Checking Termination of Datalog with Function Symbols Through Linear Constraints

Checking Termination of Datalog with Function Symbols Through Linear Constraints

Marco Calautti, Sergio Greco, Cristian Molinaro, and Irina Trubitsyna
{calautti,greco,molinaro,trubitsyna}@dimes.unical.it

DIMES, Università della Calabria, 87036 Rende (CS), Italy

Discussion Paper

Abstract. Enriching Datalog programs with function symbols makes modeling easier, increases the expressive power, and allows us to deal with infinite domains. However, this comes at a cost: common inference tasks become undecidable. To cope with this issue, recent research has focused on finding trade-offs between expressivity and decidability by identifying classes of logic programs that impose limitations on the use of function symbols but guarantee decidability of common inference tasks. Despite the significant body of work in this area, current approaches do not include many simple practical programs whose evaluation terminates. In this paper, we present the novel class of *rule-bounded programs*. While current techniques perform a limited analysis of how terms are propagated from an individual argument to another, our technique is able to perform a more global analysis, thereby overcoming several limitations of current approaches. We show different results on the correctness and the expressivity of the proposed technique.

Keywords: Datalog with function symbols, bottom-up evaluation, program evaluation termination

1 Introduction

Enriching datalog programs with function symbols has recently seen a surge in interest. Function symbols make modeling easier, increase the expressive power, and allow us to deal with infinite domains. At the same time, this comes at a cost: common inference tasks (e.g., cautious and brave reasoning) become undecidable.

Recent research has focused on identifying classes of logic programs that impose some limitations on the use of function symbols but guarantee decidability of common inference tasks. Efforts in this direction are the class of *finitely-ground* programs [9] and the more general class of *bounded term-size* programs [24]. The minimal model of a finitely-ground datalog program is finite, and the well-founded model of a bounded term-size datalog program is finite. Unfortunately, checking if a program is bounded term-size or even finitely-ground is semi-decidable.

Considering the minimal model semantics, decidable subclasses of finitely-ground programs have been proposed (for more details see the last section of this paper). However, there are still many simple practical programs which are finitely-ground but are not detected by any of the current termination criteria. Below are two examples.

Example 1. The following program $\mathcal{P}_1$ implements the bubble sort algorithm:

$$
\begin{array}{ll}
r_0: & \mathtt{bub}(\mathtt{L}, [\,], [\,]) \leftarrow \mathtt{input}(\mathtt{L}).\\
r_1: & \mathtt{bub}([\mathtt{Y}|\mathtt{Z}], [\mathtt{X}|\mathtt{Cur}], \mathtt{Sol}) \leftarrow \mathtt{bub}([\mathtt{X}|[\mathtt{Y}|\mathtt{Z}]], \mathtt{Cur}, \mathtt{Sol}), \mathtt{X} \leq \mathtt{Y}.\\
r_2: & \mathtt{bub}([\mathtt{X}|\mathtt{Z}], [\mathtt{Y}|\mathtt{Cur}], \mathtt{Sol}) \leftarrow \mathtt{bub}([\mathtt{X}|[\mathtt{Y}|\mathtt{Z}]], \mathtt{Cur}, \mathtt{Sol}), \mathtt{Y} < \mathtt{X}.\\
r_3: & \mathtt{bub}(\mathtt{Cur}, [\,], [\mathtt{X}|\mathtt{Sol}]) \leftarrow \mathtt{bub}([\mathtt{X}|[\,]], \mathtt{Cur}, \mathtt{Sol}).
\end{array}
$$

Here `input` is a base predicate symbol whose extension is a fact containing the list we would like to sort. The bottom-up evaluation of this program always terminates for any input list. The ordered list `Sol` can be obtained from the atom $\mathtt{bub}([\,], [\,], \mathtt{Sol})$ in the program's minimal model. $\square$

