# Bounded Nondeterminism of Logic Programs[*]

Dino Pedreschi and Salvatore Ruggieri

*Dipartimento di Informatica, Università di Pisa*
*Via F. Buonarroti 2, 56125 Pisa, Italy*
e-mail: {`pedre`,`ruggieri`}`@di.unipi.it`

**Abstract**

We introduce the notion of bounded nondeterminism for logic programs and queries. A program and a query have bounded nondeterminism if there are finitely many refutations for them via any selection rule. We offer a declarative characterization of the class of programs and queries that have bounded nondeterminism by defining *bounded* programs and queries. The characterization is provided in terms of Herbrand interpretations and level mappings, in the style of existing characterizations of universal termination.

A direct application of the theoretical framework is concerned with the automatic generation of a terminating control. We present a transformational approach that given a bounded program and a bounded query yields a terminating program and query with the same set of refutations.

Concerning the issue of automating the approach, by means of an example we sketch how an automatic method for proving left termination can be adapted to the purpose of inferring boundedness. Such an adaption reveals that the method does not scale up to medium/large sized programs due to scarce modularity of the required proof obligations. We provide then a modular refinement of boundedness for the significant class of well-moded programs and queries.

**Keywords:** *Logic programming, universal termination, bounded nondeterminism, strong termination.*

[*]A preliminary version of this paper appeared in *Proceedings of the 16th International Conference on Logic Programming,* [24].

## 1 From Universal Termination to Bounded Nondeterminism

Logic programming is a declarative paradigm, where nondeterministic specifications can be directly executed as programs and the generation of a complete control is demanded to the underlying system. By *a complete control,* it is usually meant a selection rule $s$ such that every logical consequence of a program and a query has a refutation via $s$. By Strong Completeness of SLD-resolution [1], any selection rule is complete in this sense. However, a stronger form of completeness is usually required for a selection rule to be useful in programming, namely termination. A *terminating control* for a program $P$ and a query $Q$ is a selection rule $s$ such that every SLD-derivation of $P$ and $Q$ via $s$ is finite. In the last decade, several classes of programs and queries that have a terminating control have been characterized declaratively. We refer the reader to [10, 25] for surveys of such characterizations, including:

- *recurrent* programs and queries, introduced by Bezem [4], for which every selection rule is a complete control (*strong termination*);
- *acceptable* programs and queries, introduced by Apt and Pedreschi [3], for which the leftmost selection rule is a complete control (*left termination*);
- *fair-bounded* programs and queries, introduced by Ruggieri [28], for which fair-selection rules are a complete control ($\exists$-*termination*).

In particular, fair-boundedness precisely characterize the class of programs and queries for which a terminating control exists, i.e. if a terminating control exists for a program and a query then any fair-selection rule is a terminating control them. In general, however, a terminating control in the sense above may not exist.

**Example 1.1** The `ODDEVEN` program:

```
even(s(X)) ← odd(X).
even(0).

odd(s(X)) ← even(X).
```

defines the `even` and `odd` predicates, with the usual intuitive meaning. The query `even(X),odd(X)` is intended to check whether the program defines a number that is both even and odd.

Even though the program is recurrent in the sense of Bezem [4] (which implies that every ground query strongly terminates), `ODDEVEN` and the (non ground) query above do not have a terminating control, i.e. they have an infinite derivation via any selection rule. Notice, however, that they have no refutation. □

Moreover, due to performance reasons, very few systems adopt fair selection rules.

**Example 1.2** This variant of the well-known program `PERMUTATION` checks whether two lists are permutations of each other.

```
(p1)    perm([], []).
(p2)    perm([X|Xs], Ys) ←
           delete(X, Ys, Zs),
           perm(Xs, Zs).

(d1)    delete(X, [X|Y], Y).
(d2)    delete(X, [H|Y], [H|Z]) ←
           delete(X, Y, Z).
```

`PERMUTATION` and the query `perm([a, b], Ys)` have a terminating control, e.g., the rightmost or any fair selection rule. However, it may be the case that the underlying system does not support that selection rule. Thus, we are still left with the termination problem. Notice, however, that the program and the query above have finitely many SLD-refutations via any selection rule. □

In this paper, we introduce and investigate the notion of *bounded nondeterminism* of (definite) logic programs and queries. We say that a program $P$ and a query $Q$ have bounded nondeterminism if they have finitely many refutations w.r.t. any selection rule. For instance, the programs and queries of the examples above have bounded nondeterminism. The relation between this definition and the notion of universal termination is tight. In fact, if $P$ and $Q$ have a terminating control, then $P$ and $Q$ have bounded nondeterminism. Conversely, if $P$ and $Q$ have bounded nondeterminism then there exists an upper bound to the length of the SLD-refutations of $P$ and $Q$. If such an upper bound is known, then we can transform $P$ and $Q$ into an equivalent program and query that strongly terminate, i.e. such that any selection rule is a terminating control for them.

We will offer a declarative characterization of programs and queries that have bounded nondeterminism, by introducing the class of *bounded* programs and queries. The definition is given in terms of level mappings and Herbrand interpretations, in the style of the already mentioned classes of terminating programs [3, 4, 28]. Notably, the definition is *purely declarative* in the sense that neither any procedural notion is needed in order to prove a program bounded, nor the definition reflects some fixed ordering of the atoms.

A direct application of the theoretical framework is concerned with the automatic generation of a terminating control for a given program and query. We present a transformational approach that given a bounded program and a bounded query yields a terminating program and query with the same set of refutations. The transformational approach adds a counter to atoms that allows for cutting derivations at an appropriate length, which is an upper bound for the length of refutations.

We discuss the issue of inferring boundedness of programs and queries showing that it is an undecidable problem. However, we argue that existing (sufficient) methods for inferring left termination can be adapted to infer bounded nondeterminism. By means of an example, we outline how the method of Decorte, De Schreye and Vandecasteele [13], originally designed for inferring left termination, can be directly adapted to infer boundedness of programs and queries. Unfortunately, such an adaption reveals that the method does not scale up to medium/large sized programs due to scarce modularity of proof obligations. We provide then a modular refinement of boundedness that is sound and complete for the significant class of well-moded programs and queries.

### Plan of the paper

In the next section we recall some background and notation on logic programs, selection rules, level mappings and termination. In section 3 we introduce the notion of bounded nondeterminism and characterize it by means of bounded programs and queries. The characterization is shown to be sound and complete in section 4. In section 5, we present a method for transforming bounded programs into terminating ones. Also, we discuss the issue of automatically inferring boundedness in section 6. Related work and summarization of the contributions conclude the paper.

## 2 Background and Notation

We use the standard notation of Apt [1], when not otherwise specified. In particular, throughout this paper we consider a fixed language $L$ in which programs and queries are written. All the results are *parametric* with respect to $L$, provided that $L$ is rich enough to contain the symbols of the programs and queries under consideration. We denote with $U_L$ and $B_L$ the Herbrand universe and the Herbrand base on $L$, and with $M_P^L$ the least Herbrand model of program $P$ on $L$. Also, $ground_L(P)$ and $ground_L(Q)$ represent the set of ground instances on $L$ of program $P$ and query $Q$ respectively. We use typewriter font for logical variables, e.g. `X`, upper case letters for arbitrary terms, e.g. $Xs$, and lower case letters for ground terms, e.g. $x$. $N$ is the set of naturals. For an atom $A$, $rel(A)$ denotes the predicate symbol of $A$. We write $p \simeq q$ if $p$ and $q$ are mutually recursive predicates (w.r.t. a given program).

### 2.1 Selection Rules

Let $INIT$ be the set of initial fragments of SLD-derivations in which the last query is non-empty. The standard definition of *selection rule* is as follows: a selection rule is a function that, when applied to an element in $INIT$, yields an occurrence of an atom in its last query [1].

The classic *leftmost* selection rule always selects the leftmost atom in the last query of an element in $INIT$. The *rightmost* selection rule always selects the rightmost atom. A selection rule $s$ is *fair* if for every SLD-derivation $\xi$ via $s$ either $\xi$ is finite or for every atom $A$ in $\xi$, (some further instantiated version of) $A$ is eventually selected.

## 2.2 Norms and Level Mappings

Most characterizations of terminating program and queries make use of the notions of norm and level mapping [10, 25]. Here, we recall the definition of [26].

**Definition 2.1** A *norm* is a function $\|.\| : U_L \rightarrow N^{\infty}$, where $N^{\infty} = N \cup \{\infty\}$.

A *level mapping* is a function $|.| : B_L \rightarrow N^{\infty}$. For a ground atom $A$, $|A|$ is called the level of $A$. □

Level mappings are used to measure the "size" of a query and show that this size decreases along a derivation, hence showing termination. Level mappings are usually defined as functions on norms of predicate arguments. One commonly used norm is the term size, defined as:

$$\begin{array}{ll} size(f(t_1, \ldots, t_n)) = 1 + size(t_1) + \ldots + size(t_n) & \text{if } n > 0 \\ size(a) = 0 & \text{if } a \text{ is a constant.} \end{array}$$

Intuitively, the size $size(t)$ of a ground term $t$ is the number of function symbols occurring in it, excluding constants. Another widely used norm is the list-length, defined as follows:

$$\begin{array}{ll} |f(\ldots)| = 0 & \text{if } f \neq \texttt{[.|.]} \\ |\texttt{[}x\,|\,t\texttt{]}| = 1 + |t|. & \text{otherwise.} \end{array}$$

In particular, the list-length of a ground list $\texttt{[}t_1\texttt{, ..., }t_n\texttt{]}$ is $n$.

