Termination Criteria for Datalog with Function Symbols

Termination Criteria for Datalog with Function Symbols

Marco Calautti, Cristian Molinaro, Chiara Pulice, and Irina Trubitsyna

DIMES, Università della Calabria
87036, Rende (CS), Italy
{calautti,cmolinaro,cpulice,trubitsyna}@dimes.unical.it

Discussion Paper

Abstract. Enriching Datalog 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 Datalog programs allowing only a limited use of function symbols but guaranteeing decidability of common inference tasks.
In this paper, we provide a survey of current *termination criteria*, which define conditions guaranteeing that a Datalog program (possibly with function symbols) has a finite number of stable models, each of them is of finite size and can be computed. We also present a technique which can be used in conjunction with current termination criteria to improve them.

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

1 Introduction and Preliminaries

In recent years, there has been a great deal of interest in enhancing Datalog by supporting function symbols. Function symbols often make modeling easier and the resulting encodings more readable and concise, but unfortunately, common inference tasks become undecidable in their presence.

The class of *finitely-ground programs*, proposed in [2], guarantees decidability of common inference tasks. In fact, a finitely-ground program has a finite number of stable models, each of them is of finite size and can be computed. Since membership in the class is semi-decidable, research has focused on identifying sufficient conditions for a program to be finitely-ground, leading to different criteria, that we call *termination criteria*. Efforts in this direction are ω*-restricted programs* [10], λ*-restricted programs* [3], *finite domain programs* [2], *argument-restricted programs* [9], *safe programs* [8], Γ*-acyclic programs* [8], *mapping-restricted programs* [1], and *bounded programs* [6].

In this paper, we give an overview of recent research on this topic. Specifically, we present some recently proposed decidable termination criteria and an orthogonal technique that can be used in conjunction with them to enlarge the class of programs recognized as finitely-ground [7].

We assume the reader is familiar with Datalog with function symbols under the *stable model semantics* [4]. Below we introduce notation and terminology used hereafter. Given a (Datalog) program $\mathcal{P}$, we denote by $arg(\mathcal{P})$ the set of all arguments of $\mathcal{P}$, i.e., expressions of the form $p[i]$ where p is a predicate symbol of arity n appearing in $\mathcal{P}$ and $1 \leq i \leq n$. We use $body(r)$ and $head(r)$ to denote the body and the head of a rule r, respectively; $body^+(r)$ denotes the conjunction of all positive literals in $body(r)$. Given a rule r, $ground(r)$ denotes the set of rules obtained by replacing variables with ground terms constructed using constants and function symbols occurring in $\mathcal{P}$. An argument $q[i] \in arg(\mathcal{P})$ is said to be *limited* if it takes values from a finite domain, that is, if for every (stable) model M of $\mathcal{P}$ the projection of Q on the i-th argument is a finite set, where Q is the set of q-atoms in M. We consider programs where rules are *range restricted*, that is all variables occurring in a rule r also occur in $body^+(r)$ and distinguish *base predicate symbols*, defined only by facts (i.e., ground rules with empty body) from *derived predicate symbols*, defined by arbitrary rules. For ease of presentation, we sometimes consider only positive programs as the techniques described can be easily extended to programs with negative body literals and disjunction in head.

Termination criteria are used to determine sets of arguments which are limited. Given a program $\mathcal{P}$ and a criterion W, $W(\mathcal{P})$ denotes the set of arguments which are recognized as limited by criterion W. The class of programs which are recognized as finitely-ground by W is the class of programs $\mathcal{P}$ such that $arg(\mathcal{P}) = W(\mathcal{P})$.

2 Basic Termination Criteria

In this section, we describe "basic" termination criteria proposed in the literature, namely *argument-restricted programs* [9] and *mapping-restricted programs* [1]. We shall not discuss other well-known basic termination criteria, such as *ω-restricted programs* [10], *λ-restricted programs* [3] and *finite domain programs* [2], as they are generalized by the criteria discussed in this section. We named the aforementioned termination criteria "basic" as their definition does not rely on other termination criteria.

Argument-restricted programs *[9].* The argument-restricted criterion checks whether we can find, for each argument, a *finite* upper bound of the depth of terms that may occur in that argument during the program evaluation. This test is based on the notion of *argument ranking function* defined below. For any atom A of the form $p(t_1, ..., t_n)$, A^0 denotes predicate symbol p, and A^i denotes term t_i, for $1 \leq i \leq n$.

