\mathcal{Q}$ is an $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ query over $\mathcal{P}$, then $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ is stratified.

Proof
Each cycle of dependencies in $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ involving predicates of $\mathcal{P}$ is also present in $\mathcal{P}$. Indeed, each magic rule has exactly one magic atom in the head and one in the body, and each modified rule is obtained by adding only magic atoms to the body of a rule belonging to $\mathcal{P}$. Since $\mathcal{P}$ is stratified by assumption, such cycles have no negative dependencies. Any new cycle stems only from magic rules, which are positive. □

Now consider the program consisting of the magic rules produced for a finitely recursive query. We can show that this program has a unique and finite stable model, that we will denote M^*.

Lemma 4.2
Let $\mathcal{Q}$ be an $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ query over $\mathcal{P}$. Then the program $\mathit{magicRules}_{\mathcal{Q},\mathcal{P}}$ has a unique and finite stable model M^*.

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

Proof
Since $magicRules_{\mathcal{Q},\mathcal{P}}$ is positive and normal, M^* is unique. If we show that M^* contains all and only the relevant atoms for $\mathcal{Q}$, then we are done because $\mathcal{Q}$ is finitely recursive on $\mathcal{P}$. To this end, note that the only fact in $magicRules_{\mathcal{Q},\mathcal{P}}$ is the query seed $\mathtt{magic\_g}(\bar{\mathtt{t}})$., and each magic rule $\mathtt{magic\_q}(\bar{\mathtt{s}})\vartheta$:– $\mathtt{magic\_p}(\bar{\mathtt{t}})\vartheta$. in $Ground(\mathtt{DMS}(\mathcal{Q},\mathcal{P}))$ (ϑ a substitution) is such that $\mathtt{q}(\bar{\mathtt{s}})\vartheta$ is relevant for $\mathtt{p}(\bar{\mathtt{t}})\vartheta$. Indeed, $\mathtt{magic\_q}(\bar{\mathtt{s}})$:– $\mathtt{magic\_p}(\bar{\mathtt{t}})$. has been produced during the *Generation* phase involving a rule $r \in \mathcal{P}$ with $\mathtt{p}(\bar{\mathtt{t}}) \in H(r)$ and $\mathtt{q}(\bar{\mathtt{s}}) \in Atoms(r) \setminus \{\mathtt{p}(\bar{\mathtt{t}})\}$; since each variable in r appears also in $\mathtt{p}(\bar{\mathtt{t}})$, $r\vartheta \in Ground(\mathcal{P})$ is such that $\mathtt{p}(\bar{\mathtt{t}})\vartheta \in H(r\vartheta)$ and $\mathtt{q}(\bar{\mathtt{s}})\vartheta \in Atoms(r\vartheta)$, i.e., $\mathtt{q}(\bar{\mathtt{s}})\vartheta$ is relevant for $\mathtt{p}(\bar{\mathtt{t}})\vartheta$. □

We can now link $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ queries and finitely ground programs.

Theorem 4.3
Let $\mathcal{Q}$ be an $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ query over $\mathcal{P}$. Then $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ is finitely ground.

Proof
Let $\gamma = \langle C_1, \ldots, C_n \rangle$ be a component ordering for $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$. Since each cycle of dependencies in $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ involving predicates of $\mathcal{P}$ is also present in $\mathcal{P}$, components with non-magic predicates are disjoint from components with magic predicates. For a component C_i with magic predicates, $\mathtt{DMS}(\mathcal{Q},\mathcal{P})^{\gamma}_i$ is a subset of M^*, which is finite by Lemma 4.2.