In contrast to the more standard definition (see e.g. [10, 25]), Definition 2.1 includes $\infty$ in the codomain of level mappings. The rationale is to use $\infty$ as a means to model uninteresting instances of program clauses and queries. The need for reasoning on a subset of $B_L$ is motivated by the fact that logic programs are untyped, and then queries may have instances that in the intended interpretation of the programmer are unintended, or ill-typed. Another reason for introducing $\infty$ is the fact that a program may have some desired property (e.g., bounded non-determinism or termination) for a proper subset of (ground) queries only. In this case, $\infty$ allows for modelling the absence of the desired property. We report from [26] the extension of the $>$ order on naturals to a relation $\rhd$ on $N^{\infty}$.

**Definition 2.2** We define the relation $n \rhd m$ for $n, m \in N^{\infty}$ as follows:

$$n \rhd m \quad \text{iff} \quad n = \infty \text{ or } n > m.$$

We write $n \unrhd m$ iff $n \rhd m$ or $n = m$. □

With this definition, $\infty \rhd \infty$ and $\infty \rhd n$ for any $n \in N$ hold.

## 2.3 Universal Termination

In general terms, the problem of universal termination of a program $P$ and a query $Q$ w.r.t. a set of admissible selection rules consists of showing that every rule in the set is a terminating control for $P$ and $Q$.

**Definition 2.3** A program $P$ and a query $Q$ *universally terminate* w.r.t. a set of selection rules $\mathcal{S}$ if every SLD-derivation of $P$ and $Q$ via any selection rule from $\mathcal{S}$ is finite. $\square$

Note that, since SLD-trees are finitely branching, by König's Lemma, "every SLD-derivation for $P$ and $Q$ via a selection rule $s$ is finite" is equivalent to stating that the SLD-tree of $P$ and $Q$ via $s$ is finite.

We refer the reader to [10, 25] for surveys of proposed declarative characterizations of universal terminating programs and queries. Among the others, we recall the class of recurrent programs and queries, introduced by Bezem [4], that models universal termination w.r.t. all selection rules, a notion called *strong termination.* Intuitively, a program is recurrent if for every ground instance of a clause, the level of the body-atoms is smaller than the level of the head.

**Definition 2.4** Let $|.| : B_L \rightarrow N$ be a level mapping into naturals. A program $P$ is *recurrent by* $|.|$ iff for every $A \leftarrow B_1, \dots, B_n$ in $ground_L(P)$ :

$$\text{for } i \in [1, n] \quad |A| \; > |B_i|.$$

A query $Q$ is *recurrent by* $|.|$ iff there exists $k \in N$ such that for every $(A_1, \dots, A_n) \in ground_L(Q)$ :

$$\text{for } i \in [1, n] \quad k > |A_i|.$$

$\square$

Termination soundness of recurrent programs is summarized in the next theorem.

**Theorem 2.5** ([4]) *If a program $P$ and a query $Q$ are both recurrent by a level mapping $|.|$, then they universally terminate w.r.t. the set of all selection rules.* $\square$

A form of completeness holds as well, yet in a restricted sense due to the use of level mappings into naturals, which are functions that must specify a value for every ground atom. As pointed out in [25], by using level mappings with $\infty$ in the codomain and the $\rhd$ relation instead of $>$, one can achieve full completeness. However, in this paper we will be only interested in termination soundness.

# 3 Bounded Programs

## 3.1 Bounded Nondeterminism

First, we formally introduce the notion of bounded nondeterminism of logic programs and queries, which turns out to be a weakening of universal termination.

**Definition 3.1** A program $P$ and a query $Q$ have bounded nondeterminism if for every selection rule $s$ there are finitely many SLD-refutations of $P$ and $Q$ via $s$. $\square$

It is readily checked that if $P$ and $Q$ universally terminate w.r.t. any non-empty set of selection rules then they have bounded nondeterminism. As a consequence, any existing (possibly automatic) characterization of universally terminating programs and queries is a sufficient method to show bounded nondeterminism. On the other side, Example 1.1 shows that the class of programs and queries with bounded nondeterminism strictly include any universally terminating class of programs and queries.

Also, note that, by Strong Completeness of SLD-resolution [1], the definition above is equivalent to require that *there exists* a selection rule $s$ such that there are finitely many SLD-refutations of $P$ and $Q$ via $s$.

## 3.2 Bounded Programs

Let us now define a class of programs and queries that will be shown to be a sound and complete characterization of bounded nondeterminism.

**Definition 3.2** Let $|.|$ be a level mapping, and $I$ a Herbrand interpretation. A logic program $P$ is *bounded by* $|.|$ *and* $I$ iff $I$ is a model of $P$ such that for every $A \leftarrow B_1, \dots, B_n$ in $ground_L(P)$ :

$$I \models B_1, \dots, B_n \quad \textit{implies} \quad \text{for } i \in [1,n] \quad |A| \ \triangleright \ |B_i|.$$

$\square$

Intuitively, the definition of boundedness only requires the decreasing of the level mapping when the body atoms are true in some model of the program, i.e. they might have a refutation. Notice that the notion of boundedness is *purely declarative*, in the sense that neither any procedural notion is needed in proof obligations[1], nor the definition reflects some fixed ordering of the atoms. Also, observe that the well-studied classes of recurrent [4], acceptable [3] and fair-bounded [28] logic programs are readily checked to be subclasses of bounded programs (this is the declarative counterpart of the fact that if a program and a query have a terminating control then they have bounded nondeterminism). The next definition extends boundedness to queries.

[1]In the program verification terminology, the term *proof obligation* is a synonymous for *any condition required to be shown.*

**Definition 3.3** Let $|.|$ be a level mapping, and $I$ a Herbrand interpretation. A query $Q$ is *bounded by* $|.|$ *and* $I$ iff there exists $k \in N$ such that for every $(A_1 , \ldots , A_n ) \in ground_L(Q)$ :

$$I \models A_1, \ldots, A_n \quad \textit{implies} \quad \text{for } i \in [1, n] \quad k \ \rhd \ |A_i|.$$

□

In the above definition, the proof obligations for a query $Q$ are derived from those for the program $\{\texttt{p} \leftarrow Q\}$, where `p` is a fresh predicate symbol such that $|\texttt{p}| \in N$. This is justified by the fact that $P$ and $Q$ have bounded nondeterminism iff the program $P \cup \{\texttt{p} \leftarrow Q\}$ and the query `p` have it. Also, it is quite intuitive that $|\texttt{p}|$ must be in $N$, since it plays the role of an upper bound. Therefore, in Definition 3.3, $k$ is the equivalent of $|\texttt{p}|$ in the proof obligations of clause $\texttt{p} \leftarrow Q$.

**Example 3.4** Consider again the `ODDEVEN` program. We write $\texttt{s}^n\texttt{(0)}$ as a shorthand for `s(s(...s(0)...))`, where `s` is repeated $n \in N$ times. It is readily checked that `ODDEVEN` is bounded by defining:

$$|\texttt{even(}t\texttt{)}| = |\texttt{odd(}t\texttt{)}| = \textit{size}(t)$$
$$I = \{\ \texttt{even(s}^{2 \cdot i}\texttt{(0))}, \texttt{odd(s}^{2 \cdot i+1}\texttt{(0))} \mid i \geq 0\ \}.$$

The query `even(X), odd(X)` is bounded by $|.|$ and $I$. In fact, since no ground instance of this query is true in $I$, Definition 3.3 imposes no requirement. □

More generally, for a query that has no instance in a model of the program (it is *unsolvable*), the value $k$ in Definition 3.2 can be chosen as 0. An automatic method to check whether a query (at a node of an SLD-tree) is unsolvable has been proposed in [7]. Of course, the example is somewhat a limit case, since one does not even need to run a query if it has been shown to be unsolvable. The next example shows a program and a *satisfiable* query that do not terminate via any selection rule but have bounded nondeterminism.

**Example 3.5** We now define the predicate `all` such that the query $\texttt{all(}n_0\texttt{,}n_1\texttt{,Xs)}$ collects in `Xs` the answers of a query $\texttt{q(}m\texttt{,A)}$ for values $m$ ranging from $n_0$ to $n_1$.

```
all(N,N,[A]) ← q(N,A).
all(N,N1,[A|As]) ← q(N,A), all(s(N),N1,As).
q(Y, Y). %just as an example
```

For any selection rule $s$, the program and the query `all(0,s(s(0)),As)` have an infinite derivation via $s$. However, the query is satisfiable, and in fact there exists a refutation via $s$ with computed answer `As = [0,s(0),s(s(0))]`. The program and the query are bounded by defining:

$$\begin{aligned} |\texttt{all(}n\texttt{, }m\texttt{, }x\texttt{)}| &= \textit{max}\{\textit{size}(m) - \textit{size}(n), 0\} + 1 \\ |\texttt{q(}x\texttt{,}y\texttt{)}| &= 0 \\ I &= \{\ \texttt{all(}n\texttt{, }m\texttt{, }x\texttt{)} \mid \textit{size}(n) \leq \textit{size}(m)\ \} \cup \{\texttt{q(}x\texttt{,}y\texttt{)}\}. \end{aligned}$$