Example 2. Program $\mathcal{P}_2$ below performs a depth-first traversal of an input tree:

$$
\begin{array}{ll}
r_0: & \mathtt{visit}(\mathtt{Tree}, [\,], [\,]) \leftarrow \mathtt{input}(\mathtt{Tree}).\\
r_1: & \mathtt{visit}(\mathtt{Left}, [\mathtt{Root}|\mathtt{Visited}], [\mathtt{Right}|\mathtt{ToVisit}]) \leftarrow\\
& \qquad\qquad\qquad \mathtt{visit}(\mathtt{tree}(\mathtt{Root}, \mathtt{Left}, \mathtt{Right}), \mathtt{Visited}, \mathtt{ToVisit}).\\
r_2: & \mathtt{visit}(\mathtt{Next}, \mathtt{Visited}, \mathtt{ToVisit}) \leftarrow \mathtt{visit}(\mathtt{null}, \mathtt{Visited}, [\mathtt{Next}|\mathtt{ToVisit}]).
\end{array}
$$

Here `input` is a base predicate symbol whose extension contains a tree-like structure represented by means of the ternary function symbol `tree`. The program visits the nodes of the tree and puts them in a list following a depth-first search. The list `L` of visited elements can be obtained from the atom $\mathtt{visit}(\mathtt{null}, \mathtt{L}, [\,])$ in the program's minimal model. For instance, if the input tree is

$$\mathtt{input}(\mathtt{tree}(\mathtt{a}, \mathtt{tree}(\mathtt{c}, \mathtt{null}, \mathtt{tree}(\mathtt{d}, \mathtt{null}, \mathtt{null})), \mathtt{tree}(\mathtt{b}, \mathtt{null}, \mathtt{null}))).$$

the program produces the list $[\mathtt{b}, \mathtt{d}, \mathtt{c}, \mathtt{a}]$ containing the nodes of the tree in opposite order w.r.t. the traversal. $\square$

The peculiarity of the above programs is that the overall size of the terms in the body does not increase during their propagation to the head, as there is only a simple redistribution of terms. One of the novelties of the technique proposed in this paper is the capability of doing this kind of analysis, thereby identifying finitely-ground programs that none of the current techniques include.

Contribution. In this paper we present a novel technique for checking if a Datalog program is finitely-ground, initially appeared in [6]. Our technique overcomes several limitations of current approaches being able to perform a more global analysis of how terms are propagated from the body to the head of rules. To this end, we use linear constraints to measure and relate the size of head and body atoms of rules and introduce the class of *rule-bounded* programs. We study the relationship between the proposed class and current termination criteria.

Organization. Section 2 reports preliminaries on Datalog programs with function symbols. Section 3 introduces the class of rule-bounded programs. Related work and conclusions are reported in Section 4.

2 Preliminaries

Syntax. We assume to have (pairwise disjoint) infinite sets of *logical variables*, *predicate symbols*, and *function symbols*. Each predicate and function symbol g is associated with an *arity*, denoted $arity(g)$, which is a non-negative integer. Function symbols of arity 0 are called *constants*. Variables appearing in logic programs are called "logical variables" and will be denoted by upper-case letters in order to distinguish them from variables appearing in linear constraints, which are called "integer variables" and will be denoted by lower-case letters. A *term* is either a logical variable, or an expression of the form $f(t_1, ..., t_m)$, where f is a function symbol of arity $m \geq 0$ and $t_1, ..., t_m$ are terms.

An *atom* is of the form $p(t_1, ..., t_n)$, where p is a predicate symbol of arity $n \geq 0$ and $t_1, ..., t_n$ are terms. A *rule* r is of the form $H \leftarrow B_1, ..., B_k$, where $k \geq 0$, and $H, B_1, ..., B_k$, are atoms. The atom H is called the *head* of r and is denoted by $head(r)$. The conjunction $B_1, ..., B_k$ is called the *body* of r and is denoted by $body(r)$. With a slight abuse of notation, we sometimes use $body(r)$ to also denote the *set* of literals appearing in the body of r. *A datalog program* is a finite set of rules. We assume that programs are *range restricted*, i.e., for every rule, every logical variable appears in some body atom. W.l.o.g., we also assume that different rules do not share logical variables.

A term (resp. atom, rule, program) is *ground* if no logical variables occur in it. A ground rule with an empty body is also called a *fact*. A predicate symbol p is *defined by* a rule r if p appears in the head of r.