For a component C_i with a non-magic predicate $\mathtt{p_u}$, we consider a modified rule $r' \in P(C_i)$ with an atom $\mathtt{p_u}(\bar{\mathtt{t}}_\mathtt{u}) \in H(r')$:

$$r' : \mathtt{p_1}(\bar{\mathtt{t}}_1)\ \mathtt{v}\ \cdots\ \mathtt{v}\ \mathtt{p_n}(\bar{\mathtt{t}}_\mathtt{n}) :\!- \mathtt{magic\_p_1}(\bar{\mathtt{t}}_1), \ldots, \mathtt{magic\_p_n}(\bar{\mathtt{t}}_\mathtt{n}), \mathtt{q_1}(\bar{\mathtt{s}}_1), \ldots, \mathtt{q_j}(\bar{\mathtt{s}}_\mathtt{j}), \mathtt{not}\ \mathtt{q_{j+1}}(\bar{\mathtt{s}}_{\mathtt{j+1}}), \ldots, \mathtt{not}\ \mathtt{q_m}(\bar{\mathtt{s}}_\mathtt{m}).$$

Thus, the component containing $\mathtt{magic\_p_u}$ precedes C_i in γ. Moreover, since $\mathcal{Q}$ is finitely recursive on $\mathcal{P}$, each variable appearing in r' appears also in $\mathtt{magic\_p_u}(\bar{\mathtt{t}}_\mathtt{u})$. Therefore, $\mathtt{DMS}(\mathcal{Q},\mathcal{P})^{\gamma}_i$ is finite also in this case. □

We are now ready for proving the decidability of brave and cautious reasoning for the class of finitely recursive queries on $\mathrm{ASP}^{\mathrm{fs}}$ programs.

Theorem 4.4
Let $\mathcal{Q}$ be an $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ query over $\mathcal{P}$. Deciding whether $\mathcal{P}$ cautiously/bravely entails $\mathcal{Q}$ is computable.

Proof
From Theorem 3.9, $\mathtt{DMS}(\mathcal{Q},\mathcal{P}) \equiv^{b}_{\mathcal{Q}} \mathcal{P}$ and $\mathtt{DMS}(\mathcal{Q},\mathcal{P}) \equiv^{c}_{\mathcal{Q}} \mathcal{P}$ hold. Since $\mathtt{DMS}(\mathcal{Q},\mathcal{P})$ is finitely ground by Theorem 4.3, decidability follows from Thereom 2.1. □

5 Expressiveness result

In this section, we show that the restrictions which guarantee the decidability of $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ queries do not limit their expressiveness. Indeed, any computable function can be encoded by an $\mathrm{ASP}^{\mathrm{fs}}_{\mathrm{fr}}$ program (even without using disjunction and negation). To this end, we show how to encode a deterministic Turing Machine as a positive program with functions and an input string by means of a query. In fact it is

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

well-known that Horn clauses (under the classic first-order semantics) can represent any computable function (Tärnlund 1977), so we just have to adapt these results for $\text{ASP}^{\text{fs}}_{\text{fr}}$ programs and queries.

A Turing Machine $\mathscr{M}$ with semi-infinite tape is a 5-tuple $\langle \Sigma, \mathscr{S}, \mathtt{s_i}, \mathtt{s_f}, \delta \rangle$, where Σ is an alphabet (i.e., a set of symbols), $\mathscr{S}$ is a set of states, $\mathtt{s_i}, \mathtt{s_f} \in \mathscr{S}$ are two distinct states (representing the initial and final states of $\mathscr{M}$, respectively), and $\delta : \mathscr{S} \times \Sigma \longrightarrow \mathscr{S} \times \Sigma \times \{\leftarrow, \rightarrow\}$ is a transition function. Given an input string $x = \mathtt{x_1} \cdots \mathtt{x_n}$, the initial configuration of $\mathscr{M}$ is such that the current state is $\mathtt{s_i}$, the tape contains x followed by an infinite sequence of blank symbols $\sqcup$ (a special tape symbol occurring in Σ; we are assuming x does not contain any blank symbol), and the head is over the first symbol of the tape. The other configurations assumed by $\mathscr{M}$ with input x are then obtained by means of the transition function δ: If $\mathtt{s}$ and $\mathtt{v}$ are the current state and symbol, respectively, and $\delta(\mathtt{s}, \mathtt{v}) = (\mathtt{s}', \mathtt{v}', \mathtt{m})$, then $\mathscr{M}$ overwrites $\mathtt{v}$ with $\mathtt{v}'$, moves its head according to $\mathtt{m} \in \{\leftarrow, \rightarrow\}$, and changes its state to $\mathtt{s}'$. $\mathscr{M}$ *accepts* x if the final state $\mathtt{s_f}$ is reached at some point of the computation.