The only non-trivial proof obligation concerns the second clause. Let

$$\texttt{all(}n\texttt{,}n'\texttt{,[}a\texttt{|}as\texttt{])} \leftarrow \ \texttt{q(}n\texttt{,}a\texttt{), all(s(}n\texttt{),}n'\texttt{,}as\texttt{).}$$

be a ground instance of that clause such that $I \models \texttt{q(}n\texttt{,}a\texttt{), all(s(}n\texttt{),}n'\texttt{,}as\texttt{)}$. $|\texttt{all(}n\texttt{, }n'\texttt{, [}a\texttt{|}as\texttt{])}| \rhd 0 = |\texttt{q(}n\texttt{,}a\texttt{)}|$ shows the decreasing from the head to the first body atom. Moreover, $I \models \texttt{all(s(}n\texttt{),}n'\texttt{,}as\texttt{)}$ means $size(\texttt{s(}n\texttt{)}) = size(n) + 1 \leq size(n')$, which implies:

$$\begin{array}{rcl} |\texttt{all(}n\texttt{, }n'\texttt{, [}a\texttt{|}as\texttt{])}| & = & size(n') - size(n) + 1 \\ & \rhd & size(n') - size(\texttt{s(}n\texttt{)}) + 1 = |\texttt{all(s(}n\texttt{),}n'\texttt{,}as\texttt{)}|, \end{array}$$

which shows the decreasing from the head to the second body atom, and $size(n) \leq size(n')$, i.e. $\texttt{all(}n\texttt{, }n'\texttt{, [}a\texttt{|}as\texttt{])} \in I$, which, in turn, shows that $I$ is a model of the instance. □

**Example 3.6** Consider again the `PERMUTATION` program and the query `perm([a, b], Ys)`. Let us show they are bounded by $|.|$ and $I$, where:

$$|\texttt{perm(}xs\texttt{, }ys\texttt{)}| = |xs|$$
$$|\texttt{delete(}x\texttt{, }xs\texttt{, }ys\texttt{)}| = |ys|,$$

$$\begin{array}{rcl} I & = & \{\ \texttt{perm(}xs\texttt{, }ys\texttt{)} \mid |xs| = |ys|\ \} \cup \\ & & \{\ \texttt{delete(}x\texttt{, }xs\texttt{, }ys\texttt{)} \mid |xs| = |ys| + 1\ \}. \end{array}$$

We recall that $|t|$ is the list-length of the ground term $t$. The only non-trivial proof obligations are those regarding clause *(p2)*. Let

$$\texttt{perm([}x\texttt{|}xs\texttt{], }ys\texttt{)} \ \leftarrow \ \texttt{delete(}x\texttt{, }ys\texttt{, }zs\texttt{), perm(}xs\texttt{, }zs\texttt{).}$$

be a ground instance of that clause. If the body is true in $I$, then $|xs| = |zs|$ and $|ys| = |zs| + 1$. This implies:

*(a)*
$$\begin{array}{rcl} |\texttt{perm([}x\texttt{|}xs\texttt{], }ys\texttt{)}| & = & |xs| + 1 \\ & \rhd & \quad \{\ |xs| = |zs|\ \} \\ & & |zs| = |\texttt{delete(}x\texttt{, }ys\texttt{, }zs\texttt{)}|, \end{array}$$

*(b)*
$$\begin{array}{rcl} |\texttt{perm([}x\texttt{|}xs\texttt{], }ys\texttt{)}| & = & |xs| + 1 \\ & \rhd & |xs| = |\texttt{perm(}xs\texttt{, }zs\texttt{)}|. \end{array}$$

*(c)*
$$\begin{array}{rl} \texttt{perm([}x\texttt{|}xs\texttt{], }ys\texttt{)} \in I & \textit{iff} \quad |xs| + 1 = |ys| \\ & \textit{iff} \quad \quad \{\ |ys| = |zs| + 1\ \} \\ & \quad\quad |xs| = |zs| \\ & \textit{iff} \quad \quad \{\ |xs| = |zs|\ \} \\ & \quad\quad \textit{true}. \end{array}$$

*(a,b)* show the decreasing of the level mapping as required in Definition 3.2, while *(c)* shows that $I$ is a model of the clause.

Finally, by fixing $k = |$ `perm([a, b], Ys)` $| + 1 = |$ `[a,b]` $| + 1 = 3$, the proof obligations of Definition 3.3 are satisfied.

More generally, every query `perm(`$Xs$`, Ys)` is bounded by $|.|$ and $I$ when $Xs$ is a list. Also, this is the case for queries `perm(Ys, `$Xs$`)` when $Xs$ is a list. □

Next, we report an example showing the role of $\infty$, which allows for excluding some unintended atoms from the proof of boundedness. Such atoms include badly-typed ones, such as `perm(a, b)`, and, more importantly, atoms for which bounded nondeterminism does not hold.

**Example 3.7** The following program `TRANSP` models the computation of the transitive closure of a graph.

```
trans(X,Y,E) ←
   member(arc(X,Y),E).

trans(X,Y,E) ←
   trans(Z,Y,E),
   member(arc(X,Z),E).

member(X,[X|Xs]).

member(X,[Y|Xs]) ←
   member(X,Xs).
```

In the intended meaning of the program, `trans(`$x$`,`$y$`,`$e$`)` succeeds iff $x \leadsto_e y$, i.e. if `arc(`$x$`,`$y$`)` is in the transitive closure of a direct acyclic graph (DAG) $e$, which is represented as a list of arcs. It is readily checked that if $e$ is a graph that contains a cycle, infinitely many refutations may occur. On the contrary, if $e$ is acyclic the program and the query have bounded nondeterminism. Let us show that `TRANSP` is bounded. We define:

$$\begin{aligned}
|\texttt{trans(}x\texttt{,}y\texttt{,}e\texttt{)}| &= \begin{cases} |e| + 1 + Card\{v \mid x \leadsto_e v\} & \text{if } e \text{ is a DAG} \\ \infty & \text{otherwise} \end{cases} \\
|\texttt{member(}x\texttt{,}e\texttt{)}| &= |e| \\
I &= \{\texttt{trans(}x\texttt{,}y\texttt{,}e\texttt{)} \mid \text{true}\} \cup \\
&\quad\ \{\texttt{member(}x\texttt{,}e\texttt{)} \mid x \text{ is in the list } e\}.
\end{aligned}$$

where $Card$ is the set cardinality operator. It is easy to check that `TRANSP` is bounded by $|.|$ and $I$. In particular, consider a ground instance of the second clause:

$$\texttt{trans(}x\texttt{,}y\texttt{,}e\texttt{)} \leftarrow \texttt{trans(}z\texttt{,}y\texttt{,}e\texttt{)},\texttt{member(arc(}x\texttt{,}z\texttt{),}e\texttt{)}.$$

It is immediate to see that $I$ is a model of it. In addition, we have the proof obligations:

$$\begin{array}{lll} \textit{(i)} & \texttt{arc(}x\texttt{,}z\texttt{)} \text{ is in } e \;\Rightarrow & |\texttt{trans(}x\texttt{,}y\texttt{,}e\texttt{)}| \rhd |\texttt{trans(}z\texttt{,}y\texttt{,}e\texttt{)}| \\ \textit{(ii)} & & |\texttt{trans(}x\texttt{,}y\texttt{,}e\texttt{)}| \rhd |\texttt{member(arc(}x\texttt{,}z\texttt{),}e\texttt{)}|. \end{array}$$

The second one is easy to show since $|\texttt{trans(}x\texttt{,}y\texttt{,}e\texttt{)}| \rhd |e|$. Considering the first one, we distinguish two cases. If $e$ is not a DAG, the conclusion is immediate. Otherwise, $\texttt{arc(}x\texttt{,}z\texttt{)}$ in $e$ implies $Card\{v \mid x \leadsto_e v\} > Card\{v \mid z \leadsto_e v\}$, and so:

$$\begin{array}{rcl} |\texttt{trans(}x\texttt{,}y\texttt{,}e\texttt{)}| & = & |e| + 1 + Card\{v \mid x \leadsto_e v\} \\ & \rhd & |e| + 1 + Card\{v \mid z \leadsto_e v\} = |\texttt{trans(}z\texttt{,}y\texttt{,}e\texttt{)}|. \end{array}$$

Finally, observe that for a DAG $e$, the queries $\texttt{trans(}x\texttt{,Y,}e\texttt{)}$ and $\texttt{trans(X,Y,}e\texttt{)}$ are bounded by $|.|$ and $I$. The first one is intended to compute all nodes $y$ such that $x \leadsto_e y$, while the second one computes the binary relation $\leadsto_e$. Therefore, the `TRANSP` program and those queries have bounded nondeterminism. As a final observation, note that they do not left-terminate (unless body atoms in the recursive clause of `trans` are switched). □

## 4 Soundness and Completeness

### 4.1 Soundness

The notion of boundedness is persistent along SLD-derivations.

**Lemma 4.1 (Persistency)** *Let $P$ be a program and $Q$ a query both bounded by $|.|$ and $I$. Every SLD-resolvent $Q'$ of $P$ and $Q$ is bounded by $|.|$ and $I$.*

*Proof.* Let $k \in N$ be such that Definition 3.3 of boundedness of $Q$ is satisfied.

Let $\theta$ be the mgu of the selected atom in $Q$ and the head of the (renamed apart) input clause $c$. Assume that $Q\theta = A_1, \dots, A_n$, and that $c\theta = A_k \leftarrow B_1, \dots, B_m$. The SLD-resolvent $Q'$ is then:

$$A_1, \dots, A_{k-1}, B_1, \dots, B_m, A_{k+1}, \dots, A_n.$$

Let us show the proof obligation of Definition 3.3 for $Q'$.