Definition 1. *An* argument ranking *for a program $\mathcal{P}$ is a partial function ϕ from $arg(\mathcal{P})$ to non-negative integers such that, for every rule r of $\mathcal{P}$, every atom A occurring in the head of r, and every variable X occurring in a term A^i, if $\phi(A^0[i])$ is defined, then $body^+(r)$ contains an atom B such that X occurs in a term B^j, $\phi(B^0[j])$ is defined, and the following condition is satisfied*

$$\phi(A^0[i]) - \phi(B^0[j]) \geq d(X, A^i) - d(X, B^j)$$

where

$$d(X, X) = 0,$$
$$d(X, f(t_1, ..., t_m)) = 1 + \max_{i \,:\, t_i \, contains \, X} d(X, t_i).$$

The set of restricted arguments *of $\mathcal{P}$ is $AR(\mathcal{P}) = \{p[i] \mid p[i] \in arg(\mathcal{P}) \wedge \exists \phi$ s.t. $\phi(p[i])$ is defined$\}$. A program $\mathcal{P}$ is said to be* argument-restricted *iff $AR(\mathcal{P}) = arg(\mathcal{P})$.*

Example 1. Consider the following program:

$$\texttt{p(f(X))} \leftarrow \texttt{q(X)}.$$
$$\texttt{q(X)} \leftarrow \texttt{p(f(X))}.$$

The program is argument-restricted. An argument ranking ϕ which allows us to conclude that the program is argument-restricted is the following: $\phi(\texttt{p[1]}) = 1$ and $\phi(\texttt{q[1]}) = 0$.

Mapping-restricted programs *[1].* The mapping-restricted (m-restricted) criterion is defined over normal positive logic programs with function symbols but it can be easily extended to general disjunctive programs with negation.

Intuitively, this technique tries to find a set of argument/string pairs $p[i]/s$, called *mappings*, representing the fact that during the bottom-up evaluation of the program, argument $p[i]$ could take values whose structure, in terms of nesting of function symbols, is described by s. For instance, if $\texttt{p(f(g(c}_1\texttt{)), c}_2\texttt{)}$ is a ground atom derivable through the bottom-up evaluation of a program, then the mappings for its arguments are $\texttt{p[1]/fg}$ and $\texttt{p[2]}/\epsilon$, where ϵ denotes the empty string. This set of mappings is called *supported m-set* and is defined as follows.

Definition 2. *Given a program $\mathcal{P}$, a set of mappings $U_{\mathcal{P}}$ is a* supported m-set *of $\mathcal{P}$ if it can be built iteratively as follows:*

1. *$q[j]/\epsilon \in U_{\mathcal{P}}$ for every argument $q[j] \in arg(\mathcal{P})$ s.t. q is a base predicate symbol, and*
2. *for every rule $r \in \mathcal{P}$ and for every variable X in r, if all occurrences of X in the body of r have a mapping to a string s in $U_{\mathcal{P}}$, then all occurrences of X in the head of r also have a mapping to s in $U_{\mathcal{P}}$.*

An occurrence of a variable X in a term t_i of an atom $p(t_1, \ldots, t_n)$ has a mapping *to a string s in $U_{\mathcal{P}}$ if $p[i]/s \in U_{\mathcal{P}} \wedge t_i = X$ or $p[i]/gs \in U_{\mathcal{P}} \wedge t_i = g(...X...)$.*

Example 2. Consider the following program $\mathcal{P}_2$, where b is a base predicate:

```
p(X, X)          ← b(X).
q(f(X), f(X)) ← p(X, X).
p(f(X), X)     ← q(X, X).
```

A supported m-set of this program is $U_{\mathcal{P}_2} = \{\mathtt{b[1]}/\epsilon, \mathtt{p[1]}/\epsilon, \mathtt{p[2]}/\epsilon, \mathtt{q[1]/f}, \mathtt{q[2]/f}$, $\mathtt{p[1]/ff}$, $\mathtt{p[2]/f}\}$. Moreover, this set is minimal, i.e. there is no supported m-set U' of $\mathcal{P}_2$ such that $U' \subset U_{\mathcal{P}_2}$.