A configuration of $\mathscr{M}$ can be encoded by an instance of $\mathtt{conf}(\mathtt{s}, \mathtt{L}, \mathtt{v}, \mathtt{R})$, where $\mathtt{s}$ is the current state, $\mathtt{v}$ the symbol under the head, $\mathtt{L}$ the list of symbols on the left of the head in reverse order, and $\mathtt{R}$ a finite list of symbols on the right of the head containing at least all the non-blank symbols. The query $\mathscr{Q}_{\mathscr{M}(x)}$ representing the initial configuration of $\mathscr{M}$ with input x is

$$\begin{array}{ll} \mathtt{conf}(\mathtt{s_i}, [\,], \mathtt{x_1}, [\mathtt{x_2}, \ldots, \mathtt{x_n}])? & \text{if } \mathtt{n} > 0; \\ \mathtt{conf}(\mathtt{s_i}, [\,], \sqcup, [\,])? & \text{otherwise.} \end{array}$$

The program $\mathscr{P}_{\mathscr{M}}$ encoding $\mathscr{M}$ contains a rule $\mathtt{conf}(\mathtt{s_f}, \mathtt{L}, \mathtt{V}, \mathtt{R})$. representing the final state $\mathtt{s_f}$, and a set of rules implementing the transition function δ. For each state $\mathtt{s} \in \mathscr{S} \setminus \{\mathtt{s_f}\}$ and for each symbol $\mathtt{v} \in \Sigma$, $\mathscr{P}_{\mathscr{M}}$ contains the following rules:

$$\begin{array}{lll} \mathtt{conf}(\mathtt{s}, [\mathtt{V}|\mathtt{L}], \mathtt{v}, \mathtt{R}) \text{ :- } \mathtt{conf}(\mathtt{s}', \mathtt{L}, \mathtt{V}, [\mathtt{v}'|\mathtt{R}]). & & \text{if } \delta(\mathtt{s}, \mathtt{v}) = (\mathtt{s}', \mathtt{v}', \leftarrow); \\ \mathtt{conf}(\mathtt{s}, \mathtt{L}, \mathtt{v}, [\mathtt{V}|\mathtt{R}]) \text{ :- } \mathtt{conf}(\mathtt{s}', [\mathtt{v}'|\mathtt{L}], \mathtt{V}, \mathtt{R}). & & \text{if } \delta(\mathtt{s}, \mathtt{v}) = (\mathtt{s}', \mathtt{v}', \rightarrow); \\ \mathtt{conf}(\mathtt{s}, \mathtt{L}, \mathtt{v}, [\,]) \text{ :- } \mathtt{conf}(\mathtt{s}', [\mathtt{v}'|\mathtt{L}], \sqcup, [\,]). & & \text{if } \delta(\mathtt{s}, \mathtt{v}) = (\mathtt{s}', \mathtt{v}', \rightarrow). \end{array}$$

Note that we do not explicitly represent the infinite sequence of blanks on the right of the tape; the last rule above effectively produces a blank whenever the head moves right of all explicitly represented symbols. The atoms therefore represent only the effectivley reached tape positions. We now show the correctness of $\mathscr{P}_{\mathscr{M}}$ and $\mathscr{Q}_{\mathscr{M}(x)}$.

Theorem 5.1
The program $\mathscr{P}_{\mathscr{M}}$ bravely/cautiously entails $\mathscr{Q}_{\mathscr{M}(x)}$ if and only if $\mathscr{M}$ accepts x.