Let $Q'\gamma = A'_1, \dots, A'_{k-1}, B'_1, \dots, B'_m, A'_{k+1}, \dots, A'_n$ be a ground instance of $Q'$. For any $A'_k$ ground instance of $A_k\gamma$, it turns out that $A'_1, \dots, A'_n$ is a ground instance of $Q\theta$ (and then of $Q$), and $A'_k \leftarrow B'_1, \dots, B'_m$ is a ground instance of $c\theta$ (and then of $c$). Suppose that $I \models Q'\gamma$. Then $I \models B'_1, \dots, B'_m$ and, since $I$ is a model of $P$, $I \models A'_1, \dots, A'_n$. By Definition 3.3, this implies $k \rhd |A'_i|$ for every $i \in [1, n]$. Moreover, since $I \models B'_1, \dots, B'_m$ and $P$ is bounded, we have that $k \rhd |A'_k| \rhd |B'_i|$ for $i \in [1, m]$. □

To show that bounded programs and queries have bounded nondeterminism, we follow an approach that relies on properties of multisets over well-founded orderings (see e.g., [14]).

Let $(W, >)$ be a pair where $> \subseteq W \times W$ is an irreflexive and transitive relation (i.e., a *strict partial order*). $(W, >)$ is well-founded if there is no infinite descending chain, i.e. no infinite sequence $a_0, \ldots, a_n, \ldots$ such that $a_0 > a_1 > \ldots > a_n > \ldots$

A multiset on $(W, >)$ is a pair $(bag(W), \succ_m)$ where $bag(W)$ is the set of unordered sequences of elements from $W$, which we call multisets. We denote a multiset of elements $a_1 , \ldots , a_n$ by $bag(a_1 , \ldots , a_n )$. Relation $\succ_m$ is induced by $>$ as the transitive closure of $\succ$, where $x \succ y$ if $y$ can be obtained from $x$ by replacing an element $a$ of $x$ by finitely many (possibly zero) elements $b \in W$ such that $a > b$. Also, we denote by $\succeq_m$ the reflexive closure of $\succ_m$ .

In particular, we associate a multiset over naturals to bounded queries.

**Definition 4.2** Let $Q = A_1 , \ldots , A_n$ be a query bounded by $|.|$ and $I$. We define the sets $|Q|^I_i$ for $i \in [1, n]$ as follows:

$$|Q|^I_i = \{|A'_i| \mid (A'_1 , \ldots , A'_n ) \in ground_L(Q) \ \wedge I \models A'_1 , \ldots , A'_n \ \}.$$

We define $|Q|^I$ as the finite multiset

$$|Q|^I = bag(max|Q|^I_1, \ldots, max|Q|^I_n),$$

if $I \models \exists \ (A_1 , \ldots , A_n )$, and $|Q|^I = bag()$ if $I \not\models \exists \ (A_1 , \ldots , A_n )$. □

Note that for $i \in [1, n]$, $max|Q|^I_i$ exists since $Q$ is assumed to be bounded by $|.|$ and $I$. The next Lemma states that the multiset associated to queries decreases along refutations. This derives from the fact that every query in a refutation is satisfiable in any model of the program.

**Lemma 4.3** *Let $P$ be a program and $Q$ a query both bounded by $|.|$ and $I$. For every SLD-resolvent $Q'$ of $P$ and $Q$, we have:*

*(i)* $|Q|^I \succeq_m |Q'|^I$, *and*

*(ii) if* $I \models \exists \ Q'$ *then* $|Q|^I \succ_m |Q'|^I$.

*Proof.* Let $\theta$ be the mgu of the selected atom in $Q$ and the head of the (renamed apart) input clause $c$. Assume that $Q\theta = A_1 , \ldots , A_n$ , and that $c\theta = A_k \leftarrow B_1, \ldots, B_m$. Then $Q'$ is

$$A_1, \ldots, A_{k-1}, B_1 , \ldots , B_m , A_{k+1}, \ldots, A_n.$$

First, suppose that $I \not\models \exists \ Q'$. Then we have to show only *(i)*, which is immediate by observing that $|Q|^I \succeq_m bag() = \ |Q'|^I$.

Suppose now that $I \models \exists\, Q'$. This implies that for every $i \in [1, n+m-1]$:

$$|Q'|_i^I \neq \emptyset. \tag{1}$$

Let $Q'\gamma = A'_1, \ldots, A'_{k-1}, B'_1, \ldots, B'_m, A'_{k+1}, \ldots, A'_n$ be a ground instance of $Q'$. For any $A'_k$ ground instance of $A_k\gamma$, it turns out that $A'_1, \ldots, A'_n$ is a ground instance of $Q\theta$ (and then of $Q$), and $A'_k \leftarrow B'_1, \ldots, B'_m$ is a ground instance of $c\theta$ (and then of $c$). As a consequence for every $i \in [1, n], i \neq k$, we have $|A'_i| \in |Q|_i^I$. This implies:

$$\text{for } i \in [1, k-1] \qquad \mathit{max}|Q'|_i^I \leq \mathit{max}|Q|_i^I, \tag{2}$$

$$\text{for } i \in [k+1, n] \qquad \mathit{max}|Q'|_{i+m-1}^I \leq \mathit{max}|Q|_i^I. \tag{3}$$

Moreover, since $I \models B'_1, \ldots, B'_m$ and $P$ is bounded, we have that $|A'_k| \rhd |B'_i|$ for every $i \in [1, m]$. By the assumption that $Q$ is bounded, we have that $\mathit{max}|Q|_k^I$ exists, and then $\mathit{max}|Q|_k^I \geq |A'_k| > |B'_i|$ for every $i \in [1, m]$. Summarizing:

$$\text{for } i \in [1, m] \qquad \forall\, x \in\ |Q'|_{k+i-1}^I\ x < \mathit{max}|Q|_k^I. \tag{4}$$

In conclusion, we calculate:

$$\begin{array}{lll}
|Q|^I & = & \{ \text{ Definition 3.3 } \} \\
 & & \mathit{bag}(\mathit{max}|Q|_1^I, \ldots, \mathit{max}|Q|_k^I, \ldots, \mathit{max}|Q\theta|_n^I) \\
 & \succ_m & \{ \ (1-4) \text{ and the fact that} \\
 & & \quad \forall\, x \in S \neq \emptyset\ x < y \ \mathit{implies\ max}\ S\ <\ y \ \} \\
 & & \mathit{bag}(\mathit{max}|Q'|_1^I, \ldots, \mathit{max}|Q'|_{n+m-1}^I) \\
 & = & \{ \text{ Definition 3.3 } \} \\
 & & |Q'|^I
\end{array}$$

which implies *(i-ii)*. □

Next, we introduce a sub-relation of $\succ_m$, which is parametric in a natural number $v$. This value denotes the maximum number of elements that may replace an element in a bag. In our context, $v$ will denote the maximum number of atoms in the body of a clause of a given program.

**Definition 4.4** Given a pair $(W, >)$ and $v \in N$, we define $\succ_m^v \subseteq \mathit{bag}(W) \times \mathit{bag}(W)$ as the transitive closure of the relation $\succ^v \subseteq \mathit{bag}(W) \times \mathit{bag}(W)$ in which $x \succ^v y$ if $y$ can be obtained from $x$ by replacing an element $a$ of $x$ by at most $v$ (possibly zero) elements $b \in N$ such that $a > b$. □

Relation $\succ_m^v$ enjoys the useful property that there are finitely many bags lower than a given one if the same property holds for $>$.

**Definition 4.5** We say that $(W, >)$ is *finitely founded* if for every $x \in W$ the set $\mathit{down}_>(x) = \{\ y \in W \mid x > y\ \}$ is finite.

The set $down_>(x)$ is known in the literature on orders and algebras as the *down-set* of $x$ [9]. Also, finitely founded orders are sometimes called *past-finite.*

**Lemma 4.6** *If $(W, >)$ is finitely founded and $v \in N$, then $(bag(W), \succ_m^v)$ is finitely founded.*

*Proof.* First, observe that $(W, >)$ is well-founded, since finitely foundedness clearly implies well-foundedness. It is well-known that if $(W, >)$ is well-founded then $(bag(W), \succ_m)$ is well-founded as well (see e.g., [14]). Since $\succ_m^v \subseteq \succ_m$, we have that $(bag(W), \succ_m^v)$ is well-founded as well.

Let us show that $(bag(W), \succ_m^v)$ is finitely founded. The proof is by induction on the well-founded ordering $\succ_m^v$.

The bottom element is the empty bag $bag()$, for which the conclusion is immediate. Consider now $a = bag(x_1, \ldots, x_n)$ with $n > 0$. First, we observe that there are finitely many $a'$ such that $a \succ^v a'$. In fact, $a'$ is obtained by replacing some $x_i$ with at most $v$ elements from $down_>(x_i)$. Since such a set is finite, there is a finite number of possible $a'$'s. Let us show now that there are finitely many $b \in bag(W)$ such that $a \succ_m^v b$. In fact, either $a \succ^v b$ or for some $a'$, $a \succ^v a' \succ_m^v b$. In the former case, we have already showed the conclusion. In the latter case, by inductive hypothesis on $a'$, there are finitely many bags $b$ such that $a' \succ_m^v b$. Since there are finitely many $a'$'s, we also conclude that there are finitely many bags $b$ such that $a \succ^v a' \succ_m^v b$. □

We are now in the position to show that bounded programs and queries have bounded nondeterminism.

