Polynomially Bounded Logic Programs with Function Symbols: A New Decidable Class

Vernon Asuncion and Yan Zhang
Artificial Intelligence Research Group
Western Sydney University, Australia
Email:{v.asuncion, yan.zhang}@westernsydney.edu.au

Heng Zhang
School of Computer Science and Technology
Huazhong University of Science and Technology, China
Email: hengzhang@hust.edu.cn

Abstract

A logic program with function symbols is called finitely ground if there is a finite propositional logic program whose stable models are exactly the same as the stable models of this program. Finite groundability is an important property for logic programs with function symbols because it makes feasible to compute such program's stable models using traditional ASP solvers. In this paper, we introduce a new decidable class of finitely ground programs called POLY-bounded programs, which, to the best of our knowledge, strictly contains all decidable classes of finitely ground programs discovered so far in the literature. We also study the related complexity property for this class of programs. We prove that deciding whether a program is POLY-bounded is EXPTIME-complete.

Introduction

A logic program with function symbols Π is called *finitely ground* if there is a finite propositional logic program Π' such that Π and Π' have the same collection of stable models. Therefore, a finitely ground logic program will have a finite number of stable models and each stable model is of a finite size. Finite groundability is an important property for programs with function symbols because this makes feasible to compute such programs' stable models using traditional ASP solvers (Calimeri et al. 2008; Baselice, Bonatti, and Criscuolo 2009a; Alviano, Faber, and Leone 2010).

Unfortunately, in general, checking whether a program is finitely ground is undecidable. In recent years, several decidable classes of finitely ground programs have been discovered under the *stable model semantics* (Gelfond and Lifschitz 1988): *ω-restricted programs* (Syrjänen 2001), *λ-restricted programs* (Gebser, Schaub, and Thiele 2007), *finite domain programs* (Calimeri et al. 2008), *argument-restricted programs* (Lierler and Lifschitz 2009), *safe programs* (Greco, Spezzano, and Trubitsyna 2012), *Γ-acyclic programs* (Greco, Spezzano, and Trubitsyna 2012), and what we refer as GMT-*bounded programs*, which has been shown to be a proper superclass of all other previous classes (Greco, Molinaro, and Trubitsyna 2013). More recently, another decidable class of finitely ground programs called *size-restricted programs* was further introduced in (Calautti et al. 2015), in which it was shown that although this class does not contain the argument-restricted and GMT-bounded classes, the underlying technique may be combined with other approaches and eventually to identify more finitely ground programs (see next section of an overview on this).

However, it comes to our attention that some simple programs which are finitely ground but do not belong to either of the class of GMT-bounded programs or size-restricted programs. Let us consider a scenario of an online photo gallery, where each paid member can view any image in the gallery, but a restriction is imposed for guest members: Although a guest member is allowed to view the gallery images, he/she can only view no more than two images in either large or thumbnail form each time. This may be expressed by the following two rules:

$$\begin{aligned} r_1:\ & \mathsf{viewLarge}(X,Y) \vee \mathsf{viewThumbnail}\big(\textit{next}(X),Y\big) \\ & \quad \leftarrow \mathsf{viewThumbnail}(X,Y),\ \mathsf{guestMember}(Y), \\ r_2:\ & \bot \leftarrow \mathsf{viewThumbnail}\big(\textit{next}\big(\textit{next}(X)\big),Y\big). \end{aligned}$$

Let $\Pi_1 = \{r_1, r_2\}$. Then it follows that program Π_1 is finitely ground because program $\Pi_1 \cup D$ will only have finite stable models for any given input database D.

Surprisingly, Π_1 is not bounded under the definition of (Greco, Molinaro, and Trubitsyna 2013), nor is size-restricted as specified in (Calautti et al. 2015). Motivated from this example, in this paper, we propose yet another decidable class of logic programs with function symbols called POLY-bounded programs, which strictly contains both GMT-bounded and size restricted programs. The reason we are able to obtain such a strict class is that we give explicit treatments to "disjunctions", "negations" and "constraints" in the underlying programs.

The rest of the paper is organized as follows: Section 2 presents necessary terminologies and background knowledge we will need throughout the paper. Section 3 defines a fixpoint upper bound of all stable models for a given program with function symbols. Based on this upper bound definition, Section 4 then specifies a new decidable class of programs called polynomially bounded programs, and proves its main properties. Section 5 studies the complexity property of this new decidable class of programs, and finally Section 6 concludes the paper with some remarks.

Preliminaries

A *disjunctive logic program* (or simply called *program*) Π is a finite set of rules r of the form:

$$A_1 \vee \ldots \vee A_k \leftarrow B_1, \ldots, B_l, \mathsf{not}\, C_1, \ldots, \mathsf{not}\, C_m, \quad (1)$$

where A_i, B_j, C_h are atoms for all $1 \leq i \leq k, 1 \leq j \leq l$ and $1 \leq h \leq m$. We denote by $Hd(r)$, $Pos(r)$, and $Neg(r)$ the sets $\{A_1, \ldots, A_k\}$, $\{B_1, \ldots, B_l\}$, and $\{C_1, \ldots, C_m\}$, which are called r's *head*, *positive body*, and *negative body*, respectively. Sometimes for convenience, we may simply denote rule r by $Hd(r) \leftarrow Pos(r) \wedge \neg Neg(r)$. When $k \leq 1$ for all $r \in \Pi$, Π is called a *normal program*; if for a rule r, $k = 0$, r is called a *constraint* and we denote its head by $\perp$; and when $Pos(r) \cup Neg(r) = \emptyset$, r is called a *fact*.

Given a predicate p of arity n, the i-th *argument* of p is an expression of the form $p[i]$. Throughout this paper, we denote by $\textsc{arity}(p)$ as p's arity, and refer to an argument as $p[i]$ for $1 \leq i \leq \textsc{arity}(p)$. We denote by $\textsc{arg}(\Pi)$ as the set of all arguments of Π, $\textsc{pred}(\Pi)$ as the set of all predicate symbols in Π, and $\textsc{atoms}(\Pi)$ as the set of all atoms mentioned in Π.

A program Π is *range restricted* if for each rule, the variables occurring in the head or in the negative body also appears in the positive body of that rule. As in (Greco, Molinaro, and Trubitsyna 2013), in this paper, we assume that all programs are range restricted. Given an atom α, we denote by $\textsc{var}(\alpha)$ and $\textsc{const}(\alpha)$ as the sets of variables and constants occurring in α, respectively. Moreover, we naturally extend this notion to a program Π (to a set of atoms A) such that $\textsc{var}(\Pi)$, $\textsc{const}(\Pi)$ and $\textsc{atoms}(\Pi)$ ($\textsc{var}(A)$, $\textsc{const}(A)$ and $\textsc{atoms}(A)$, resp.) denote the set of variables, constants and atoms occurring in Π (A, resp.), respectively. For convenience, we further denote by $\textsc{varConst}(\Pi)$ ($\textsc{varConst}(A)$) as the union $\textsc{var}(\Pi) \cup \textsc{const}(\Pi)$ ($\textsc{var}(A) \cup \textsc{const}(A)$, resp.).

Given two sets of atoms A_1 and A_2, we say that A_1 *is embeddable* into A_2, denoted as $A_1 \ll A_2$, if there exists a mapping $\theta : \textsc{var}(A_1) \longrightarrow \textsc{varConst}(A_2)$ such that for each atom $a_1 \in A_1$, there exists some atom $a_2 \in A_2$ such that $a_1\theta = a_2$[1].