A *substitution* θ is of the form $\{X_1/t_1, ..., X_n/t_n\}$, where $X_1, ..., X_n$ are distinct logical variables and $t_1, ..., t_n$ are terms. The result of applying θ to an atom (or term) A, denoted $A\theta$, is the atom (or term) obtained from A by simultaneously replacing each occurrence of a logical variable X_i in A with t_i if X_i/t_i belongs to θ. Two atoms A_1 and A_2 *unify* if there exists a substitution θ, called a *unifier* of A_1 and A_2, such that $A_1\theta = A_2\theta$.

Semantics. Consider a program $\mathcal{P}$. The *Herbrand universe* $H_\mathcal{P}$ of $\mathcal{P}$ is the possibly infinite set of ground terms which can be built using constants and function symbols appearing in $\mathcal{P}$. The *Herbrand base* $B_\mathcal{P}$ of $\mathcal{P}$ is the set of ground atoms which can be built using predicate symbols appearing in $\mathcal{P}$ and ground terms of $H_\mathcal{P}$. A rule r' is a *ground instance* of a rule r in $\mathcal{P}$ if r' can be obtained from r by substituting every logical variable in r with some ground term in $H_\mathcal{P}$. We use $ground(r)$ to denote the set of all ground instances of r and $ground(\mathcal{P})$ to denote the set of all ground instances of the rules in $\mathcal{P}$, i.e., $ground(\mathcal{P}) = \cup_{r \in \mathcal{P}} ground(r)$. An *interpretation* of $\mathcal{P}$ is any subset I of $B_\mathcal{P}$. The truth value of a ground atom A w.r.t. I, denoted $value_I(A)$, is *true* if $A \in I$, *false* otherwise. A ground rule r is *satisfied* by I, denoted $I \models r$, if there is a ground atom A in $body(r)$ s.t. $value_I(A) = false$ or there $value_I(head(r)) = true$. Thus, if the body of r is empty, r is satisfied by I if $value_I(head(r)) = true$. An interpretation of $\mathcal{P}$ is a *model* of $\mathcal{P}$ if it satisfies every ground rule in $ground(\mathcal{P})$. A model M of $\mathcal{P}$ is minimal if no proper subset of M is a model of $\mathcal{P}$. The minimal model of $\mathcal{P}$ is unique and is denoted by $\mathcal{MM}(\mathcal{P})$.

3 Rule-bounded Programs

In this section, we present *rule-bounded programs*, a class of programs whose evaluation always terminates and for which checking membership in the class is decidable. Their definition relies on a novel technique which uses linear inequalities to measure terms and atoms' sizes and checks if the size of the head of a rule is always bounded by the size of some body atom.

We use $\mathbb{N}$ to denote the set of natural numbers $\{1, 2, 3, ...\}$ and $\mathbb{N}_0$ to denote the set of natural numbers including zero. Moreover, $\mathbb{N}^k = \{(v_1, ..., v_k) \mid v_i \in \mathbb{N} \text{ for } 1 \le i \le k\}$ and $\mathbb{N}_0^k = \{(v_1, ..., v_k) \mid v_i \in \mathbb{N}_0 \text{ for } 1 \le i \le k\}$. Given a k-vector $\overline{v} = (v_1, ..., v_k)$ in $\mathbb{N}_0^k$, we use $\overline{v}[i]$ to refer to v_i, for $1 \le i \le k$. Given two k-vectors $\overline{v} = (v_1, ..., v_k)$ and $\overline{w} = (w_1, ..., w_k)$ in $\mathbb{N}_0^k$, we use $\overline{v} \cdot \overline{w}$ to denote the classical scalar product, i.e., $\overline{v} \cdot \overline{w} = \sum_{i=1}^{k} v_i \cdot w_i$.

The following definition introduces the notions of term and atom size.