**Theorem 4.7 (Soundness)** *Let $P$ be a program and $Q$ a query both bounded by $|.|$ and $I$. Then $P$ and $Q$ have bounded nondeterminism.*

*Proof.* Consider an SLD-refutation $Q, Q_1, \ldots, Q_n$ of $P$ and $Q$ via a selection rule $s$. By Soundness of SLD-resolution, for every $i \in [1, n]$, $I \models \exists\ Q_i$. Moreover, by Lemma 4.1 for every $i \in [1, n]$, $Q_i$ is bounded by $|.|$ and $I$. By Lemma 4.3 *(ii)*, $|Q|^I \succ_m |Q_1|^I \succ_m \ldots \succ_m |Q_n|^I$ is a descending chain of bags over naturals. Let $v$ be the maximum number of atoms in the body of a clause from $P$. Since SLD-resolution replaces an atom by at most $v$ atoms, the stronger statement:

$$|Q|^I \succ_m^v |Q_1|^I \succ_m^v \ldots \succ_m^v |Q_n|^I$$

holds. Since $(N, >)$ is finitely founded, by Lemma 4.6 $(bag(N), \succ_m^v)$ is finitely founded, and, in particular, there are finitely many bags lower than $|Q|^I$, say $k$. Since it must necessarily be $k \geq n$, then the maximum length of a derivation of $P$ and $Q$ via $s$ is bounded by $k$. Since SLD-trees are finitely branching, by Köning's Lemma there are finitely many SLD-refutations for $P$ and $Q$ via $s$. □

**Example 4.8** The `ODDEVEN` program and the query `even(X), odd(X)` have bounded nondeterminism, since they are both bounded. The same conclusion holds for `PERMUTATION` and the query `perm([a, b], Ys)`. □

## 4.2 Completeness

In this section we show that boundedness is a complete characterization of bounded nondeterminism. First, we introduce a function that measures the maximum refutation length.

**Definition 4.9** We define $rlength_s^P(Q)$ as $\infty$ if there exist infinitely many SLD-refutations of $P$ and $Q$ via the selection rule $s$, and as the maximum length of an SLD-refutation of $P$ and $Q$ via $s$ otherwise. □

Note that $rlength_s^P(Q)$ is well-defined since SLD-trees are finitely branching, and then the maximum length of an SLD-refutation of $P$ and $Q$ is finite iff there are finitely many of them.

**Lemma 4.10** *Let $P$ be a program, $Q$ a query and $s$ a selection rule. Then:*

*(i) for every $Q'$ SLD-resolvent of $P$ and $Q$ via $s$*

$$rlength_s^P(Q) \ \rhd \ rlength_s^P(Q'),$$

*(ii) for every $Q'$ instance of $Q$, $rlength_s^P(Q) \ \unrhd \ rlength_s^P(Q')$.*

*Proof.* *(i)* Let $\xi'$ be an SLD-refutation of $P$ and $Q'$ via $s$ whose length is $l$. Since $Q'$ is an SLD-resolvent of $P$ and $Q$, there exists an SLD-refutation $\xi$ of $P$ and $Q$ that first selects the atom selected in $Q$ by $s$, then selects according to the selections of $\xi'$ . By the Independence Lemma [1, Theorem 3.33], there exists an SLD-refutation of $P$ and $Q$ via $s$ whose length is equal to the length of $\xi$, namely to $l+1$.

Consider now two cases.

Suppose that there exist infinitely many SLD-refutations of $P$ and $Q'$ via $s$. Since SLD-trees are finitely branching, by König's Lemma lengths are unbounded, and then by reasoning as above we can find infinitely many SLD-refutations of $P$ and $Q$ via $s$. Summarizing, *(i)* holds.

Suppose that there exist finitely many SLD-refutations of $P$ and $Q'$ via $s$. By reasoning as above in the case that $\xi'$ is the longest SLD-refutation, we get that there exists an SLD-refutations of $P$ and $Q$ via $s$ longer than $\xi'$, which implies *(i)*.

*(ii)*. Consider an SLD-refutation $\xi'$ of $P$ and $Q'$ via $s$. By the Lifting Theorem [1, Theorem 3.22] there exists an SLD-refutation $\xi'$ of $P$ and $Q$ which is of the same length of $\xi$. By the Independence Lemma [1, Theorem 3.33] there exists an SLD-refutation of $P$ and $Q$ via $s$ using the same clauses of $\xi'$. Therefore, $rlength_s^P(Q) \unrhd rlength_s^P(Q')$. □

The next lemma states that every program is bounded by a level mapping defined in terms of the length of SLD-refutations.

**Lemma 4.11** *Let $P$ be a program and $s$ a selection rule. Then there exist a level mapping $|.|$ and a Herbrand interpretation $I$ such that:*

*(i)* *$P$ is bounded by $|.|$ and $I$, and*

*(ii)* *for every $A \in B_L$, $|A| \in N$ iff there are finitely many SLD-refutations of $P$ and $A$ via $s$.*

*Proof.* We define $I = M_P^L$ and $|A| = \mathit{rlength}_s^P(A)$.

First, we observe that *(ii)* is immediate by definition of $|\ |$. Let us now consider *(i)*. We show the proof obligations of Definition 3.2. Obviously, $I$ is a model of $P$. Consider now $A \leftarrow B_1, \ldots, B_n$ in $\mathit{ground}_L(P)$ . Suppose that $I \models B_1, \ldots, B_n$. Since $I = M_P^L$, by Strong Completeness of SLD-resolution there exists an SLD-refutation $\xi$ of $P$ and $B_1, \ldots, B_n$ via $s$. We claim that for $i \in [1, n]$:

$$\mathit{rlength}_s^P(B_1, \ldots, B_n) \ \trianglerighteq \ \mathit{rlength}_s^P(B_i). \tag{5}$$

In fact, consider an SLD-refutation $\xi'$ of $P$ and $B_i$ via $s$. Since $\xi$ is successful, there exists an SLD-refutation of $P$ and $B_1, \ldots, B_n$ via $s$ where all the atoms in $\xi'$ are eventually selected, and the other selections are made accordingly to $\xi$. Thus, we obtain an SLD-refutation of $P$ and $B_1, \ldots, B_n$ via $s$ whose length is greater or equal than the length of $\xi'$, i.e. (5) holds. Observing that $B_1, \ldots, B_n$ is an instance of an SLD-resolvent $Q$ of $P$ and $A$, we calculate for $i \in [1, n]$:

$$\begin{array}{rcl} |A| & = & \mathit{rlength}_s^P(A) \\ & \triangleright & \quad \{ \text{ Lemma 4.10 } \textit{(i)} \ \} \\ & & \mathit{rlength}_s^P(Q) \\ & \trianglerighteq & \quad \{ \text{ Lemma 4.10 } \textit{(ii)} \ \} \\ & & \mathit{rlength}_s^P(B_1, \ldots, B_n) \\ & \trianglerighteq & \quad \{ \ (5) \ \} \\ & & \mathit{rlength}_s^P(B_i) = |B_i|. \end{array}$$

□

Let us show the completeness result. It is worth noting that the result states that, for a program $P$, there exists $|.|$ and $I$ such that they make bounded every query $Q$ such that $P$ and $Q$ have bounded nondeterminism. Boundedness of $P$ and $Q$ by such $|.|$ and $I$ turns out then to be the precise characterization of bounded nondeterminism.

**Theorem 4.12 (Completeness)** *Let $P$ be a program. Then there exist $|.|$ and $I$ such that $P$ is bounded by $|.|$ and $I$. Moreover, for every query $Q$ such that $P$ and $Q$ have bounded nondeterminism, $Q$ is bounded by $|.|$ and $I$.*

*Proof.* Let $s$ be any selection rule. Consider now the program $P' = P \cup \{ \texttt{new} \leftarrow Q \}$, on the language $L' = L \cup \{\texttt{new}\}$, where `new` is a fresh predicate symbol. By Lemma 4.11 *(i)*, $P'$ is bounded by some $|.|'$ and $I'$.

Let $|.|$ and $I$ be the restrictions of $|.|'$ and $I'$ on $B_L$.

Obviously, $P$ is bounded by $|.|$ and $I$ since the predicate symbol `new` is fresh.