Now for a given program Π, by $\mathrm{HU}(\Pi)$ and $\mathrm{HB}(\Pi)$, we denote Π's *Herbrand universe* and *Herbrand base*, respectively. Specifically, $\mathrm{HU}(\Pi)$ is the set of all ground terms that can be built using the constants and function symbols in Π (if Π does not contain any constant, we introduce a constant in Π's domain), while $\mathrm{HB}(\Pi)$ is the set of all atoms that can be built from terms in $\mathrm{HU}(\Pi)$ and predicate symbols of Π. Clearly, both $\mathrm{HU}(\Pi)$ and $\mathrm{HB}(\Pi)$ can be infinite. We say that a set of atoms I is an *interpretation* of Π iff $I \subseteq \mathrm{HB}(\Pi)$. A rule r' is a *ground instance* of $r \in \Pi$ if r' is obtained from r by substituting each variable in r by some ground term from $\mathrm{HU}(\Pi)$. We use $\textsc{ground}(r)$ to denote all ground instances of r, and $\textsc{ground}(\Pi) = \bigcup_{r \in \Pi} \textsc{ground}(r)$ as the *grounding* of the program Π, which could be infinite. Given an interpretation I and a ground rule r', we say that I *satisfies* r', denoted as $I \models r'$, iff $I \cap Hd(r') \neq \emptyset$ whenever $I \subseteq Pos(r)$ and $I \cap Neg(r) = \emptyset$ holds. Then we say that I is a *model* of a ground program Π', denoted as $I \models \Pi$, iff $I \models r'$ for all $r' \in \Pi'$.

Given an interpretation $I \subseteq \mathrm{HB}(\Pi)$ and the grounding Π' $= \textsc{ground}(\Pi)$ of Π, we denote by $(\Pi')^I$ as the *reduced* (or *reduct*) of the (ground) program Π' such that it is denoted as the set of rules $\{Hd(r) \leftarrow Pos(r) \mid r \in \Pi' \text{ and } Neg(r) \cap I = \emptyset\}$. Then we say that I is an *stable model* of Π iff I is the minimal set that satisfies all the rules in $(\Pi')^I$ (Gelfond and Lifschitz 1988; 1991).

A program Π is in *functional normal form* if for each $p(t_1, \cdots, t_k) \in \textsc{atoms}(\Pi)$, $\textsc{dep}(t_i) \leq 1$ for all $1 \leq i \leq k$, where $\textsc{dep}(t_i)$ denotes the *greatest term depth* of a complex term in t_i. It is obvious that for a given program Π, by introducing new predicate symbols, we can always rewrite Π to a model equivalent program in functional normal form. So as in (Greco, Molinaro, and Trubitsyna 2013; Eiter and Simkus 2009), in the rest of this paper, we will only consider programs in their functional normal form. Also, for a given program Π, a finite set D of facts (D can be empty) is called an *input database* of Π when we consider program $\Pi \cup D$.

GMT-bounded and Size-restricted Programs

Now we introduce the notions of GMT-bounded and size-restricted programs as proposed in (Greco, Molinaro, and Trubitsyna 2013) and (Calautti et al. 2015), respectively.

Generally speaking, for a given program Π, the GMT-boundedness and size-restrictedness are defined through two operators $\Psi_{(\textsc{gmt},\Pi)} : 2^{\textsc{arg}(\Pi)} \longrightarrow 2^{\textsc{arg}(\Pi)}$ and $\Psi_{(\textsc{sr},\Pi)} : 2^{\textsc{arg}(\Pi)} \longrightarrow 2^{\textsc{arg}(\Pi)}$, respectively, such that for a given $\mathcal{A} \subseteq \textsc{arg}(\Pi)$, we say that Π is $\mathcal{A}$-GMT-*bounded* (resp. $\mathcal{A}$-*size-restricted*) iff $\Psi^{\infty}_{(\textsc{gmt},\Pi)}(\mathcal{A}) = \textsc{arg}(\Pi)$ (resp. $\Psi^{\infty}_{(\textsc{sr},\Pi)}(\mathcal{A}) = \textsc{arg}(\Pi)$), where for $i \geq 0$, $\Psi^{i}_{(\textsc{gmt},\Pi)}(\mathcal{A})$ ($\Psi^{i}_{(\textsc{sr},\Pi)}(\mathcal{A})$, resp.) is defined inductively as follows: (1) $\Psi^{0}_{(\textsc{gmt},\Pi)}(\mathcal{A}) = \mathcal{A}$ ($\Psi^{0}_{(\textsc{sr},\Pi)}(\mathcal{A}) = \mathcal{A} \cup \mathcal{R}_{\mathcal{A}}(\Pi)$, where $\mathcal{R}_{\mathcal{A}}(\Pi)$ denotes the $\mathcal{A}$-*size-restricted arguments*[2] of Π, resp.); (2) $\Psi^{i+1}_{(\textsc{gmt},\Pi)}(\mathcal{A}) = \Psi_{(\textsc{gmt},\Pi)}(\Psi^{i}_{(\textsc{gmt},\Pi)}(\mathcal{A}))$ ($\Psi^{i+1}_{(\textsc{sr},\Pi)}(\mathcal{A}) = \Psi_{(\textsc{sr},\Pi)}(\Psi^{i}_{(\textsc{sr},\Pi)}(\mathcal{A}))$). $\Psi^{\infty}_{(\textsc{gmt},\Pi)}(\mathcal{A})$ ($\Psi^{\infty}_{(\textsc{sr},\Pi)}(\mathcal{A})$, resp.) denotes the fixpoint of $\Psi^{i}_{(\textsc{gmt},\Pi)}(\mathcal{A})$ ($\Psi^{i}_{(\textsc{sr},\Pi)}(\mathcal{A})$, resp.).

We say that a program Π is GMT-*bounded* iff $\Psi^{\infty}_{(\textsc{gmt},\Pi)}(\mathrm{AR}(\Pi)) = \textsc{arg}(\Pi)$, where $\mathrm{AR}(\Pi)$ denotes the set of *restricted arguments* (Lierler and Lifschitz 2009). On the other hand, although size-restrictedness specified in (Greco, Molinaro, and Trubitsyna 2013; Calautti et al. 2015) does not readily capture GMT-boundedness, it can nevertheless be incorporated through an "adornment" process achieved by making the set of input argument $\mathcal{A}$ in $\Psi^{\infty}_{(\textsc{sr},\Pi)}(\mathcal{A})$ to be the GMT-bounded arguments. As such, in this paper, when we say that a program Π is *size-restricted*, we mean that $\Psi^{\infty}_{(\textsc{sr},\Pi)}(\mathcal{A}) = \textsc{arg}(\Pi)$ such that $\mathcal{A}$ is the GMT-bounded arguments of Π. It can be showed that this

[1]We slightly generalize the notion $a_1\theta$ where θ maps the tuple of variables in a_1 to a corresponding tuple from $\textsc{varConst}(A_2)$.

[2]We refer readers to (Calautti et al. 2015) for more details about "$\mathcal{A}$-size-restrictedness."

simple program $\{\mathsf{p}(f(X,X),Y,Z) \leftarrow \mathsf{p}(X,g(Z),g(Y))\}$ is size-restricted (Calautti et al. 2015).