Proof Sketch
$\mathscr{P}_{\mathscr{M}}$ bravely/cautiously entails $\mathscr{Q}_{\mathscr{M}(x)}$ if and only if the unique stable model of $\mathscr{P}_{\mathscr{M}}$ contains a sequence of atoms $\mathtt{conf}(\bar{\mathtt{t}}_1), \ldots, \mathtt{conf}(\bar{\mathtt{t}}_\mathtt{m})$ such that $\mathtt{conf}(\bar{\mathtt{t}}_1)$ is the query atom, $\mathtt{conf}(\bar{\mathtt{t}}_\mathtt{m})$ is an instance of $\mathtt{conf}(\mathtt{s_f}, \mathtt{L}, \mathtt{V}, \mathtt{R})$, and there is a rule in $\mathit{Ground}(\mathscr{P}_{\mathscr{M}})$ (implementing the transition function of $\mathscr{M}$) having $\mathtt{conf}(\bar{\mathtt{t}}_\mathtt{i})$ in head and $\mathtt{conf}(\bar{\mathtt{t}}_{\mathtt{i}+1})$ in the body, for each $\mathtt{i} = 1, \ldots, \mathtt{m} - 1$. Since instances of $\mathtt{conf}(\bar{\mathtt{t}})$ represent configurations of $\mathscr{M}$, the claim follows. □

We can now link computable sets (or functions) and finitely recursive queries.

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

Theorem 5.2
Let L be a computable set (or function). Then, there is an ASP$^{\text{fs}}$ program $\mathscr{P}$ such that, for each string x, the query $\mathscr{Q}_x$ is finitely recursive on $\mathscr{P}$, and $\mathscr{P}$ cautiously/bravely entails $\mathscr{Q}_x$ if and only if $x \in L$.

Proof
Let $\mathscr{M}$ be a Turing Machine computing L and $\mathscr{P}_{\mathscr{M}}$ be the program encoding $\mathscr{M}$. Program $\mathscr{P}_{\mathscr{M}}$ is clearly in ASP$^{\text{fs}}$ (actually, it is even negation-free). By Theorem 5.1, it only remains to prove that $\mathscr{Q}_{\mathscr{M}(x)}$ is finitely recursive on $\mathscr{P}_{\mathscr{M}}$. By construction of $\mathscr{P}_{\mathscr{M}}$, for each ground atom $\texttt{conf}(\bar{\texttt{t}})$ in $\mathscr{B}_{\mathscr{P}_{\mathscr{M}}}$, there is exactly one rule in $\mathit{Ground}(\mathscr{P}_{\mathscr{M}})$ having $\texttt{conf}(\bar{\texttt{t}})$ in head. This rule has at most one atom $\texttt{conf}(\bar{\texttt{t}}')$ in its body, and implements either the transition function or the final state of $\mathscr{M}$. Thus, the atoms relevant for $\mathscr{Q}_{\mathscr{M}(x)}$ are exactly the atoms representing the configurations assumed by $\mathscr{M}$ with input x. The claim then follows because $\mathscr{M}$ halts in a finite number of steps by assumption. □

We note that when applying magic sets on the Turing machine encoding, the magic predicates effectively encode all reachable configurations, and a bottom-up evaluation of the magic program corresponds to a simulation of the Turing machine. Hence only encodings of Turing machine invocations that visit all (infinitely many) tape cells are not finitely recursive. We also note that recognizing whether an ASP$^{\text{fs}}$ query or a program is finitely recursive is RE-complete[8].

6 Related work

The extension of ASP with functions has been the subject of intensive research in the last years. The main proposals can be classified in two groups:

1. *Syntactically restricted fragments*, such as *ω-restricted programs* (Syrjänen 2001), *λ-restricted programs* (Gebser *et al.* 2007), *finite-domain programs* (Calimeri *et al.* 2008a), *argument-restricted programs* (Lierler and Lifschitz 2009), 𝔽𝔻ℕ𝕔 *programs* (Simkus and Eiter 2007), *bidirectional programs* (Eiter and Simkus 2009), and the proposal of (Lin and Wang 2008); these approaches introduce syntactic constraints (which can be easily checked at small computational cost) or explicit domain restrictions, thus allowing computability of answer sets and/or decidability of querying;

2. *Semantically restricted fragments*, such as *finitely ground programs* (Calimeri *et al.* 2008a), *finitary programs* (Bonatti 2002; Bonatti 2004), *disjunctive finitely-recursive programs* (Baselice *et al.* 2009) and *queries* (Calimeri *et al.* 2009); with respect to syntactically restricted fragments, these approaches aim at identifying broader classes of programs for which computational tasks such as querying are decidable. However, the membership of programs in these fragments is undecidable in general.