Consider now $Q$. Since there are finitely many SLD-refutations of $P$ and $Q$ via $s$, the same holds for $P'$ and `new` via $s$. By Lemma 4.11 *(ii)*, $|\texttt{new}|' \in N$. Since $\texttt{new} \leftarrow Q$ is bounded by $|.|'$ and $I'$, we have that for every ground instance $A_1, \ldots, A_n$ of $Q$, if $I \models A_1, \ldots, A_n$ then $I' \models A_1, \ldots, A_n$ and then for $i \in [1, n]$, $|\texttt{new}|' \rhd |A_i|' = |A_i|$. In conclusion $Q$ is bounded by $|.|$ and $I$, by fixing $k = |\texttt{new}|'$ in Definition 3.3. □

**Example 4.13** Reconsider the `TRANSP` program of Example 3.7. Let $e$ be a graph containing a cycle $a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n = a_1$, with $n > 1$, where $x \rightarrow y$ means there is an arc from $x$ to $y$. Let us show that `TRANSP` and the query $\texttt{trans(}a_1\texttt{,}a_1\texttt{,}e\texttt{)}$ cannot be both bounded by the same $|.|$ and $I$, and then they cannot have bounded nondeterminism. Let $i$ be in $\{1, \ldots, n-1\}$. Since $I$ must be a model of `TRANSP`, it can be easily shown by induction on $n$ that $\texttt{trans(}a_{i+1}\texttt{,}a_n\texttt{,}e\texttt{)} \in I$. Also, $\texttt{member(arc(}a_i\texttt{,}a_{i+1}\texttt{),}e\texttt{)} \in I$ since $e$ is defined as a list of arcs. Consider now a ground instance of the second clause of `TRANSP`:

$$\texttt{trans(}a_i\texttt{,}a_n\texttt{,}e\texttt{)} \leftarrow \texttt{trans(}a_{i+1}\texttt{,}a_n\texttt{,}e\texttt{),member(arc(}a_i\texttt{,}a_{i+1}\texttt{),}e\texttt{).}$$

Since the body of the clause is true in $I$, by boundedness of `TRANSP`, we have $\texttt{trans(}a_i\texttt{,}a_n\texttt{,}e\texttt{)} \rhd \texttt{trans(}a_{i+1}\texttt{,}a_n\texttt{,}e\texttt{)}$, for $i \in \{1, \ldots, n-1\}$. As a consequence, $\texttt{trans(}a_1\texttt{,}a_1\texttt{,}e\texttt{)} = \texttt{trans(}a_1\texttt{,}a_n\texttt{,}e\texttt{)} \rhd \texttt{trans(}a_2\texttt{,}a_n\texttt{,}e\texttt{)} \rhd \ldots \rhd \texttt{trans(}a_n\texttt{,}a_n\texttt{,}e\texttt{)}$ $= \texttt{trans(}a_1\texttt{,}a_1\texttt{,}e\texttt{)}$. This implies $\texttt{trans(}a_1\texttt{,}a_1\texttt{,}e\texttt{)} = \infty$, i.e. it cannot be bounded by $|.|$ and $I$. □

## 5 From Bounded Nondeterminism to Strong Termination

So far, we have developed a theoretical framework for the characterization of the class of programs and queries that have the bounded nondeterminism property. The definition of bounded programs and queries can be readily used for *paper & pencil* proofs of bounded nondeterminism. We will see in the next section how automatic techniques proposed for inferring left termination can be adapted to infer boundedness. In this section, we show how a proof (paper & pencil or automatic) of boundedness can be useful in order to generate a terminating control for programs and queries.

We propose a simple syntactic transformation that prunes SLD-derivations in such a way that the SLD-tree is cut at a level that includes all refutations. Such a transformation provides us with a strong terminating program and query that retain the set of refutations of the original program and query. In order to cut derivations at an appropriate level, it is necessary to provide an upper bound $k$ satisfying the proof obligations of boundedness of the query under consideration. Such a $k$ should be readily available with the proof of boundedness of the query.

In the next definition, we introduce a syntactic transformational approach that prunes SLD-derivations at a certain length. For notational convenience, we denote by $\mathbf{T}$ a sequence $T_1, \ldots, T_n$ of terms, with $n \geq 0$.

**Definition 5.1** Let $P$ be a program and $Q$ a query both bounded by $|.|$ and $I$, and let $k \in N$. We define $Ter(P)$ as the program such that:

- for every clause in $P$

  $$\texttt{p}_0\texttt{(}\mathbf{T}_0\texttt{)} \leftarrow \texttt{p}_1\texttt{(}\mathbf{T}_1\texttt{), ..., p}_n\texttt{(}\mathbf{T}_n\texttt{).}$$

  with $n > 0$, the clause

  $$\texttt{p}_0\texttt{(}\mathbf{T}_0\texttt{, s(D) )} \leftarrow \texttt{p}_1\texttt{(}\mathbf{T}_1\texttt{, D), ..., p}_n\texttt{(}\mathbf{T}_n\texttt{, D).}$$

  is in $Ter(P)$, where `D` is a fresh variable,

- and, for every clause in $P$

  $$\texttt{p}_0\texttt{(}\mathbf{T}_0\texttt{).}$$

  the clause

  $$\texttt{p}_0\texttt{(}\mathbf{T}_0\texttt{, D).}$$

  is in $Ter(P)$, where `D` is a fresh variable.

Also, for $Q = \texttt{p}_1\texttt{(}\mathbf{T}_1\texttt{), ..., p}_n\texttt{(}\mathbf{T}_n\texttt{)}$, with $n \geq 0$, define $Ter(Q, k)$ as the query:

$$\texttt{p}_1\texttt{(}\mathbf{T}_1\texttt{, s}^k\texttt{(0)), ..., p}_n\texttt{(}\mathbf{T}_n\texttt{, s}^k\texttt{(0)).}$$

□

The next result shows that the transformation yields strongly terminating programs and queries. Also, the definition of boundedness for queries provides us with an upper bound for cutting derivations without loosing any refutation.

**Theorem 5.2** *Let $P$ be a program and $Q$ a query both bounded by $|.|$ and $I$, and let $k$ be a given natural number satisfying Definition 3.3.*

*Then, for every $b \in N$, $Ter(P)$ and $Ter(Q, b)$ universally terminate via every selection rule.*

*Moreover, there is a bijection between SLD-refutations of $P$ and $Q$ via a selection rule $s$ and SLD-refutations of $Ter(P)$ and $Ter(Q, k-1)$ via $s$.*

*Proof.* It is readily checked that $Ter(P)$ and $Ter(Q, b)$ are recurrent by a level mapping $|.|$ such that $|\texttt{p(t, }n\texttt{)}| = size(n)$. Therefore, universal termination follows from Theorem 2.5. Consider now the second part of the Theorem.
First, assume that $s$ is the leftmost selection rule. We define a mapping $\psi$ as follows. Consider an SLD-refutation $\xi'$ for $Ter(P)$ and $Ter(Q, k-1)$ via $s$. $\psi$ maps $\xi'$ into the derivation obtained by removing the rightmost term in every atom in $\xi'$. Since $s$

is the leftmost selection rule, such a derivation is readily checked to be a refutation for $P$ and $Q$ via $s$. Consider now an SLD-refutation $\xi = Q_0, Q_1, \dots, Q_n$ for $P$ and $Q$ via $s$. It can be obtained as $\psi(\xi')$ for $\xi' = Q'_0, Q'_1, \dots, Q'_n$ SLD-refutation for $Ter(P)$ and $Ter(Q, b)$ for some $b \in N$. Let us show that for $b = k-1$. Observe that, by Lemma 4.3 *(ii)*,

$$|Q_0|^I \succ_m |Q_1|^I \succ_m \dots \succ_m |Q_n|^I.$$

Called $bag(Q'_i)$ the bag with elements the rightmost terms in $Q'_i$, we claim that $bag(Q'_i) \succeq_m |Q_i|^I$ for $i \in [0:n]$. In fact, $bag(Q'_0) = bag(k-1, \dots, k-1) \succeq_m |Q_0|^I = |Q|^I$ since $Q$ is bounded using $k$. Moreover, $bag(Q'_{i+1})$ is obtained from $bag(Q'_i)$ by replacing an element $a$ with a number $v$ of elements $a-1$. On the contrary, $|Q_{i+1}|$ is obtained from $|Q_i|$ by replacing an element $a$ with a number $v$ of elements lower or equal than $a-1$. Summarizing, $bag(Q'_i) \succeq_m |Q_i|^I$ for $i \in [0:n]$ implies that $\xi'$ cannot fail due to unification of the rightmost term in the selected atom, i.e. $\xi'$ is a refutation. Therefore, by fixing $b = k-1$ the refutation $\xi$ can be obtained as $\psi(\xi')$ for some refutation $\xi'$ of $Ter(P)$ and $Ter(Q, b)$.

We have shown that $\psi$ is a bijection (modulo renaming apart) when $s$ is the leftmost selection rule. Consider now the case that the selection rule $s$ is arbitrary. By Strong Completeness of SLD-resolution there is a bijection $\phi$ mapping SLD-refutations via $s$ to SLD-derivations via the leftmost selection rule. Therefore, the conclusion of the Theorem follows by considering the bijection $\phi \circ \psi \circ \phi^{-1}$. □

It is worth noting that no assumption is made on the selection rule $s$, i.e. any selection rule is a terminating control for the transformed program and query.

**Example 5.3** Reconsider the program `PERMUTATION` and the query `perm([a, b], Ys)` of Example 1.2. The transformed program $Ter$(`PERMUTATION`) is:

```
perm([], [], D).
perm([X|Xs], Ys, s(D)) ←
   delete(X, Ys, Zs, D),
   perm(Xs, Zs, D).

delete(X, [X|Y], Y, D).
delete(X, [H|Y], [H|Z], s(D)) ←
   delete(X, Y, Z, D).
```

and the transformed query for $k = 3$ is `perm([a,b], Ys, s`$^2$`(0))`. By Theorem 5.2, the transformed program and query provide us with a terminating control for the original program and query independently of the selection rule adopted, modulo the extra argument added to each predicate. □

The transformations $Ter(P)$ and $Ter(Q, k)$ are of purely theoretical interest. In practice, one would implement these counters directly into the compiler/interpreter.

Also, the compiler/interpreter should include a module that infers an upper bound $k$ automatically. The issue of the automatic inference of level mappings and models is discussed in the next section.

## 6 On the Automatic Inference of Boundedness

### 6.1 An undecidable problem

On the theoretical level, the problem of deciding whether a program is bounded is undecidable.