Definition 3. *A program $\mathcal{P}$ is* m-restricted *if $arg(\mathcal{P}) = MR(\mathcal{P})$, where $MR(\mathcal{P})$ denotes the set of* m-restricted arguments *$p[i]$ of $\mathcal{P}$. An argument $p[i]$ is* m-restricted *if the minimal supported m-set $U_{\mathcal{P}}$ of $\mathcal{P}$ contains a finite set of mappings $p[i]/s$.*

Example 3. Program $\mathcal{P}_2$ of Example 2 is m-restricted, since for every argument $p[i]$ in its minimal supported m-set there is a finite number of mappings of the form $p[i]/s$.

3 Iterated Termination Criteria

In this section, we present recently proposed termination criteria which, starting from a set of limited arguments defined through the application of a basic criterion, compute a possibly larger set of limited arguments.

***Safe programs** [8].* The first technique we present relies on a function, called *safe function*, which iteratively extends a given set of limited arguments. Its definition is based on the notion of activation graph.

The *activation graph* of a program $\mathcal{P}$, denoted $\Omega(\mathcal{P})$, is a directed graph whose nodes are the rules of $\mathcal{P}$, and there is an edge (r_i, r_j) in the graph iff r_i activates r_j, i.e. there exist two ground rules $r'_i \in ground(r_i)$, $r'_j \in ground(r_j)$ and a set of ground atoms I such that *(i)* $I \not\models r'_i$, *(ii)* $I \models r'_j$, and *(iii)* $I \cup head(r'_i) \not\models r'_j$. This intuitively means that if I does not satisfy r'_i, I satisfies r'_j, and $head(r'_i)$ is added to I to satisfy r'_i, this causes r'_j not to be satisfied anymore (and then to be "activated").

Definition 4. *Given a program $\mathcal{P}$ and a basic termination criterion W, the set of* W-safe arguments *$S\text{-}W(\mathcal{P})$ is computed by first setting $S\text{-}W(\mathcal{P}) = W(\mathcal{P})$ and next iteratively adding each argument $q[k]$ to $S\text{-}W(\mathcal{P})$ such that for all rules $r \in \mathcal{P}$ where q appears in the head,*
(i) r does not depend on a cycle of the activation graph $\Omega(\mathcal{P})$, or
(ii) for every head atom $q(t_1, ..., t_n)$ of r, every variable X appearing in t_k also appears in some W-safe argument of $body^+(r)$.

A program $\mathcal{P}$ is said to be W-safe *if $S\text{-}W(\mathcal{P}) = arg(\mathcal{P})$.*

The criterion obtained by combining a basic criterion W with the safe function is denoted by $S\text{-}W$.

Example 4. The following simple program $\mathcal{P}_4$ is not recognized as finitely-ground by all current basic termination criteria.

$$\begin{array}{l}\texttt{p(X,X)} \leftarrow \texttt{base(X)}.\\ \texttt{p(f(X),X)} \leftarrow \texttt{p(X,X)}.\\ \texttt{p(X,f(X))} \leftarrow \texttt{p(X,X)}.\end{array}$$

However, the bottom-up evaluation of this program always terminates. Indeed, $\mathcal{P}_4$ is *W-safe*, for every basic criterion W, since the activation graph of $\mathcal{P}_4$ does not contain any cycle.

Bounded Programs *[6].* The definition of bounded programs relies on the notion of *labelled argument graph.* This graph, denoted $\mathcal{G}_L(\mathcal{P})$, is derived from the argument graph by labelling edges as follows: for each pair of nodes $p[i], q[j] \in arg(\mathcal{P})$ and for every rule $r \in \mathcal{P}$ such that *(i)* an atom $p(t_1, ..., t_n)$ appears in $head(r)$, *(ii)* an atom $q(u_1, ..., u_m)$ appears in $body^+(r)$, *(iii)* terms t_i and u_j have a common variable X, there is an edge $(q[j], p[i], \langle \alpha, r, h, k \rangle)$, where h and k are natural numbers denoting the positions of $p(t_1, ..., t_n)$ in $head(r)$ and $q(u_1, ..., u_m)$ in $body^+(r)$, respectively[1], whereas $\alpha = \epsilon$ if $t_i = u_j$, $\alpha = f$ if $u_j = X$ and $t_i = f(..., X, ...)$, $\alpha = \bar{f}$ if $u_j = f(..., X, ...)$ and $t_i = X$. For the sake of simplicity and without loss of generality, we assume that if a variable X appears in two terms occurring in the head and the body of a rule, then only one of the two terms is a complex term and that the nesting level of complex terms is at most one.