An Upper Bound of Progression

Our idea of discovering a new decidable class of programs is described as follows: for a given program Π, (1) we firstly propose a progression based procedure to specify an approximating upper bound $\Gamma(\Pi)$ for all stable models S of program $\Pi \cup D$ for all input database D; and (2) by imposing a proper *polynomial* bound on $\Gamma(\Pi)$, we eventually are able to derive a new decidable class of programs with function symbols which are finitely ground when the corresponding bounds are met by these programs.

For a program Π, by Π^{DEF}, we denote the program obtained from Π via the following transformation:

$$\big\{\, Hd(r') \leftarrow Pos(r') \wedge \neg Neg(r') \;\mid\; r' \text{ is a rule such that } \exists r \in \Pi \text{ where: } (1)\ Hd(r') \in Hd(r);\ (2)\ Pos(r') = Pos(r);\ (3)\ Neg(r') = Neg(r) \cup \big(Hd(r) \setminus \{Hd(r')\}\big) \,\big\}.$$

Intuitively, Π^{DEF} is the normal program obtained from Π by "shifting" (Dix, Gottlob, and Marek 1996). That is, for each rule of Π, only one atom occurring in the head of this rule, while all the other head atoms are shifted to the negative body of the new generated rule.

Now based on programs Π and Π^{DEF}, from the Herbrand base of Π, we specify a procedure that computes the set of all facts that must be *true* or *false* derived from Π.

Definition 1. ***[Deriving lower bound]*** *Let Π be a program. Then $\mathcal{K}^k(\Pi)$ ($k \geq 0$) is inductively defined as follows:*

$$\begin{aligned}\mathcal{K}^0(\Pi) = \;& \big\{\langle \alpha, + \rangle \mid \text{there exists a rule "}\alpha \leftarrow \top\text{"} \in \Pi \text{ and } \alpha \text{ is a ground atom}\big\} \cup \\ & \big\{\langle \alpha\theta, - \rangle \mid \text{there exist a rule "}\bot \leftarrow \alpha\text{"} \in \Pi \text{ and a mapping } \theta : \textsc{var}(\alpha) \longrightarrow \textsc{varConst}(\Pi)\big\};\end{aligned} \tag{2}$$

$$\begin{aligned}\mathcal{K}^{k+1}(\Pi) = \;& \mathcal{K}^k(\Pi) \cup \\ & \big\{\langle \alpha_1\theta_1, + \rangle \mid \text{there exist rules "}\alpha_1 \leftarrow \beta_1, \widehat{Bd}_1\text{"},\ \text{"}\alpha_2 \leftarrow \mathsf{not}\, \beta_2, \widehat{Bd}_2\text{"} \in \Pi^{\text{DEF}} \text{ and mappings } \theta_i (i=1,2): \\ & \quad \theta_i : \textsc{var}(\alpha_i) \cup \textsc{var}(\beta_i) \longrightarrow \textsc{varConst}(\mathcal{K}^k(\Pi)) \text{ such that:} \\ & \quad (1)\ \alpha_1\theta_1 = \alpha_2\theta_2 \text{ and } (\beta_1\theta_1 = \beta_2\theta_2 \text{ or } \{\beta_1, \beta_2\} = \emptyset); \\ & \quad (2)\ \widehat{Bd}_1\theta_1 \subseteq \mathcal{K}^k(\Pi) \text{ and } \widehat{Bd}_2\theta_2 \subseteq \mathcal{K}^k(\Pi)\big\} \cup \end{aligned} \tag{3}$$

$$\begin{aligned}& \big\{\langle \alpha_1\theta_1, - \rangle \mid \text{there exist rules "}\alpha'_1 \leftarrow \alpha_1, \beta_1, \widehat{Bd}_1\text{"},\ \text{"}\alpha'_2 \leftarrow \alpha_2, \mathsf{not}\, \beta_2, \widehat{Bd}_2\text{"} \in \Pi^{\text{DEF}} \text{ and mappings } \theta_i (i=1,2): \\ & \quad \theta_i : \textsc{var}(\alpha_i) \cup \textsc{var}(\beta_i) \longrightarrow \textsc{varConst}(\mathcal{K}^k(\Pi)) \text{ such that:} \\ & \quad (1)\ \alpha_1\theta_1 = \alpha_2\theta_2 \text{ and } (\beta_1\theta_1 = \beta_2\theta_2 \text{ or } \{\beta_1, \beta_2\} = \emptyset); \\ & \quad (2)\ \widehat{Bd}_1\theta_1 \subseteq \mathcal{K}^k(\Pi) \text{ and } \widehat{Bd}_2\theta_2 \subseteq \mathcal{K}^k(\Pi); \\ & \quad (3)\ \alpha'_1\theta_1 \in \mathcal{K}^k(\Pi)^- \text{ and } \alpha'_2\theta_2 \in \mathcal{K}^k(\Pi)^- \big\},\end{aligned} \tag{4}$$

$$\mathcal{K}^\infty(\Pi) = \bigcup_{i=0}^{\infty} \mathcal{K}^i(\Pi).$$

Let $\odot \in \{+,-\}$, $S \subseteq \mathcal{K}^\infty(\Pi)$, and $S^\odot = \{p(\mathbf{X}) \mid \langle p(\mathbf{X}), \odot \rangle \in S\}$. In (3) and (4), $\widehat{Bd}_i$ ($i = 1,2$) denotes the remaining body atoms of the rules "$\alpha'_i \leftarrow \alpha_i, \beta_i, \widehat{Bd}_i$". By $\widehat{Bd}_i\theta_i \subseteq \mathcal{K}^k(\Pi)$, we mean that for all positive and negative atoms $\alpha''_i\theta_i$ and $\mathsf{not}\, \beta''_i\theta_i$ in $\widehat{Bd}_i\theta_i$, $\alpha''_i\theta_i \in \mathcal{K}^k(\Pi)^+$ and $\beta''_i\theta_i \in \mathcal{K}^k(\Pi)^-$.

Let us take a closer look at Definition 1. Generally speaking, $\mathcal{K}^\infty(\Pi)$ induces two subsets of non-ground atoms of Π whose *types* are known to be *definitely true* and *false*, as contained in $\mathcal{K}^\infty(\Pi)^+$ and $\mathcal{K}^\infty(\Pi)^-$, respectively. The base case (2) of the form $\langle \alpha, + \rangle$ and $\langle \alpha\theta, - \rangle$ are derived from the facts and atomic constraints in program Π. Then sets (3) and (4), specified in the inductive step, represent propagations of positive and negative facts, respectively, based on previous stages. Consider (3) for instance, the tuple $\langle \alpha_1\theta_1, + \rangle$ is derived from rules "$\alpha_1 \leftarrow \beta_1, \widehat{Bd}_1$", "$\alpha_2 \leftarrow \beta_2, \widehat{Bd}_2$" $\in \Pi^{\text{DEF}}$ and mappings θ_1 and θ_2, under two conditions (1) and (2) as illustrated in the definition. This is because under these conditions, the resolution rule can be used to derive fact $\alpha_1\theta_1$ from Π. A similar explanation applies to the derived negative fact specified as $\langle \alpha_1\theta_1, - \rangle$.