Definition 1. *Let t be a term. The* size *of t is recursively defined as follows:*

$$size(t) = \begin{cases} x & \text{if } t \text{ is a logical variable } X; \\ m + \sum\limits_{i=1}^{m} size(t_i) & \text{if } t = f(t_1, ..., t_m). \end{cases}$$

where x is an integer variable. The size *of an atom $A = p(t_1, ..., p_n)$, denoted $size(A)$, is the n-vector $(size(t_1), ..., size(t_n))$.* □

In the definition above, an integer variable x intuitively represents the possible sizes that the logical variable X can have during the bottom-up evaluation. The size of a term of the form $f(t_1, ..., t_m)$ is defined by summing up the size of its terms t_i's plus the arity m of f. Note that from the definition above, the size of every constant is 0.

Example 3. Consider rule r_1 of program $\mathcal{P}_1$ (see Example 1). Using `lc` to denote the list constructor operator "|", the rule can be rewritten as follows:

$$\texttt{bub}(\texttt{lc}(\texttt{Y},\texttt{Z}), \texttt{lc}(\texttt{X},\texttt{Cur}), \texttt{Sol}) \leftarrow \texttt{bub}(\texttt{lc}(\texttt{X}, \texttt{lc}(\texttt{Y},\texttt{Z})), \texttt{Cur}, \texttt{Sol}), \texttt{X} \le \texttt{Y}.$$

Let A (resp. B) be the atom in the head (resp. the first atom in the body). Then,

$$\begin{array}{l} size(A) = (2 + y + z, \quad 2 + x + cur, \quad sol) \\ size(B) = (2 + [x + (2 + y + z)], \quad cur, \quad sol) \end{array}$$ □

We are now ready to define rule-bounded programs.

Definition 2 (Rule-bounded programs). *Let $\mathcal{P}$ be a program and $pred(\mathcal{P}) = \{p_1, ..., p_k\}$. We say that $\mathcal{P}$ is* rule-bounded *if there exist k vectors $\overline{\alpha}_{p_h} \in \mathbb{N}^{arity(p_h)}$, $1 \le h \le k$, such that for every rule $r \in \mathcal{P}$ with $A = head(r) = p_i(t_1, ..., t_n)$, there exists an atom $B = p_j(u_1, ..., u_m)$ in $body(r)$ s.t. the following inequality is satisfied*

$$\overline{\alpha}_{p_j} \cdot size(B) - \overline{\alpha}_{p_i} \cdot size(A) \ge 0$$

for every non-negative value of the integer variables in $size(B)$ and $size(A)$. □

Intuitively, for every rule of $\mathcal{P}$, Definition 2 checks if the size of the head atom is bounded by the size some body atom for all possible sizes the terms can assume.

Example 4. Consider again program $\mathcal{P}_1$ of Example 1. To determine if the program is rule-bounded we need to find $\overline{\alpha}_{input} \in \mathbb{N}$ and $\overline{\alpha}_{bub} \in \mathbb{N}^3$ such that there is an atom in each rule's body which satisfy the inequalities derived from rules of $\mathcal{P}_1$ for all non-negative values of the integer variables therein. In particular, selecting the first body atom in each rule we obtain the following linear constraints:

$$\begin{cases} \overline{\alpha}_{input} \cdot l_0 - \overline{\alpha}_{bub} \cdot (l_0, 0, 0) \geq 0 \\ \overline{\alpha}_{bub} \cdot (4 + x_1 + y_1 + z_1, cur_1, sol_1) - \overline{\alpha}_{bub} \cdot (2 + y_1 + z_1, 2 + x_1 + cur_1, sol_1) \geq 0 \\ \overline{\alpha}_{bub} \cdot (4 + x_2 + y_2 + z_2, cur_2, sol_2) - \overline{\alpha}_{bub} \cdot (2 + x_2 + z_2, 2 + y_2 + cur_2, sol_2) \geq 0 \\ \overline{\alpha}_{bub} \cdot (2 + x_3 + 0,\ cur_3,\ sol_3) - \overline{\alpha}_{bub} \cdot (cur_3,\ 0,\ 2 + x_3 + sol_3) \geq 0 \end{cases}$$

where subscripts associated with integer variables are used to refer to the occurrences of logical variables in different rules (e.g., y_2 is the integer variable associated to the logical variable `Y` in rule r_2).