Given a path $\rho = (a_1, b_1, \langle \alpha_1, r_1, h_1, k_1 \rangle), \ldots, (a_m, b_m, \langle \alpha_m,\ r_m, h_m, k_m \rangle)$, we define $\lambda_1(\rho) = \alpha_1 ... \alpha_m$, $\lambda_2(\rho) = r_1, ..., r_m$, and $\lambda_3(\rho) = \langle r_1, h_1, k_1 \rangle ... \langle r_m, h_m, k_m \rangle$. Given a cycle π consisting of n labelled edges $e_1, ..., e_n$, we can derive n different cyclic paths starting from each of the e_i's—we use $\tau(\pi)$ to denote the set of such cyclic paths.

Given two cycles π_1 and π_2, we write $\pi_1 \approx \pi_2$ iff $\exists \rho_1 \in \tau(\pi_1)$ and $\exists \rho_2 \in \tau(\pi_2)$ such that $\lambda_3(\rho_1) = \lambda_3(\rho_2)$. Given a program $\mathcal{P}$, we say that a cycle π in $\mathcal{G}_L(\mathcal{P})$ is *active* iff $\exists \rho \in \tau(\pi)$ such that $\lambda_2(\rho) = r_1, ..., r_m$ and $(r_1, r_2), ..., (r_{m-1}, r_m), (r_m, r_1)$ is a cyclic path in the activation graph $\Omega(\mathcal{P})$.

Given a program $\mathcal{P}$ and a path ρ in $\mathcal{G}_L(\mathcal{P})$, we denote with $\hat{\lambda}_1(\rho)$ the string obtained from $\lambda_1(\rho)$ by iteratively eliminating pairs of the form $\gamma\bar{\gamma}$ from the string until the resulting string cannot be further reduced.

Given a program $\mathcal{P}$, a cycle π in $\mathcal{G}_L(\mathcal{P})$ can be classified as follows. We say that π is i) *balanced* if $\exists \rho \in \tau(\pi)$ s.t. $\hat{\lambda}_1(\rho)$ is empty, ii) *growing* if $\exists \rho \in \tau(\pi)$ s.t. $\hat{\lambda}_1(\rho)$ does not contain a symbol of the form $\bar{\gamma}$, iii) *failing* otherwise.

Definition 5. *Given a program $\mathcal{P}$ and a basic termination criterion W, the set of* W-bounded arguments *B-$W(\mathcal{P})$ is computed by first setting B-$W(\mathcal{P}) = W(\mathcal{P})$ and next iteratively adding each argument $q[k]$ to B-$W(\mathcal{P})$ such that for each cycle π in $\mathcal{G}_L(\mathcal{P})$ on which $q[k]$ depends, at least one of the following conditions holds:*

[1] We assume that literals in the head (resp. body) are ordered with the first one being associated with 1, the second one with 2, etc.

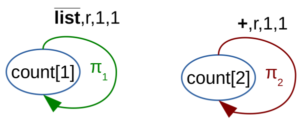

Fig. 1. Labelled argument graph.

1. *π is not active or is not growing;*
2. *π contains an edge $(s[j], p[i], \langle f, r, l_1, l_2\rangle)$ and, letting $p(t_1, ..., t_n)$ be the l_1-th atom in the head of r, for every variable X in t_i, there is an atom $b(u_1, ..., u_m)$ in body$^+(r)$ s.t. X appears in a term u_h and $b[h]$ is W-bounded;*
3. *there is a basic cycle π' in $\mathcal{G}_L(\mathcal{P})$ s.t. $\pi' \approx \pi$, π' is not balanced, and π' passes only through W-bounded arguments.*

A program $\mathcal{P}$ is said to be W-bounded *if B-$W(\mathcal{P}) = arg(\mathcal{P})$.*

The criterion obtained by combining a basic criterion W with the bounded criterion of the previous definition is denoted by B-W. A relevant aspect that distinguishes this technique from others is that this technique analyzes how groups of arguments are related with one another—this is illustrated in the following example.