Example 1. Let Π be a program consisting of the following rules:

r_0: $\bot \leftarrow \mathsf{q}(X)$,
r_1: $\bot \leftarrow \mathsf{not}\, \mathsf{p}(X)$,
r_2: $\mathsf{p}(f(X)) \leftarrow \mathsf{p}(X),\ \mathsf{r}(X),\ \mathsf{not}\, \mathsf{q}(X)$,
r_3: $\mathsf{r}(X) \leftarrow \mathsf{r}(f(X))$,
r_4: $\bot \leftarrow \mathsf{r}(X),\ \mathsf{p}(g(X)),\ \mathsf{not}\, \mathsf{q}(X)$,
r_5: $\bot \leftarrow \mathsf{r}(Y),\ \mathsf{not}\, \mathsf{p}(g(Y)),\ \mathsf{not}\, \mathsf{q}(Y)$.

Then according to Definition 1, we have:
$\mathcal{K}^0(\Pi) = \{\langle \mathsf{q}(X), - \rangle, \langle \mathsf{q}(Y), - \rangle\}$;
$\mathcal{K}^1(\Pi) = \mathcal{K}^0(\Pi) \cup \{\langle \mathsf{r}(X), - \rangle, \langle \mathsf{r}(Y), - \rangle\}$;
$\mathcal{K}^2(\Pi) = \mathcal{K}^1(\Pi) \cup \{\langle \mathsf{r}(f(X)), - \rangle, \langle \mathsf{r}(f(Y)), - \rangle\}$;
$\mathcal{K}^3(\Pi) = \mathcal{K}^2(\Pi)$. □

Now based on Definition 1, we define an upper bound for a given program Π as follows.

Definition 2. ***[Deriving upper bound]*** *Let Π be a program and $S \subseteq \mathcal{K}^\infty(\Pi)$. Then $\Gamma^i_\Pi(S)$ ($i \geq 0$) is inductively defined as follows:*

$$\Gamma^0_\Pi(S) = \big\{\, Hd(r)\theta \mid \exists r \in \Pi^{\text{DEF}} \text{ and mapping } \theta : \textsc{var}(r) \longrightarrow \textsc{varConst}(\Pi) \text{ such that: } (1)\ Pos(r)\theta \cap S^- = \emptyset;\ (2)\ Neg(r)\theta \cap S^+ = \emptyset \,\big\}; \tag{5}$$

$$\Gamma^{i+1}_\Pi(S) = \Gamma^i_\Pi(S) \cup \big\{Hd(r)\theta \mid \exists r \in \Pi^{\text{DEF}} \text{ and mapping } \theta : \textsc{var}(r) \longrightarrow \textsc{varConst}(\Pi) \text{ such that: } (1)\ Pos(r)\theta \subseteq \Gamma^i_\Pi(S);\ (2)\ Pos(r)\theta \cap S^- = \emptyset;\ (3)\ Neg(r)\theta \cap S^+ = \emptyset \,\big\}. \tag{6}$$

Finally, we define $\Gamma_{\Pi}^{\infty}(S) = \bigcup_{i=0}^{\infty} \Gamma_{\Pi}^{i}(S)$ *to be its fixpoint.*

From (5) in Definition 2, we consider $Hd(r)$ and mapping θ only for those rules whose negative or positive bodies are not "defeated" yet by the sets S^+ and S^- (where $S \subseteq \mathcal{K}^{\infty}(\Pi)$), respectively. Then inductively, we have that (6) simply extends those derived "types" of atoms based on the ones obtained from previous steps.

Theorem 1. ***[Upper bound for stable models]*** *Let* Π *be a program and* $S \subseteq \mathcal{K}^{\infty}(\Pi)$. *Then for every input database* D *and stable model* A *of* $\Pi \cup D$, $A \ll \Gamma_{\Pi}^{\infty}(S)$ [3].

Proof. For convenience, given a set of atoms $S' \subseteq \Gamma_{\Pi}^{\infty}(S)$, denote $S' \restriction_{\text{CONST}(\Pi \cup D)}$ as the set of ground atoms $\{ \alpha\theta \mid \alpha \in S' \text{ and } \theta : \text{VAR}(\alpha) \longrightarrow \text{CONST}(\Pi \cup D) \}$. Thus to prove this result, it will be sufficient to show that $A \subseteq \Gamma_{\Pi}^{\infty}(S)\restriction_{\text{CONST}(\Pi \cup D)}$ for any stable model A of $\Pi \cup D$, because it then follows that $A \ll \Gamma_{\Pi}^{\infty}(S)$.

On the contrary, assume that A is a stable model of $\Pi' = \text{GROUND}(\Pi \cup D)$ such that $A \not\subseteq \Gamma_{\Pi}^{\infty}(S)\restriction_{\text{CONST}(\Pi \cup D)}$. Then since A is a stable model of Π', it follows that A is a minimal model of $(\Pi')^A$ (where $(\Pi')^A$ denotes the reduct of Π' on A). Let $A' = A \setminus \{a \in A \mid a \notin \Gamma_{\Pi}^{\infty}(S)\restriction_{\text{CONST}(\Pi \cup D)}\}$, i.e., A' is the set obtained from A by deleting all the atoms that are not mentioned in $\Gamma_{\Pi}^{\infty}(S)\restriction_{\text{CONST}(\Pi \cup D)}$. Then since $A \not\subseteq \Gamma_{\Pi}^{\infty}(S)\restriction_{\text{CONST}(\Pi \cup D)}$ (which implies that $\exists a \in A$ such that $a \notin \Gamma_{\Pi}^{\infty}(S)\restriction_{\text{CONST}(\Pi \cup D)}$), it follows that $A' \subset A$ where $A' \subseteq \Gamma_{\Pi}^{\infty}(S)\restriction_{\text{CONST}(\Pi \cup D)}$. Then we can prove the result: $A' \models (\Pi')^A$ (we omit the full proof of this result here due to a space limit), from which and the fact that $A' \subset A$, we conclude a contradiction that A is a minimal model of $(\Pi')^A$. □

Polynomially Bounded Programs

According to Theorem 1, it is clear that if $\Gamma_{\Pi}^{\infty}(S)$ (for some $S \subseteq \mathcal{K}^{\infty}(\Pi)$) is a finite set, then Π is finitely ground. Also, from Definitions 1 and 2, we can see that if for each atom in $\Gamma_{\Pi}^{\infty}(S)$, its term depth is bounded by a fixed integer, then $\Gamma_{\Pi}^{\infty}(S)$ must be a finite set. For a given set A of atoms, let $\text{DEP}(A)$ denote the set of all maximum term depths of all arguments $p[i]$ mentioned in A. Now our attempt is to impose a bound $\mathcal{B}$ on $\text{DEP}\big(\Gamma_{\Pi}^{\infty}(S)\big)$ such that $\text{DEP}\big(\Gamma_{\Pi}^{\mathcal{B}}(S)\big) = \text{DEP}\big(\Gamma_{\Pi}^{\infty}(S)\big)$. In this section, we will identify a new class of programs called *polynomially bounded programs* by defining a term depth bound for $\Gamma_{\Pi}^{\mathcal{B}}(S)$ to be a polynomial in the size of program Π.

Firstly, given a program Π, we define

$$\mathcal{I}(\Pi) \;=\; N + N^3, \tag{7}$$

where $N = |\Pi^{\text{DEF}}| \cdot \text{MAXART}(\Pi) \cdot \text{MAXPOS}(\Pi)$. Here $\text{MAXPOS}(\Pi)$ denotes the maximum number of atoms in the positive body of a rule in Π, and $\text{MAXART}(\Pi)$ denotes the product of the maximum arities of a predicate and function symbols occurring in Π, i.e., $\text{MAXART}(\Pi) = m \times n$, where m and n are the maximum arities of the predicates and function symbols occurring in Π, respectively.