There have been a few other proposals that treat function symbols not in the traditional LP sense, but as in classical logic, where most prominently the unique names assumption does not hold. We refer to (Cabalar 2008) for an overview.

[8] That is, complete for the class of recursively enumerable decision problems.

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

Our work falls in the group 2. It is most closely related to (Bonatti 2002), (Baselice *et al.* 2009), and especially (Calimeri *et al.* 2009), which all focus on *querying* for *disjunctive* programs.

The work in (Bonatti 2002) studies how to extend finitary programs (Bonatti 2004) preserving decidability for ground querying in the presence of disjunction. To this end, an extra condition on disjunctive heads is added to the original definition of finitary program of (Bonatti 2004). Interestingly, the class of $\text{ASP}^{\text{fs}}_{\text{fr}}$ programs, which features decidable reasoning (as proved in Theorem 4.4), enlarges the stratified subclass of disjunctive finitary programs of (Bonatti 2002). Indeed, while all stratified finitary programs trivially belong to the class of $\text{ASP}^{\text{fs}}_{\text{fr}}$ programs, the above mentioned extra condition on disjunctive heads is not guaranteed to be fulfilled by $\text{ASP}^{\text{fs}}_{\text{fr}}$ programs (even if negation is stratified or forbidden at all). Instead, in (Baselice *et al.* 2009), a redefinition (including disjunction) of finitely recursive programs is considered, initially introduced in (Bonatti 2004) as a superclass of finitary programs allowing function symbols and negation. The authors show a compactness property and semi-decidability results for cautious ground querying, but no decidability results are given.

Our paper extends and generalizes the work (Calimeri *et al.* 2009), in which the decidability of querying over finitely recursive *negation-free* disjunctive programs is proved via a magic-set rewriting. To achieve the extension, we had to generalize the magic set technique used in (Calimeri *et al.* 2009) to deal also with stratified negation. The feasibility of such a generalization was not obvious at all, since the magic set rewriting of a stratified program can produce unstratified negation (Kemp *et al.* 1995), which can lead to undecidability in the presence of functions. We have proved that, thanks to the structure of $\text{ASP}^{\text{fs}}_{\text{fr}}$ programs and the adopted SIPS, the magic set rewriting preserves stratification. The presence of negation also complicates the proof that the rewritten program is query-equivalent to the original one. To demonstrate this result, we have exploited the characterization of stable models via unfounded sets of (Leone *et al.* 1997), and generalized the equivalence proof of (Alviano *et al.* 2009) to the case of programs with functions.

Finally, our studies on computable fragments of logic programs with functions are loosely related to termination studies of SLD-resolution for Prolog programs (see e.g. (Bruynooghe *et al.* 2007)).

7 Conclusion

In this work we have studied the language of $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries and programs. By adapting a magic set technique, any $\text{ASP}^{\text{fs}}_{\text{fr}}$ query can be transformed into an equivalent query over a finitely ground program, which is known to be decidable and for which an implemented system is available. We have also shown that the $\text{ASP}^{\text{fs}}_{\text{fr}}$ language can express any decidable function. In total, the proposed language and techniques provide the means for a very expressive, yet decidable and practically usable logic programming framework.

Concerning future work, we are working on adapting an existing implementation of a magic set technique to handle $\text{ASP}^{\text{fs}}_{\text{fr}}$ queries as described in this article,

https://doi.org/10.1017/S1471068410000244 Published online by Cambridge University Press

integrating it into DLV-Complex (Calimeri *et al.* 2008b), thus creating a useable $\text{ASP}^{\text{fs}}_{\text{fr}}$ system. We also intend to explore practical application scenarios; promising candidates are query answering over ontologies and in particular the Semantic Web, reasoning about action and change, or analysis of dynamic multi-agent systems.

References

Alviano, M., Faber, W., Greco, G., and Leone, N. 2009. *Magic Sets for Disjunctive Datalog Programs.* Technical Report 09/2009, Dipartimento di Matematica, Università della Calabria, Italy. `http://www.wfaber.com/research/papers/TRMAT092009.pdf`.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press.