Example 5. Consider the following logic program $\mathcal{P}_5$:

$$\begin{array}{ll} r_0 : & \texttt{count}([\texttt{a},\texttt{b},\texttt{c}], 0). \\ r \;\; : & \texttt{count}(\texttt{L}, \texttt{I}+1) \leftarrow \texttt{count}([\texttt{X}|\texttt{L}], \texttt{I}). \end{array}$$

The bottom-up evaluation of $\mathcal{P}_5$ terminates yielding the atoms $\texttt{count}([\texttt{a},\texttt{b},\texttt{c}], 0)$, $\texttt{count}([\texttt{b},\texttt{c}], 1)$, $\texttt{count}([\texttt{c}], 2)$, and $\texttt{count}([\,], 3)$. The query goal $\texttt{count}([\,], \texttt{L})$ can be used to retrieve the length L of list $[\texttt{a},\texttt{b},\texttt{c}]$.[2] In order to make the function symbols occurring in $\mathcal{P}_5$ more explicit, we rewrite $\mathcal{P}_5$ as follows:

$$\begin{array}{ll} r_0 : & \texttt{count}(\texttt{list}(\texttt{a}, \texttt{list}(\texttt{b}, \texttt{list}(\texttt{c}, \texttt{nil}))), 0). \\ r \;\; : & \texttt{count}(\texttt{L}, +(\texttt{I}, 1)) \leftarrow \texttt{count}(\texttt{list}(\texttt{X}, \texttt{L}), \texttt{I}). \end{array}$$

The labelled argument graph of the program above is depicted in Figure 1, where `list` and + denote the list constructor and the sum operators, respectively.

Basically, considering the argument-restricted technique as the basic criterion W, we can first establish that the argument count[1] is limited, that is, count[1] $\in B$-$AR(\mathcal{P}_5)$. Then, by analyzing the two cycles involving arguments count[1] and count[2] and using Condition 3 of Definition 5, it is possible to detect that also argument count[2] is limited, that is count[2] $\in B$-$AR(\mathcal{P}_5)$. Consequently, $\mathcal{P}_5$ is AR-bounded.

[2] Notice that $\mathcal{P}_5$ has been written so as to count the number of elements in a list when evaluated in a bottom-up fashion, and therefore differs from the classical formulation relying on a top-down evaluation strategy. However, programs relying on a top-down evaluation strategy can be rewritten into programs whose bottom-up evaluation gives the same result.

4 Adornment-based Technique

In this section, we give the basic idea of the rewriting technique introduced in [7]. This technique can be used in conjunction with current termination criteria, enabling us to detect more programs as finitely-ground. The technique takes a logic program $\mathcal{P}$ and transforms it into an adorned program $\mathcal{P}^{\mu}$ with the aim of applying termination criteria to $\mathcal{P}^{\mu}$ rather than $\mathcal{P}$. The transformation is sound in that if the adorned program satisfies a certain termination criterion, then the original program is finitely-ground. Importantly, as shown by the example below, applying termination criteria to adorned programs rather than the original ones strictly enlarges the class of programs recognized as finitely-ground. This technique is much more general than those used to deal with chase termination (see [5]).

Example 6. Consider the following program $\mathcal{P}_6$, where `base` is a base predicate symbol (defined by facts that are not shown here).

$$\begin{aligned} r_0 &: \ \mathtt{p(X,f(X))} \leftarrow \mathtt{base(X)}.\\ r_1 &: \ \mathtt{p(X,f(X))} \leftarrow \mathtt{p(Y,X)},\ \mathtt{base(Y)}.\\ r_2 &: \ \mathtt{p(X,Y)} \leftarrow \mathtt{p(f(X),f(Y))}. \end{aligned}$$

First, base predicate symbols are adorned with strings of ϵ's; thus, we get the adorned predicate symbol $\mathtt{base}^{\epsilon}$. This is used to adorn the body of r_0 so as to get

$$\rho_0 : \ \mathtt{p}^{\epsilon\mathtt{f}_1}(\mathtt{X},\mathtt{f(X)}) \leftarrow \mathtt{base}^{\epsilon}(\mathtt{X}).$$

from which we derive the new adorned predicate symbol $\mathtt{p}^{\epsilon\mathtt{f}_1}$, and the adornment definition $\mathtt{f}_1 = \mathtt{f}(\epsilon)$. Next, $\mathtt{p}^{\epsilon\mathtt{f}_1}$ and $\mathtt{base}^{\epsilon}$ are used to adorn the body of r_1 so as to get