Intuitively, $\mathcal{I}(\Pi)$, as defined via (7), gives an approximation of the minimum bound on the number of iterations of $\Gamma_{\Pi}^{k}(S)$ that has to be done in order to determine if an infinite propagation of terms may actually take place. In a nutshell, iterating through $\Gamma_{\Pi}^{k}(S)$, for $1 \leq k \leq \mathcal{I}(\Pi)$, considers all the possible "transpositions" and "propagations" of an argument, say $p[i]$, within the program Π as we iterate through each step.

At the same time, such iteration will compute the maximum possible depth of any restricted arguments, as well as the possible "undoing" of these complex terms. (i.e., the failing cycles). Indeed, from the definition of $\mathcal{I}(\Pi)$ in (7), the factor $\text{MAXART}(\Pi)$ considers all the possible transpositions of $p[i]$ within the arities of predicates and functions. In addition, the number $|\Pi^{\text{DEF}}| \cdot \text{MAXPOS}(\Pi)$ also factors in the possible transposition that can be propagated through each positive atom in the program.

In fact, the number of iterative steps $\mathcal{I}(\Pi)$ does three things: (1) the number $N = |\Pi^{\text{DEF}}| \cdot \text{MAXART}(\Pi) \cdot \text{MAXPOS}(\Pi)$ considers the iterative steps required to generate the deepest term of a restricted argument because it bounds the length of the greatest possible path that can derive a complex term of a restricted argument; (2) the additional number N^3 further adds the additional steps that are required to "undo" the complex terms compounded in the restricted arguments from doing the aforementioned first N-steps because it is the product of the number of the deepest possible term (bounded by N) with that of the maximum cycle length (also bounded by N) that can "undo" the complexity of the term and where the one more factor of N (which makes the term "cubed" in (7)) considers the possibility that each can take N-steps to exhaust each of the possible positions of arguments in atoms; and (3) the fact that doing $\mathcal{I}(\Pi)$-steps considers all the possible transpositions and propagations of an argument allows us to detect any growing cycles corresponding to unlimited growth of complex terms within the argument as well since it will also allow us to detect recursive information about function applications.

Given a set A of atoms, we denote by $\text{DEP}_{p[i]}(A)$ as the maximum term depth of the argument $p[i]$ mentioned in A, where we define $\text{DEP}_{p[i]}(S) = 0$ if $S = \emptyset$. Now, given a predicate $p \in \text{PRED}(\Pi)$, denote by $\text{POLYLA}_p(\Pi)$ as the set of arguments:

$$\Big\{ p[i] \mid \text{DEP}_{p[i]}\big(\Gamma_{\Pi}^{\mathcal{I}(\Pi)}(S)\big) = \text{DEP}_{p[i]}\big(\Gamma_{\Pi}^{2\cdot\mathcal{I}(\Pi)+1}(S)\big) \Big\},$$

where $S = \mathcal{K}^{\mathcal{I}(\Pi)}(\Pi)$, i.e., the set of arguments of the predicate p that does not grow beyond the stage of iterations $\mathcal{I}(\Pi)$ of $\Gamma_{\Pi}^{k}(S)$ with S to be the set of non-ground atoms obtained from $\mathcal{K}^{\mathcal{I}(\Pi)}(\Pi)$. More generally, we call the arguments in the set $\bigcup_{p \,\in\, \text{PRED}(\Pi)} \text{POLYLA}_p(\Pi)$ as the POLY-*limited arguments* of $\text{ARG}(\Pi)$, which we denote by $\text{POLYLA}(\Pi)$, i.e., just omitting the subscript of the particular predicate p.

Definition 3. ***[POLY-bounded programs]*** *Given a program* Π*, we say that* Π *is polynomially bounded, or simply called* POLY-bounded, *iff* $\text{POLYLA}(\Pi) = \text{ARG}(\Pi)$.

---

[3] Recall from Section Preliminaries that given two sets of atoms A_1 and A_2, $A_1 \ll A_2$ denotes that A_1 is embeddable into A_2.

Intuitively, Definition 3 says that if a program is POLY-bounded, then we have that all arguments cannot grow beyond the number of $2 \cdot \mathcal{I}(\Pi) + 1$ iterations of $\Gamma_{\Pi}^{k}(S)$, where $S = \mathcal{K}^{\mathcal{I}(\Pi)}(\Pi)$. Note that Definition 3 defines the polynomial bound $2 \cdot \mathcal{I}(\Pi) + 1$ for computing $\Gamma_{\Pi}^{k}(S)$, instead of $\mathcal{I}(\Pi) + 1$. This is due to the possibility that the growth of term depth in Π may run through multiple arguments, from which we may only gain sufficient information to predict if the iteration will continue or stop, by computing the second run of iterations through all arguments.

Example 2. Consider program Π_1 we discussed in Section 1. We first rewrite this program to its functional normal form Π_1' as follows:

$$
\begin{aligned}
r_1: \ & \mathsf{viewLarge}(X,Y) \vee \mathsf{viewThumbnail}\big(\mathit{next}(X),Y\big) \\
& \quad \leftarrow \mathsf{viewThumbnail}(X,Y),\ \mathsf{guestMember}(Y), \\
r_3: \ & \bot \leftarrow \mathsf{viewThumbnail}_1\big(\mathit{next}(X),Y\big), \\
r_4: \ & \mathsf{viewThumbnail}_1(X,Y) \\
& \quad \leftarrow \mathsf{viewThumbnail}\big(\mathit{next}(X),Y\big).
\end{aligned}
$$

Then from r_3 and r_4 we can conclude that $\mathsf{viewThumbnail}_1\big(\mathit{next}(X), Y\big)$, $\mathsf{viewThumbnail}\big(\mathit{next}(\mathit{next}(X)), Y\big) \in S = \mathcal{K}^{\mathcal{I}(\Pi_1')}(\Pi_1')^-$. Thus, it follows from Definitions 2 and 3 that the growth of the term in the argument $\mathsf{viewThumbnail}[1]$ as propagated via rule r_1 is bounded, which implies $\text{POLYLA}(\Pi_1') = \text{ARG}(\Pi_1')$. That is, Π_1' is POLY-bounded. $\square$

Proposition 1. *If Π is POLY-bounded, then for every input database D (D can be empty), program $\Pi \cup D$ is finitely ground.*

Theorem 2. *GMT-bounded $\subsetneq$ POLY-bounded, and SR $\subsetneq$ POLY-bounded, where GMT-bounded, SR and POLY-bounded denote the three classes of GMT-bounded, size-restricted and POLY-bounded programs, respectively.*

Computational Complexity

In this section, we study the complexity properties of POLY-bounded programs. We are mainly interested in the complexity of membership decision problem for this class of programs. Firstly, we have proved the following result for GMT-bounded and size-restricted programs.

Proposition 2. *Deciding whether a program Π is GMT-bounded or size-restricted is in PSPACE.*

Theorem 3. *Deciding whether a program Π is POLY-bounded is EXPTIME-complete. The hardness holds even if Π's maximum function arity is 2.*

Proof. (*Sketch*) The membership can be obtained by outlining a membership decision procedure based on Definitions 1, 2 and 3.