By expanding the scalar products and isolating every integer variable we get:

$$\begin{cases} (\overline{\alpha}_{input}[1] - \overline{\alpha}_{bub}[1]) \cdot l_0 \geq 0 \\ (\overline{\alpha}_{bub}[1] - \overline{\alpha}_{bub}[2]) \cdot x_1 + (2\overline{\alpha}_{bub}[1] - 2\overline{\alpha}_{bub}[2]) \geq 0 \\ (\overline{\alpha}_{bub}[1] - \overline{\alpha}_{bub}[2]) \cdot y_2 + (2\overline{\alpha}_{bub}[1] - 2\overline{\alpha}_{bub}[2]) \geq 0 \\ (\overline{\alpha}_{bub}[1] - \overline{\alpha}_{bub}[3]) \cdot x_3 + (\overline{\alpha}_{bub}[2] - \overline{\alpha}_{bub}[1]) \cdot cur_3 + (2\overline{\alpha}_{bub}[1] - 2\overline{\alpha}_{bub}[3]) \geq 0 \end{cases}$$

The previous inequalities must hold for all $l_0, x_1, y_2, x_3, cur_3 \in \mathbb{N}_0$; it is easy to see that this is the case iff the following system admits a solution:

$$\begin{cases} \overline{\alpha}_{input}[1] - \overline{\alpha}_{bub}[1] \geq 0 \\ \overline{\alpha}_{bub}[1] - \overline{\alpha}_{bub}[2] \geq 0 \qquad 2\overline{\alpha}_{bub}[1] - 2\overline{\alpha}_{bub}[2] \geq 0 \\ \overline{\alpha}_{bub}[1] - \overline{\alpha}_{bub}[2] \geq 0 \qquad 2\overline{\alpha}_{bub}[1] - 2\overline{\alpha}_{bub}[2] \geq 0 \\ \overline{\alpha}_{bub}[1] - \overline{\alpha}_{bub}[3] \geq 0 \qquad 2\overline{\alpha}_{bub}[2] - 2\overline{\alpha}_{bub}[1] \geq 0 \qquad 2\overline{\alpha}_{bub}[1] - 2\overline{\alpha}_{bub}[3] \geq 0 \end{cases}$$

A possible solution is $\overline{\alpha}_{input} = (1)$ and $\overline{\alpha}_{bub} = (1, 1, 1)$. Consequently, $\mathcal{P}_1$ is rule-bounded. □

The method in the previous example to find vectors $\overline{\alpha}_p$ for all $p \in pred(\mathcal{P})$ can always be applied. That is, we can always isolate the integer variables in the original inequalities and then derive one inequality for each expression that multiplies an integer variable plus the one for the constant term, imposing that all such expressions must be greater than or equal to 0.

Example 5. Applying the method above to the program $\mathcal{P}_2$ of Example 2, we obtain the following constraints:

$$\begin{cases} \overline{\alpha}_{input} \cdot 0 - \overline{\alpha}_v \cdot (0, 0, 0) \geq 0 \\ (\overline{\alpha}_v[1] - \overline{\alpha}_v[2]) \cdot root_1 + (\overline{\alpha}_v[1] - \overline{\alpha}_v[3]) \cdot right_1 + (3\overline{\alpha}_v[1] - 2\overline{\alpha}_v[2] - 2\overline{\alpha}_v[3]) \geq 0 \\ (\overline{\alpha}_v[3] - \overline{\alpha}_v[1]) \cdot next_2 + 2\overline{\alpha}_v[3] \geq 0 \end{cases}$$

where subscript v stands for predicate symbol `visit`. By setting $\overline{\alpha}_v = (2, 1, 2)$, we get positive integer values of $\overline{\alpha}_v[1], \overline{\alpha}_v[2], \overline{\alpha}_v[3]$ s.t. the inequalities above are satisfied for all $root_1$, $right_1$, $next_2 \in \mathbb{N}_0$. Thus, $\mathcal{P}_2$ is rule-bounded. □

Two key properties of rule-bounded programs are: they are finitely-ground and it is decidable to check whether a given program is rule-bounded.