$$\rho_1 : \ \mathtt{p}^{\mathtt{f}_1\mathtt{f}_2}(\mathtt{X},\mathtt{f(X)}) \leftarrow \mathtt{p}^{\epsilon\mathtt{f}_1}(\mathtt{Y},\mathtt{X}), \mathtt{base}^{\epsilon}(\mathtt{Y})$$

from which we derive the new adorned predicate symbol $\mathtt{p}^{\mathtt{f}_1\mathtt{f}_2}$, and the adornment definition $\mathtt{f}_2 = \mathtt{f}(\mathtt{f}_1)$. Intuitively, the body of ρ_1 is coherently adorned because `Y` is always associated with the same adornment symbol ϵ. Using the new adorned predicate symbol $\mathtt{p}^{\mathtt{f}_1\mathtt{f}_2}$, we can adorn rule r_2 and get

$$\rho_2 : \ \mathtt{p}^{\epsilon\mathtt{f}_1}(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^{\mathtt{f}_1\mathtt{f}_2}(\mathtt{f(X)},\mathtt{f(Y)}).$$

At this point, we are not able to generate new adorned rules (using the adorned predicate symbols generated so far) with coherently adorned bodies and the transformation terminates. In fact, $\mathtt{p}^{\mathtt{f}_1\mathtt{f}_2}(\mathtt{Y},\mathtt{X}), \mathtt{base}^{\epsilon}(\mathtt{Y})$ is not coherently adorned because the same variable `Y` is associated with both $\mathtt{f}_1$ and ϵ; moreover, $\mathtt{p}^{\epsilon\mathtt{f}_1}(\mathtt{f(X)},\mathtt{f(Y)})$ is not coherently adorned because `f(X)` does not comply with the (simple) term structure described by ϵ.

To determine termination of the bottom-up evaluation of $\mathcal{P}_6$, we can apply current termination criteria to $\mathcal{P}_6^{\mu} = \{\rho_0, \rho_1, \rho_2\}$ rather than $\mathcal{P}_6$.

It is worth noting that the rewriting technique ensures that if $\mathcal{P}_6^\mu$ is recognized as finitely-ground, so is $\mathcal{P}_6$. Notice also that both $\mathcal{P}_6$ and $\mathcal{P}_6^\mu$ are recursive, but while some termination criteria (e.g., the argument-restricted criterion) are able to detect that the bottom-up evaluation of $\mathcal{P}_6^\mu$ always terminates, none of the current termination criteria is able to realize that the bottom-up evaluation of $\mathcal{P}_6$ always terminates.

5 Conclusion

Recent years have witnessed an increasing interest in extending Datalog with function symbols. The main problem with the introduction of function symbols is that common inference tasks become undecidable. To cope with this issue, there have been several proposals of subclasses of Datalog with function symbols that impose limitations on the use of function symbols but guarantee decidability of common inference tasks. In this paper, we have discussed the main approaches recently proposed in the literature.

References

1. Marco Calautti, Sergio Greco, and Irina Trubitsyna. Detecting decidable classes of finitely ground logic programs with function symbols. In *Principles and Practice of Declarative Programming*, pages 239–250, 2013.
2. Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Computable functions in ASP: Theory and implementation. In *International Conference on Logic Programming*, pages 407–424, 2008.
3. Martin Gebser, Torsten Schaub, and Sven Thiele. Gringo: A new grounder for answer set programming. In *Logic Programming and Non-Monotonic Reasoning*, pages 266–271, 2007.
4. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *International Conference on Logic Programming/SLP*, pages 1070–1080, 1988.
5. Sergio Greco, Cristian Molinaro, and Francesca Spezzano. *Incomplete Data and Data Dependencies in Relational Databases.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
6. Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Bounded programs: A new decidable class of logic programs with function symbols. In *International Joint Conference on Artificial Intelligence*, pages 926–932, 2013.
7. Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Logic programming with function symbols: Checking termination of bottom-up evaluation through program adornments. *Theory and Practice of Logic Programming*, 13(4-5):737–752, 2013.
8. Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. On the termination of logic programs with function symbols. In *International Conference on Logic Programming (Technical Communications)*, pages 323–333, 2012.
9. Yuliya Lierler and Vladimir Lifschitz. One more decidable class of finitely ground programs. In *International Conference on Logic Programming*, pages 489–493, 2009.
10. Tommi Syrjanen. Omega-restricted logic programs. In *Logic Programming and Non-Monotonic Reasoning*, pages 267–279, 2001.