Here we provide the main idea and procedure of proving the hardness. Let L be an arbitrary decision problem in EXPTIME. Then from the definition of complexity class EXPTIME (Papadimitriou 1994), there exists some *deterministic Turing machine* M such that for any string $\mathbf{s}$, $\mathbf{s} \in L$ iff M accepts $\mathbf{s}$ in at most $2^{p(|\mathbf{s}|)}$ steps for some polynomial $p(n)$. Thus, assume the Turing machine M to be the tuple $\langle Q, \Gamma, \Box, \Sigma, \delta, q_0, F\rangle$ such that: (1) $Q \neq \emptyset$ is a finite set of states; (2) $\Gamma \neq \emptyset$ is a finite set of alphabet symbols; (3) $\Box \in \Gamma$ is the "blank" symbol; (4) $\Sigma \subseteq \Gamma \setminus \{\Box\}$ is the set of input symbols; (5) $\delta : (Q \setminus F) \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function; (6) $q_0 \in Q$ is the initial state; and lastly, (7) $F \subseteq Q$ is the set of final/accepting states.

Now given a string $\mathbf{s} = a_0 \ldots a_{|\mathbf{s}|-1}$ such that $a_i \in \Sigma$ for $0 \leq i < |\mathbf{s}|$, we construct a program[4] $\Pi_{M(\mathbf{s})} = \Pi^{\text{ORD}}_{M(\mathbf{s})} \cup \Pi^{\text{STR}}_{M(\mathbf{s})} \cup \Pi^{\text{STRASSOC}}_{M(\mathbf{s})} \cup \Pi^{\text{EDGES}}_{M(\mathbf{s})} \cup \Pi^{\text{TRANS}}_{M(\mathbf{s})} \cup \Pi^{\text{ACCEPT}}_{M(\mathbf{s})} \cup \Pi^{\text{BOUND}}_{M(\mathbf{s})}$.

Program $\Pi^{\text{ORD}}_{M(\mathbf{s})}$ is to generate the linear ordering on the $p(|\mathbf{s}|)$-length tuples that will encode the computation time/steps as well as the individual cell-positions in the tape.

Program $\Pi^{\text{STR}}_{M(\mathbf{s})}$ generates all possible strings of lengths from 1 to $|\Gamma|^{p(|\mathbf{s}|)}$ under the alphabets of Γ, as defined in the following:

$$\Pi^{\text{STR}}_{M(\mathbf{s})} =$$

$$\big\{\, s_0^0(a_i, \bar{\mathbf{i}}, \bar{\mathbf{i}}) \leftarrow \top \ \mid\ 0 \leq i \leq n-1 \,\big\} \cup \tag{8}$$

$$\big\{\, s_0^0(\Box, \mathbf{T}, \mathbf{T}) \leftarrow \overline{\mathbf{n-1}} < \mathbf{T} \,\big\} \cup \tag{9}$$

$$\begin{aligned}\big\{\, & s_0^k(X \circ Y, \mathbf{T}_1, \mathbf{T}_4) \leftarrow s_0^i(X, \mathbf{T}_1, \mathbf{T}_2), s_0^j(Y, \mathbf{T}_3, \mathbf{T}_4), \\ & \qquad \mathbf{T}_1 \leq \mathbf{T}_2, \mathbf{T}_2 \prec \mathbf{T}_3, \mathbf{T}_3 \leq \mathbf{T}_4 \\ & \mid 0 \leq i, j < k \leq p(|\mathbf{s}|) \,\big\} \cup \end{aligned} \tag{10}$$

$$\begin{aligned}\big\{\, & s^0(a, \mathbf{T}, \mathbf{T}) \leftarrow num(T_1), \ldots, num(T_{p(|\mathbf{s}|)}) \\ & \mid a \in \Gamma \text{ and } \mathbf{T} = T_1 \ldots T_{p(|\mathbf{s}|)} \,\big\} \cup \end{aligned} \tag{11}$$

$$\begin{aligned}\big\{\, & s^k(X \circ Y, \mathbf{T}_1, \mathbf{T}_4) \leftarrow s^i(X, \mathbf{T}_1, \mathbf{T}_2), s^j(Y, \mathbf{T}_3, \mathbf{T}_4), \\ & \qquad \mathbf{T}_1 \leq \mathbf{T}_2, \mathbf{T}_2 \prec \mathbf{T}_3, \mathbf{T}_3 \leq \mathbf{T}_4 \\ & \mid 0 \leq i, j < k \leq p(|\mathbf{s}|) \,\big\} \cup \end{aligned} \tag{12}$$

$$\begin{aligned}\big\{\, & s_0(X, \mathbf{T}_1, \mathbf{T}_2) \leftarrow s_0^i(X, \mathbf{T}_1, \mathbf{T}_2), \\ & s(X, \mathbf{T}_1, \mathbf{T}_2) \leftarrow s^i(X, \mathbf{T}_1, \mathbf{T}_2) \ \mid 0 \leq i \leq p(|\mathbf{s}|) \,\big\}. \end{aligned} \tag{13}$$

Without loss of generality, we assume $|\Gamma| > 2$, therefore, it is sufficient to use strings of length from 1 to $|\Gamma|^{p(|\mathbf{s}|)}$ to encode all possible $2^{p(|\mathbf{s}|)}$ $M(\mathbf{s})$ configurations. In program $\Pi^{\text{STR}}_{M(\mathbf{s})}$, we define the function "$\circ$" taking arguments of two strings $\mathbf{s}_1$ and $\mathbf{s}_2$ so that $\mathbf{s}_1 \circ \mathbf{s}_2$ denotes the concatenation of $\mathbf{s}_1$ and $\mathbf{s}_2$. Then due to the transitivity rules (10) and (12), it is observed that it would only take $O\big(p(|\mathbf{s}|)\big)$-steps to generate all such strings of lengths from 1 to $|\Gamma|^{p(|\mathbf{s}|)}$.

Here predicate s_0 is used to represent the input string on the tape, and predicates s_0^i ($0 \leq i \leq p(|\mathbf{s}|)$) are used to generate such initial string; while predicate s represents an arbitrary string on the tape, which is generated from predicates s^i ($0 \leq i \leq p(|\mathbf{s}|)$).

The program $\Pi^{\text{STRASSOC}}_{M(\mathbf{s})}$, on the other hand, encodes the string *associative property* axioms that are needed by the concatenation function "$\circ$" (here we omit the definition of this program).

[4] Due to a space limit, below we only provide the major program definitions.

Programs $\Pi^{\text{EDGES}}_{M(\mathbf{s})}$ and $\Pi^{\text{TRANS}}_{M(\mathbf{s})}$ described below then encode the machine $M(\mathbf{s})$'s configuration changes based on the corresponding state transitions in $M(\mathbf{s})$, for that we view that the input string s is accepted by machine $M(\mathbf{s})$ as the problem of reachability from the initial configuration of $M(\mathbf{s})$ to $M(\mathbf{s})$'s final (accepting) configuration.