Baselice, S., Bonatti, P. A., and Criscuolo, G. 2009. On finitely recursive programs. *Theory and Practice of Logic Programming 9,* 2, 213–238.

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

Bonatti, P. A. 2002. Reasoning with infinite stable models II: Disjunctive programs. In *Proceedings of the 18th International Conference on Logic Programming (ICLP 2002).* Lecture Notes in Computer Science, vol. 2401. Springer, 333–346.

Bonatti, P. A. 2004. Reasoning with infinite stable models. *Artificial Intelligence 156,* 1, 75–111.

Bruynooghe, M., Codish, M., Gallagher, J. P., Genaim, S., and Vanhoof, W. 2007. Termination analysis of logic programs through combination of type-based norms. *ACM Transactions on Programming Languages and Systems (TOPLAS) 29,* 2, 10.

Cabalar, P. 2008. Partial Functions and Equality in Answer Set Programming. In *Proceedings of the 24th International Conference on Logic Programming (ICLP 2008).* Lecture Notes in Computer Science, vol. 5366. Springer, Udine, Italy, 392–406.

Calimeri, F., Cozza, S., Ianni, G., and Leone, N. 2008a. Computable Functions in ASP: Theory and implementation. In *Proceedings of the 24th International Conference on Logic Programming (ICLP 2008).* Lecture Notes in Computer Science, vol. 5366. Springer, Udine, Italy, 407–424.

Calimeri, F., Cozza, S., Ianni, G., and Leone, N. 2008b. `DLV-Complex` homepage. `http://www.mat.unical.it/dlv-complex`.

Calimeri, F., Cozza, S., Ianni, G., and Leone, N. 2009. Magic sets for the bottom-up evaluation of finitely recursive programs. In *Logic Programming and Nonmonotonic Reasoning — 10th International Conference (LPNMR 2009),* E. Erdem, F. Lin, and T. Schaub, Eds. Lecture Notes in Computer Science, vol. 5753. Springer, 71–86.

Eiter, T. and Simkus, M. 2009. Bidirectional answer set programs with function symbols. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09),* C. Boutilier, Ed. Springer, Pasadena, CA, 765–771.

Gebser, M., Schaub, T., and Thiele, S. 2007. Gringo : A new grounder for answer set programming. In *Logic Programming and Nonmonotonic Reasoning—9th International Conference, LPNMR'07,* C. Baral, G. Brewka, and J. Schlipf, Eds. Lecture Notes in Computer Science, vol. 4483. Springer, Tempe, Arizona, 266–271.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9,* 365–385.

Greco, S. 2003. Binding propagation techniques for the optimization of bound disjunctive queries. *IEEE Transactions on Knowledge and Data Engineering 15,* 2 (March/April), 368–385.

Kemp, D. B., Srivastava, D., and Stuckey, P. J. 1995. Bottom-up evaluation and query optimization of well-founded models. *Theoretical Computer Science 146*, 145–184.

Leone, N., Rullo, P., and Scarcello, F. 1997. Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation 135,* 2 (June), 69–112.

Lierler, Y. and Lifschitz, V. 2009. One more decidable class of finitely ground programs. In *Proceedings of the 25th International Conference on Logic Programming (ICLP 2009)*. Lecture Notes in Computer Science, vol. 5649. Springer, Pasadena, CA, 489–493.

Lin, F. and Wang, Y. 2008. Answer set programming with functions. In *Proceedings of Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR2008)*. AAAI Press, Sydney, Australia, 454–465.

Simkus, M. and Eiter, T. 2007. FDNC: Decidable non-monotonic disjunctive logic programs with function symbols. In *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR2007)*. Lecture Notes in Computer Science, vol. 4790. Springer, 514–530.

Syrjänen, T. 2001. Omega-restricted logic programs. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, Vienna, Austria, 267–279.

Tärnlund, S.-Å. 1977. Horn clause computability. *BIT Numerical Mathematics 17,* 2 (June), 215–226.

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