Theorem 1. *Every rule-bounded program is finitely-ground.* □

Theorem 2. *Checking whether a program is rule-bounded is in NP.* □

Theorem 3. *Rule-bounded programs are incomparable with mapping-restricted and bounded programs.* □

Observe that in the previous theorem we have considered only the most general subclasses of finitely-ground programs proposed so far, which generalize previous classes such as argument-restricted programs [19].

The proposed technique can be further improved by identifying in each rule body atoms that are mutually recursive with the rule head. If the head atom contains only variables occurring in some atom not mutually recursive with its head, the rule is not dangerous and its consideration can be omitted. Moreover, in the remaining rules only body atoms that are mutually recursive with the rule head should be considered to construct the inequality.

4 Discussion and Conclusions

A significant body of work has been done on termination of logic programs under top-down evaluation [11,34,20,23,10,28,22,26,27,21,5,4,3] and in the area of term rewriting [35,30,2,12,13]. Termination properties of query evaluation for normal programs under tabling have been studied in [24,25,32].

In this paper, we consider Datalog programs with function symbols under the minimal model semantics, and thus all the excellent works above cannot be straightforwardly applied to our setting—for a discussion on this see, e.g., [9,1]. In our context, [9] introduced the class of finitely-ground programs, guaranteeing the existence of a finite set of stable models, each of finite size, for programs in the class. Since membership in the class is not decidable, decidable subclasses have been proposed: *ω-restricted* [31], *λ-restricted* [14], *finite domain* [9], *argument-restricted* [19], *safe* and *Γ-acyclic* [7], *mapping-restricted* [8], *bounded programs* [16]. An adornment-based approach that can be used in conjunction with the techniques above to detect more programs as finitely-ground has been proposed in [17].

Compared with the aforementioned classes, rule-bounded programs allow us to perform a more global analysis and identify *many practical programs* as finitely-ground, such as those where terms in the body are rearranged in the head, which are not included in any of the classes above. We observe that there are also programs which are not rule-bounded but are recognized as finitely-ground by some of the aforementioned techniques (see Theorems 3).

Similar concepts of "term size" have been considered to check termination of logic programs evaluated in a top-down fashion [29], in the context of partial evaluation to provide conditions for strong termination and quasi-termination [33,18], and in the context of tabled resolution [24,25]. These approaches are geared to work under top-down evaluation, looking at how terms are propagated from the head to the body, while our approach is developed to work under bottom-up evaluation, looking at how terms are propagated from the body to the head. This gives rise to significant differences in how the program analysis is carried out, making one approach not applicable in the setting of the other. As a simple example, the rule $\texttt{p(X)} \leftarrow \texttt{p(X)}$ leads to a non-terminating top-down evaluation, while it is completely harmless under bottom-up evaluation.

We notice that this topic is also related to research done in the database community on termination of the chase procedure, see [15] for a recent survey.

As a direction for future work, we plan to investigate how our techniques can be combined with current termination criteria. Since they look at programs from different standpoints, an interesting issue is to study how they can be integrated so that they can benefit from each other.

References

1. M. Alviano, W. Faber, and N. Leone. Disjunctive ASP with functions: Decidable queries and effective computation. *TPLP*, 10(4-6):497–512, 2010.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
3. S. Baselice, P. A. Bonatti, and G. Criscuolo. On finitely recursive programs. *TPLP*, 9(2):213–238, 2009.
4. P. A. Bonatti. Reasoning with infinite stable models. *Artificial Intelligence*, 156(1):75–111, 2004.
5. M. Bruynooghe, M. Codish, J. P. Gallagher, S. Genaim, and W. Vanhoof. Termination analysis of logic programs through combination of type-based norms. *ACM Trans. Program. Lang. Syst.*, 29(2), 2007.
6. M. Calautti, S. Greco, C. Molinaro, and I. Trubitsyna. Checking termination of logic programs with function symbols through linear constraints. In *RuleML*, pages 97–111, 2014.
7. M. Calautti, S. Greco, F. Spezzano, and I. Trubitsyna. Checking termination of bottom-up evaluation of logic programs with function symbols. *TPLP*, 2014.
8. M. Calautti, S. Greco, and I. Trubitsyna. Detecting decidable classes of finitely ground logic programs with function symbols. In *PPDP*, pages 239–250, 2013.
9. F. Calimeri, S. Cozza, G. Ianni, and N. Leone. Computable functions in ASP: Theory and implementation. In *ICLP*, pages 407–424, 2008.
10. M. Codish, V. Lagoon, and P. J. Stuckey. Testing for termination with monotonicity constraints. In *ICLP*, pages 326–340, 2005.
11. D. De Schreye and S. Decorte. Termination of logic programs: The never-ending story. *Journal of Logic Programming*, 19/20:199–260, 1994.
12. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *J. Autom. Reas.*, 40(2-3):195–220, 2008.
13. M. C. F. Ferreira and H. Zantema. Total termination of term rewriting. *Appl. Algebra Eng. Commun. Comput.*, 7(2):133–162, 1996.