$$\begin{aligned}
&\Pi^{\text{EDGES}}_{M(\mathbf{s})} = \\
&\big\{\, cf\big(\mathbf{X}_t, q, X \circ a, \overline{\mathbf{0}}, \mathbf{X}_{tp} \parallel c \circ Y, \mathbf{Y}_{tp}, \overline{\mathbf{N}}\big) \\
&\quad \vdash cf\big(\mathbf{Y}_t, q', X \circ b \circ c, \overline{\mathbf{0}}, \mathbf{Y}_{tp} \parallel Y, \mathbf{Z}_{tp}, \overline{\mathbf{N}}\big) \\
&\quad\quad \leftarrow \mathbf{X}_t \prec \mathbf{Y}_t, \mathbf{X}_{tp} \prec \mathbf{Y}_{tp}, \mathbf{Y}_{tp} \prec \mathbf{Z}_{tp}, \\
&\quad\quad\quad s(X \circ a, \overline{\mathbf{0}}, \mathbf{X}_{tp}),\ s(c \circ Y, \mathbf{Y}_{tp}, \overline{\mathbf{N}}), \\
&\quad\quad\quad s(X \circ b \circ c, \overline{\mathbf{0}}, \mathbf{Y}_{tp}),\ s(Y, \mathbf{Z}_{tp}, \overline{\mathbf{N}}), \\
&\quad \mid\ \delta(q,a) = (q', b, R),\ c \in \Gamma \text{ and } N = |\Gamma|^{p(|\mathbf{s}|)} \,\big\} \cup
\end{aligned} \tag{14}$$

$$\begin{aligned}
&\big\{\, cf\big(\mathbf{X}_t, q, X \circ c \circ a, \overline{\mathbf{0}}, \mathbf{X}_{tp} \parallel Y, \mathbf{Y}_{tp}, \overline{\mathbf{N}}\big) \\
&\quad \vdash cf\big(\mathbf{Y}_t, q', X \circ c, \overline{\mathbf{0}}, \mathbf{Z}_{tp} \parallel b \circ Y, \mathbf{X}_{tp}, \overline{\mathbf{N}}\big) \\
&\quad\quad \leftarrow \mathbf{X}_t \prec \mathbf{Y}_t, \mathbf{Z}_{tp} \prec \mathbf{X}_{tp}, \mathbf{X}_{tp} \prec \mathbf{Y}_{tp}, \\
&\quad\quad\quad s(X \circ c \circ a, \overline{\mathbf{0}}, \mathbf{X}_{tp}),\ s(Y, \mathbf{Y}_{tp}, \overline{\mathbf{N}}), \\
&\quad\quad\quad s(X \circ c, \overline{\mathbf{0}}, \mathbf{Z}_{tp}),\ s(b \circ Y, \mathbf{X}_{tp}, \overline{\mathbf{N}}), \\
&\quad \mid\ \delta(q,a) = (q', b, L),\ c \in \Gamma \text{ and } N = |\Gamma|^{p(|\mathbf{s}|)} \,\big\};
\end{aligned} \tag{15}$$

$$\begin{aligned}
&\Pi^{\text{TRANS}}_{M(\mathbf{s})} = \\
&\quad \big\{\, cf(\overline{\mathbf{X}}) \Vdash cf(\overline{\mathbf{Y}}) \leftarrow cf(\overline{\mathbf{X}}) \vdash cf(\overline{\mathbf{Y}}), \\
&\quad\quad cf(\overline{\mathbf{X}}) \Vdash cf(\overline{\mathbf{Z}}) \leftarrow cf(\overline{\mathbf{X}}) \Vdash cf(\overline{\mathbf{Y}}), \\
&\quad\quad\quad cf(\overline{\mathbf{Y}}) \Vdash cf(\overline{\mathbf{Z}}) \,\big\};
\end{aligned} \tag{16}$$

Here notation "$cf\big(\mathbf{X}_t, q, V, \overline{\mathbf{0}}, \mathbf{X}_{tp} \parallel W, \mathbf{Y}_{tp}, \overline{\mathbf{N}}\big)$" mentioned in (14), and denoted as "$cf(\overline{\mathbf{X}})$" in (16), represents the machine's configuration. For a space reason, we omit the detailed explanation on this encoding. The expression

$$\begin{aligned}
&\text{“}cf\big(\mathbf{X}_t, q, V, \overline{\mathbf{0}}, \mathbf{X}_{tp} \parallel W, \mathbf{Y}_{tp}, \overline{\mathbf{N}}\big) \\
&\quad \vdash cf\big(\mathbf{Y}_t, q', X, \overline{\mathbf{0}}, \mathbf{Y}_{tp} \parallel Y, \mathbf{Z}_{tp}, \overline{\mathbf{N}}\big)\text{”},
\end{aligned} \tag{17}$$

denotes a relation encoding a configuration change, for a given state transition $\delta(q,a) = (q', b, R)$ in $M(\mathbf{s})$. The expression "$cf(\overline{\mathbf{X}}) \Vdash cf(\overline{\mathbf{Y}})$" in (16) encodes the transitive extension of "$\vdash$", for which it is read: configuration "$cf(\overline{\mathbf{Y}})$" is reached from configuration "$cf(\overline{\mathbf{X}})$".

What program $\Pi^{\text{EDGES}}_{M(\mathbf{s})}$ does is to establish the initial connections between any two configurations based on the input state transitions from $M(\mathbf{s})$, which we call *edges*. For instance, suppose $M(\mathbf{s})$ accepts string s in $2^{p(|\mathbf{s}|)}$ steps, through the sequence of configuration changes: $cf_0, cf_1, \cdots, cf_{2^{p(|\mathbf{s}|)}-1}$, then $\Pi^{\text{EDGES}}_{M(\mathbf{s})}$ will establish edges $cf_0 \vdash cf_1, cf_1 \vdash cf_2, \cdots, cf_{2^{p(|\mathbf{s}|)}-2} \vdash cf_{2^{p(|\mathbf{s}|)}-1}$.

Then the transitive closure of $\Vdash$, which is defined based on $\vdash$ through transitive rules in $\Pi^{\text{TRANS}}_{M(\mathbf{s})}$, is derived by the following manner: firstly, the reachability between any two configurations within two steps is derived: $cf_0 \Vdash cf_2, cf_1 \Vdash cf_3, cf_2 \Vdash cf_4, \cdots, cf_{2^{p(|\mathbf{s}|)}-3} \Vdash cf_{2^{p(|\mathbf{s}|)}-1}$, then in the second run of the evaluation, the reachability between any two configurations within four steps are derived: $cf_0 \Vdash cf_4$, $cf_1 \Vdash cf_5$, $\cdots$, $cf_{2^{p(|\mathbf{s}|)}-5} \Vdash cf_{2^{p(|\mathbf{s}|)}-1}$. This process continues until the reachability from cf_0 to $cf_{2^{p(|\mathbf{s}|)}-1}$. i.e., $cf_0 \Vdash cf_{2^{p(|\mathbf{s}|)}-1}$, is derived . As we will prove in Lemma 1, $cf_0 \Vdash cf_{2^{p(|\mathbf{s}|)}-1}$ will be derived within polynomial steps iff $M(\mathbf{s})$ accepts s in $2^{p(|\mathbf{s}|)}$ steps.

Finally, we have

$$\begin{aligned}
&\Pi^{\text{ACCEPT}}_{M(\mathbf{s})} = \\
&\big\{\, accept \leftarrow cf\big(\overline{\mathbf{0}}, q_0, a_0, \overline{\mathbf{0}}, \overline{\mathbf{0}} \parallel a_1 \circ X, \overline{\mathbf{1}}, \overline{\mathbf{N}}\big) \\
&\quad\quad \Vdash cf\big(\mathbf{X}_t, q, Y, \overline{\mathbf{0}}, \mathbf{X}_{tp} \parallel Z, \mathbf{Y}_{tp}, \overline{\mathbf{N}}\big), \\
&\quad \overline{\mathbf{0}} < \mathbf{X}_t,\ \mathbf{X}_{tp} \prec \mathbf{Y}_{tp}, \\
&\quad s_0(a_0, \overline{\mathbf{0}}, \overline{\mathbf{0}}), s_0(a_1 \circ X, \overline{\mathbf{1}}, \overline{\mathbf{N}}), \\
&\quad s(Y, \overline{\mathbf{0}}, \mathbf{X}_{tp}), s(Z, \mathbf{Y}_{tp}, \overline{\mathbf{N}}) \\
&\mid q \in F \text{ and } N = |\Gamma|^{p(|\mathbf{s}|)} \big\};
\end{aligned} \tag{18}$$

$$\Pi^{\text{BOUND}}_{M(\mathbf{s})} = \big\{\, r\big(f(X)\big) \leftarrow r(X), \text{not}\, accept \,\big\}, \tag{19}$$

where $\Pi^{\text{ACCEPT}}_{M(\mathbf{s})}$ simply derives the propositional atom "*accept*" if a configuration at an accepting state in $F \subseteq Q$ can be reached from the initial configuration under the input string represented via predicate s_0; otherwise, program $\Pi^{\text{BOUND}}_{M(\mathbf{s})}$ will make the entire program $\Pi_{M(\mathbf{s})}$ unbounded. Then we can prove the following lemma:

Lemma 1. *M accepts* s *iff* $\Gamma^{\mathcal{I}(\Pi_{M(\mathbf{s})})}(S) = \Gamma^{2\cdot\mathcal{I}(\Pi_{M(\mathbf{s})})+1}(S)$, *i.e.,* $\Pi_{M(\mathbf{s})}$ *is* POLY*-bounded, where* $\mathcal{I}(\Pi_{M(\mathbf{s})}) = O(p(|\mathbf{s}|^{10}))$*, and* $S = \mathcal{K}^{\mathcal{I}(\Pi_{M(\mathbf{s})})}$.

□

According to Theorem 3, we can see that compared to other decidable classes, the membership of POLY-bounded programs requires extra computations, which is consistent with the fact that this class of programs strictly contains all previous decidable classes.

Concluding Remarks

The problem of logic program termination has been extensively studied under the *top-down evaluation* approach over the years: (Schreye and Decorte 1994; Voets and Schreye 2011; Marchiori 1996; Ohlebusch 2001; Genaim and Codish 2005; Serebrenik and Schreye 2005; Nishida and Vidal 2010; Schneider-Kamp, Giesl, and Nguyen 2009; Schneider-Kamp et al. 2009; 2010; Nguyen et al. 2007; Bruynooghe et al. 2007; Bonatti 2004; Baselice, Bonatti, and Criscuolo 2009b; Zhang, Zhang, and You 2015). However, as pointed by Greco, *et al*. (2013), under the stable model semantics, these methods are not generally applicable to identify finitely ground programs.

In this paper, by proposing the stable model polynomial upper bound for logic programs, we discovered a new decidable class of finitely ground programs, which strictly contains the other newly identified decidable classes named GMT-bounded and size-restricted programs (Greco, Molinaro, and Trubitsyna 2013; Calautti et al. 2015).

References

Alviano, M.; Faber, W.; and Leone, N. 2010. Disjunctive ASP with functions: Decidable queries and effective computation. *Theory and Practice of Logic Programming* 10(4-6):497–512.

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

Baselice, S.; Bonatti, P. A.; and Criscuolo, G. 2009b. On finitely recursive programs. *TPLP* 9(2):213–238.

Bonatti, P. A. 2004. Reasoning with infinite stable models. *Artif. Intell.* 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 Trans. Program. Lang. Syst.* 29(2).

Calautti, M.; Greco, S.; Molinaro, C.; and Trubitsyna, I. 2015. logic program termination analysis using atom sizes. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI- 2015)*, 2833–2839.

Calimeri, F.; Cozza, S.; Ianni, G.; and Leone, N. 2008. Computable functions in ASP: theory and implementation. In *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, 407–424.

Dix, J.; Gottlob, G.; and Marek, W. 1996. Reducing disjunctive to non-disunctive semantics by shift-operations. *Fundamenta Informaticae* 28(12):87–100.

Eiter, T., and Simkus, M. 2009. Fdnc: Decidable nonmonotonic disjunctive logic programs with function symbols. *ACM Transactions on Computational Logic* 9:1–45.

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 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, 266–271.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of ICLP/SLP 1988*, 1070–1080.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programming and disjunctive databases. *New Generation Computing* 9:365–386.

Genaim, S., and Codish, M. 2005. Inferring termination conditions for logic programs using backwards analysis. *TPLP* 5(1-2):75–91.

Greco, S.; Molinaro, C.; and Trubitsyna, I. 2013. Bounded programs: A new decidable class of logic programs with function symbols. In *Proceedings of IJCAI-2013*, 926–913.

Greco, S.; Spezzano, F.; and Trubitsyna, I. 2012. On the termination of logic programs with function symbols. In *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, 323–333.

Lierler, Y., and Lifschitz, V. 2009. One more decidable class of finitely ground programs. In *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, 489–493.

Marchiori, M. 1996. Proving existential termination of normal logic programs. In *Algebraic Methodology and Software Technology, 5th International Conference, AMAST '96, Munich, Germany, July 1-5, 1996, Proceedings*, 375–390.

Nguyen, M. T.; Giesl, J.; Schneider-Kamp, P.; and Schreye, D. D. 2007. Termination analysis of logic programs based on dependency graphs. In *Logic-Based Program Synthesis and Transformation, 17th International Symposium, LOPSTR 2007, Kongens Lyngby, Denmark, August 23-24, 2007, Revised Selected Papers*, 8–22.

Nishida, N., and Vidal, G. 2010. Termination of narrowing via termination of rewriting. *Appl. Algebra Eng. Commun. Comput.* 21(3):177–225.

Ohlebusch, E. 2001. Termination of logic programs: Transformational methods revisited. *Appl. Algebra Eng. Commun. Comput.* 12(1/2):73–116.

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison Wesley.

Schneider-Kamp, P.; Giesl, J.; Serebrenik, A.; and Thiemann, R. 2009. Automated termination proofs for logic programs by term rewriting. *ACM Trans. Comput. Log.* 11(1).

Schneider-Kamp, P.; Giesl, J.; Ströder, T.; Serebrenik, A.; and Thiemann, R. 2010. Automated termination analysis for logic programs with cut. *TPLP* 10(4-6):365–381.

Schneider-Kamp, P.; Giesl, J.; and Nguyen, M. T. 2009. The dependency triple framework for termination of logic programs. In *Logic-Based Program Synthesis and Transformation, 19th International Symposium, LOPSTR 2009, Coimbra, Portugal, September 2009, Revised Selected Papers*, 37–51.

Schreye, D. D., and Decorte, S. 1994. Termination of logic programs: The never-ending story. *J. Log. Program.* 19/20:199–260.

Serebrenik, A., and Schreye, D. D. 2005. On termination of meta-programs. *TPLP* 5(3):355–390.

Syrjänen, T. 2001. Omega-restricted logic programs. In *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, 267–279.

Voets, D., and Schreye, D. D. 2011. Non-termination analysis of logic programs with integer arithmetics. *TPLP* 11(4-5):521–536.

Zhang, H.; Zhang, Y.; and You, J.-H. 2015. Existential rule languages with finite chase: Complexity and expressiveness. In *Proceedings of AAAI-2015*, 691–697.