14. M. Gebser, T. Schaub, and S. Thiele. Gringo: A new grounder for answer set programming. In *LPNMR*, pages 266–271, 2007.
15. S. Greco, C. Molinaro, and F. Spezzano. *Incomplete Data and Data Dependencies in Relational Databases.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
16. S. Greco, C. Molinaro, and I. Trubitsyna. Bounded programs: A new decidable class of logic programs with function symbols. In *IJCAI*, pages 926–932, 2013.
17. S. Greco, C. Molinaro, and I. Trubitsyna. Logic programming with function symbols: Checking termination of bottom-up evaluation through program adornments. *TPLP*, 13(4-5):737–752, 2013.
18. M. Leuschel and G. Vidal. Fast offline partial evaluation of logic programs. *Information and Computation*, 235(0):70–97, 2014.
19. Y. Lierler and V. Lifschitz. One more decidable class of finitely ground programs. In *ICLP*, pages 489–493, 2009.
20. M. Marchiori. Proving existential termination of normal logic programs. In *Algebraic Methodology and Software Technology*, pages 375–390, 1996.
21. M. T. Nguyen, J. Giesl, P. Schneider-Kamp, and D. De Schreye. Termination analysis of logic programs based on dependency graphs. In *LOPSTR*, pages 8–22, 2007.
22. N. Nishida and G. Vidal. Termination of narrowing via termination of rewriting. *Appl. Algebra Eng. Commun. Comput*, 21(3):177–225, 2010.
23. E. Ohlebusch. Termination of logic programs: Transformational methods revisited. *Appl. Algebra Eng. Commun. Comput.*, 12(1/2):73–116, 2001.
24. F. Riguzzi and T. Swift. Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. *TPLP*, 13(2):279–302, 2013.
25. F. Riguzzi and T. Swift. Terminating evaluation of logic programs with finite three-valued models. *ACM Transactions on Computational Logic*, 2014.
26. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination proofs for logic programs by term rewriting. *ACM Trans. Comput. Log.*, 11(1), 2009.
27. P. Schneider-Kamp, J. Giesl, T. Ströder, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs with cut. *TPLP*, 10(4-6):365–381, 2010.
28. A. Serebrenik and D. De Schreye. On termination of meta-programs. *TPLP*, 5(3):355–390, 2005.
29. K. Sohn and A. Van Gelder. Termination detection in logic programs using argument sizes. In *PODS*, pages 216–226, 1991.
30. C. Sternagel and A. Middeldorp. Root-labeling. In *Rewriting Techniques and Applications*, pages 336–350, 2008.
31. T. Syrjanen. Omega-restricted logic programs. In *LPNMR*, pages 267–279, 2001.
32. S. Verbaeten, D. De Schreye, and K. F. Sagonas. Termination proofs for logic programs with tabling. *ACM Trans. Comput. Log.*, 2(1):57–92, 2001.
33. G. Vidal. Quasi-terminating logic programs for ensuring the termination of partial evaluation. In *PEPM*, pages 51–60, 2007.
34. D. Voets and D. De Schreye. Non-termination analysis of logic programs with integer arithmetics. *TPLP*, 11(4-5):521–536, 2011.
35. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24(1/2):89–105, 1995.