**Theorem 6.1** *It is undecidable whether there exist $|.|$ and $I$ such that a program $P$ and a query $Q$ are both bounded by $|.|$ and $I$.*

*Proof.* Devienne et al. [16, Theorem 8] showed that it is undecidable whether given a program consisting of only one clause of the form:

$$\texttt{p(}T_1,\ldots,T_n\texttt{)} \leftarrow \texttt{p(}S_1,\ldots,S_n\texttt{).} \tag{6}$$

and a query $\texttt{p(}V_1,\ldots,V_n\texttt{)}$, the SLD-resolution stops. The particular form (6) implies that there is only one SLD-derivation for the program and the goal. Consider now for every program $P$ of the form (6), the program $P'$ obtained by adding the fact $\texttt{p(X}_1,\ldots,\texttt{X}_n\texttt{)}$, where $\texttt{X}_1,\ldots,\texttt{X}_n$ are distinct variables. Due to their particular form, we observe that $P$ and the query $\texttt{p(}V_1,\ldots,V_n\texttt{)}$ have a finite derivation iff $P'$ and $\texttt{p(}V_1,\ldots,V_n\texttt{)}$ have bounded nondeterminism, i.e. they are bounded by some $|.|$ and $I$. Therefore, if it were decidable whether a given program and query are bounded by some level mapping and Herbrand interpretation, then it would be decidable whether the SLD-resolution stops for programs and queries of the form (6). □

### 6.2 Insights on automation

On the practical level, many approaches are currently available to automatically infer sufficient conditions for termination – usually left termination.

On the one hand, we observe that any of the existing automated methods for proving termination via any selection rule is actually a sufficient method for proving bounded nondeterminism. When, in addition, the automated method can provide us with an upper bound on the length of derivations of a program $P$ and a query $Q$ then such an upper bound is, obviously, an upper bound for the length of refutations of $P$ and $Q$.

On the other hand, we argue that some of the approaches can be directly adapted for proving the proof obligations of boundedness, since they closely resembles the ones for acceptability (the characterization of left terminating programs [3]). In the following, we outline the adaptation of the constraint-based method of

Decorte et al. [13] to the case of an example program. First of all, however, we need to recall some standard notions concerning modes.

For a predicate $p/n$, a *mode* is an atom $p(m_1, \ldots, m_n)$, where $m_i \in \{I, O\}$ for $i \in [1, n]$. Positions with $I$ are called *input positions*, and positions with $O$ are called *output positions* of $p$. To simplify the notation, an atom written as $p(\mathbf{s}, \mathbf{t})$ means: $\mathbf{s}$ is the vector of terms filling in the input positions, and $\mathbf{t}$ is the vector of terms filling in the output positions. Intuitively, a mode specifies the use of predicate arguments of $p$, with the intended meaning that terms occurring in output positions are determined from the computation of the terms occurring in input positions. We now define level mappings where the dependency on the modes is made explicit [17].

**Definition 6.2** A *moded level mapping* $|.|$ is a level mapping such that for any ground $\mathbf{s}$, $\mathbf{t}$ and $\mathbf{u}$, $|p(\mathbf{s}, \mathbf{t})| = |p(\mathbf{s}, \mathbf{u})|$.

The condition $|p(\mathbf{s}, \mathbf{t})| = |\mathbf{p}(\mathbf{s}, \mathbf{u})|$ states that the *level* of an atom is independent from the terms in its output positions.

**Example 6.3** Let us consider again `PERMUTATION`, and the query `perm([a, b], Ys)`. Consider now the problem to infer that `PERMUTATION` and the query above are bounded by a level mapping $|.|$ and a Herbrand interpretation $I$. Here, we make the following assumptions:

**(Assumption A1).** Every $n$-ary predicate symbol $p$ is annotated with exactly one mode. As an example, the following are intuitive modes for the predicates of `PERMUTATION`:

$$\texttt{perm(}I,\ O\texttt{)} \quad \texttt{delete(}I,\ O,\ I\texttt{)}.$$

**(Assumption A2).** $|.|$ is a moded level mapping defined as a linear combination of the list-length of the predicate arguments which occur in input positions, i.e.:

$$\begin{aligned} |\texttt{perm(}xs,\ ys\texttt{)}| &= p_0 + p_1|xs| \\ |\texttt{delete(}x,\ xs,\ ys\texttt{)}| &= d_0 + d_1|x| + d_2|ys|, \end{aligned}$$

where $p_0, p_1, d_0, d_1, d_2$ denote natural numbers that need to be determined. Strictly speaking, the formulas above do not define a single level mapping, but a set of level mappings parametric in $p_0, p_1, d_0, d_1, d_2$.

**(Assumption A3)** $I$ is characterized as the set of atoms whose predicate arguments satisfy a linear inequation, such as:

$$\begin{aligned} I \quad = \quad & \{\ \texttt{perm(}xs,\ ys\texttt{)} \mid p'_0 + p'_1|xs| \geq p'_2|ys|\ \} \cup \\ & \{\ \texttt{delete(}x,\ xs,\ ys\texttt{)} \mid d'_0 + d'_1|x| + d'_3|ys| \geq d'_2|xs|\ \} \end{aligned}$$

where $p'_0, p'_1, p'_2, d'_0, d'_1, d'_2, d'_3$ denote natural numbers that need to be determined. Observe that the linear inequations are syntactically derived by fixing the arguments occurring in input positions to the left hand side of the inequation, and those occurring in output positions to the right hand side. As in **(Assumption A2)**, note that formula above actually defines a set of models. Intuitively, the sets of level mappings and models represent our *search space* for a level mapping and a model that satisfy the proof obligations of boundedness.

Let us write the proof obligations of Definition 3.2 for clauses of `PERMUTATION`.

*(p1)* For clause *(p1)*, we have only to show that $I$ is a model of it, i .e. `perm([], [])` is in $I$. This leads us to the symbolic constraint $p'_0 + p'_1 \cdot 0 \;\geq p'_2 \cdot 0$, i.e.:

$c1$. $p'_0 \;\geq\; 0$.

*(p2)* Consider a ground instance of *(p2)*:

`perm([`$x$`|`$xs$`], `$ys$`)` $\leftarrow$ `delete(`$x$`, `$ys$`, `$zs$`), perm(`$xs$`, `$zs$`).`

The body is true in $I$ iff $\phi$ holds, where:

$$\phi \;\equiv\; d'_0 + d'_1|x| + d'_3|zs| - d'_2|ys| \;\geq\; 0 \;\wedge\; p'_0 + p'_1|xs| - p'_2|zs| \;\geq\; 0.$$

Therefore, the decreasing of the level mapping from the head to the leftmost atom in the body imposes the constraint:

$$\forall\, x, xs, ys, zs \quad \phi \;\Rightarrow\; p_0 + p_1|xs| + p_1 \;>\; d_0 + d_1|x| + d_2|zs|$$

which after some rearrangement, can be written as:

$c2$. $\forall\, x, xs, ys, zs \quad \phi \;\Rightarrow\; p_1|xs| - d_1|x| - d_2|zs| + (p_0 + p_1 - d_0) \;>\; 0.$

Note that have used $>$ instead of $\rhd$ , as required in the definition of boundedness. This is due to the form of level mappings of **(Assumption A2)**, which can never map into $\infty$. The decreasing from the head to the second body atom, and the requirement that $I$ must be a model of the clause lead to:

$c3$. $\forall\, x, xs, ys, zs \quad \phi \;\Rightarrow\; p_1 \;>\; 0.$

$c4$. $\forall\, x, xs, ys, zs \quad \phi \;\Rightarrow\; p'_1|xs| - p'_2|ys| + (p'_1 + p'_0) \;\geq\; 0.$

*(d1)* We have only to show that $I$ is a model of it, i.e.

$c5$. $\forall\, x, y,\; (d'_3 - d'_2)|y| + d'_1|x| + (d'_0 - d'_2) \geq 0.$

*(d2)* The decreasing from the head to the body atom, and the requirement that $I$ must be a model of the clause yield:

$c6.\ \forall\, x, y, z \quad d'_1|x| + d'_3|z| - d'_2|y| + d'_0 \ \geq\ 0 \ \Rightarrow\ d_2 \ >\ 0.$

$c7.\ \forall\, x, y, z \quad d'_1|x|+d'_3|z|-d'_2|y|+d'_0 \ \geq\ 0 \Rightarrow d'_1|x|+d'_3|z|-d'_2|y|+(d'_3+d'_0-d'_2) \ \geq\ 0.$

A general method to solve conditional constraints $c1 - c7$ of the form above is not known. However, Decorte et al. [13] propose a method that reduces those constraints to a set of linear constraints over the variables $p_0, p_1, p_2, p'_0, p'_1, \ldots$. The basic idea consists of observing that $c1 - c7$ are solvable by imposing that all the coefficients appearing in the inequations at the right of the implications are non-negative[2]. However, since this *sufficient* condition is in many cases too strong, they propose to apply first a rule[3] that nondeterministically selects a constraint of the form $\forall\ \ldots e \geq 0\ \wedge\ \ldots\ \Rightarrow\ e' \geq 0$ (resp., $> 0$) and rewrites it into:

$$\forall\ \ldots e \geq 0\ \wedge\ \ldots\ \Rightarrow\ e' - e \geq 0 \quad \text{(resp., } > 0\text{)}.$$

For instance, when applied to $c2$ and the second conjunct in $\phi$, this rule yields:

$c2'.\ \forall\, x, xs, ys, zs \quad \phi\ \Rightarrow$
$$-d_1|x| + (p_1 - p'_1)|xs| + (-d_2 + p'_2)|zs| + (p_0 + p_1 - d_0 - p'_0) \ >\ 0.$$

A sufficient condition to satisfy this constraint is then to require:

$$\begin{array}{ll} -d_1 \ \geq\ 0, & p_1 - p'_1 \ \geq\ 0 \\ -d_2 + p'_2 \ \geq\ 0, & p_0 + p_1 - d_0 - p'_0 \ >\ 0. \end{array}$$

With the same approach, we derive the following constraints from $c1, c3 - c7$:

$$\begin{array}{ll} p_1 \ >\ 0, & -d'_1 \ \geq\ 0, \\ p'_2 - d'_3 \ \geq\ 0, & d'_2 - p'_2 \ \geq 0, \\ p'_1 - d'_0 \geq 0, & d'_3 - d'_2 \ \geq\ 0, \\ d'_0 - d'_2 \ \geq\ 0, & d_2 \ >\ 0, \end{array}$$

where variables range over naturals. Such constraints are directly solvable by a constraint solver over finite domains – and often over boolean suffices. A solution of those constraints is the following:

$$\begin{array}{c} p_1 = d_2 = p'_1 = p'_2 = d'_0 = d'_2 = d'_3 = 1 \\ p_0 = d_0 = d_1 = d'_1 = p'_0 = d'_0 = a_1 = 0, \end{array}$$

[2]More precisely, since $|.|$ maps ground terms into naturals, the right hand side of implications is of the form $c_0 + c_1x_1 + \ldots + c_nx_n \geq 0$ or $c_0 + c_1x_1 + \ldots + c_nx_n > 0$. These inequalities are satisfied if $c_i \geq 0$ for all $i \in [0, n]$, or if ($c_i \geq 0$ for all $i \in [1, n]$ and $c_0 > 0$) respectively.

[3]In addition to the rewrite rule we consider in this paper, [13] proposes also a *substitution rule* that is meaningful in presence of the equality predicate.

which leads to the level mapping and the interpretation:

$$|\texttt{perm(}xs\texttt{, }ys\texttt{)}| = |xs|$$
$$|\texttt{delete(}x\texttt{, }xs\texttt{, }ys\texttt{)}| = |ys|,$$

$$\begin{aligned} I \;=\; & \{\ \texttt{perm(}xs\texttt{, }ys\texttt{)} \mid |xs| \geq |ys|\ \} \cup \\ & \{\ \texttt{delete(}x\texttt{, }xs\texttt{, }ys\texttt{)} \mid 1 + |ys| \geq |xs|\ \}. \end{aligned}$$

Notice how $|.|$ and $I$ closely resemble the level mapping and the interpretation of Example 3.6. Let us see now the proof obligations of Definition 3.3. Consider a ground instance perm($xs$, $ys$) of an atomic query. We have to find out a natural $k$ such that:

$$I \models \texttt{perm(}xs\texttt{, }ys\texttt{)} \ \Rightarrow\ k > |\texttt{perm(}xs\texttt{, }ys\texttt{)}|,$$

which by definition of $|.|$ and $I$, can be rewritten as: $|xs| \ \geq\ |ys| \ \Rightarrow\ k > |xs|$. In the case of the query `perm([a,b], Ys)`, we have then the constraint:

$$\forall\, ys \quad 2\ \geq\ |ys|\ \Rightarrow\ k > 2.$$

It is worth noting that it is of the same form as the constraints derived from program clauses. Therefore, it can be solved by the same approach, which yields the solution $k = 3$, which coincides with the $k$ used in Example 3.6.

Finally, we observe that, even though the assumptions **(A2)** and **(A3)** use a predefined function on terms, namely the list-length, the approach can be defined in general terms, i.e. with also the construction of those functions involved in the termination analysis. This is actually the approach of Decorte et al. [13]. □

## 6.3 Discussion and the modularity issue

The example of the last section reports on the adaption of an existing method for left termination to infer boundedness. In general, for a clause $A \leftarrow B_1 , \ldots , B_n$ there are $n+1$ symbolic constraints to satisfy, one about showing that the symbolic interpretation is a model of the clause and $n$ about showing the decreasing of the symbolic level mapping from the head to each body atom. The proposed sufficient condition to satisfy the symbolic constraints is to solve a set of linear inequations over the coefficients of variables in the symbolic constraints.

Assume that solving such a set of inequations requires an acceptably low cost. Since the proposed condition is only sufficient, it can happens that one or more weakening steps are required. Performing a weakening step represents, in such a context, searching the space of solutions. It is then legitimate to ask ourselves how large is such a space. A weakening step nondeterministically selects a symbolic constraint (over $n+1$ available) and a conjunct at its left hand side (over $n$ available). In total, there are $n \cdot (n+1)$ possible choices. Therefore, we can see our search space as an (infinite) tree where at each node we can follow one out of $n \cdot (n+1)$ weakening steps. Realistically, this means that we can explore at most the top few levels of such a tree (e.g., for $n = 3$ there are 144 nodes at level 3, 1728 at level 4, and 20736 at level 5).

To better understand the problem, let us consider what differentiates the approach of Decorte et al. [13] from ours. In their approach, proof obligations (for the left termination problem) of the following form are considered:

$$\text{for } i \in [1,n] \;\; I \models B_1, \ldots, B_{i-1} \;\; \wedge \;\; \mathit{rel}(A) \simeq \mathit{rel}(B_i) \;\; \mathit{implies} \;\; |A| \; > \; |B_i|,$$

where $\mathit{rel}(A) \simeq \mathit{rel}(B)$ if the predicate symbols of $A$ and $B$ are mutually recursive. On the one hand, such proof obligations yield fewer symbolic constraints, i.e. one for every mutually recursive body atom, against one for every body atom in our approach. On the other hand, they yield symbolic constraints with fewer conjuncts, i.e. $i - 1$ against $n$ in our approach.

As an example, considering the collection of definite programs of the Apt book [1, page xiii], we have calculated that 19.6% of clauses with non-empty body have zero mutually recursive calls, 69.1% have only one mutually recursive call, and 11.3% have more than one mutually recursive call. For such a collection of programs, we have that:

- in 19.6% of clauses, the approach of Decorte et al. yields no symbolic constraint at all – i.e., an empty search tree;
- in 69.1% of clauses it yields one symbolic constraint with *at most* $n - 1$ conjuncts, i.e. a search tree whose branching degree is linear in the number $n$ of body atoms;
- and in the remaining 11.3% it yields *at most* $n$ symbolic constraints for a total of *at most* $n \cdot (n-1)/2$ conjuncts, i.e. a search tree whose branching degree is *in the worst case* in the same order of our approach.

Even more importantly, since the proof obligations of the approach of Decorte et al. are modular (i.e., they do not take into account predicates defined in lower modules), the method scales up to large programs made up of many small modules. In this context, a module is the set of program clauses defining predicates mutually recursive among them. Each module is analyzed apart by the proof method.

On the contrary, our approach must consider the (large) program as a whole, since the weakening of a symbolic constraint in a lower module can affect solvability of symbolic constraints at higher modules. In other words, *the search space for a program is the cartesian product of the search spaces for each clause in the program.* As a consequence, except for tiny programs, the search space is dramatically large, and the approach results infeasible in the general case.

Summarizing, while we are confident that automatic approaches for left termination can be adapted to infer boundedness, it is first necessary to investigate on a refinement of the proof obligations of boundedness that takes into account modularity. In fact, since the modularity issue is originated from the definition of boundedness itself, we expect that it will arise independently of the approach used to automate the inference of boundedness. As a first step towards tackling the modularity issue, we propose a refinement that mimics the approach of Bossi et al. [17] in making acceptability modular for a restricted class of programs.

**Definition 6.4** Let $|.|$ be a level mapping, and $I$ a Herbrand interpretation. A logic program $P$ is *well-bounded by* $|.|$ *and* $I$ iff $I$ is a model of $P$ such that for every $A \leftarrow B_1\,, \dots\,, B_n$ in $ground_L(P)$ :

$$I \models B_1\,, \dots\,, B_n \quad implies \quad \text{for } i \in [1,n] \quad \text{if } rel(A) \simeq rel(B_i) \text{ then } |A| \ \rhd\ |B_i|.$$

□

Well-boundedness requires the decreasing for mutually recursive calls only. As one could expect, some additional requirements must hold on a given program and query in order to conclude bounded nondeterminism. Intuitively, since there can be finitely many non-mutually recursive calls between two mutually recursive ones, a sufficient condition is to impose that non-mutually recursive calls introduce atoms whose level (of its ground instances) is bounded by some natural number. In this way, there can be finitely many recursive calls only (by well-boundedness), each of them introduces finitely many non-recursive calls (since the number of predicates in $P$ is finite) and the level of each of these calls is bounded (by some sufficient condition to be determined). In order to achieve this, we first recall the notion of well-modedness [1].

**Definition 6.5** A query $Q = p_1(\mathbf{s}_1, \mathbf{t}_1), \dots, p_n(\mathbf{s}_n, \mathbf{t}_n)$ is *well-moded* if for all $i \in [1,n]$ and $K = 1$

$$vars(\mathbf{s}_i) \subseteq \bigcup_{j=K}^{i-1} vars(\mathbf{t}_j) \tag{7}$$

The clause $p(\mathbf{t}_0, \mathbf{s}_{n+1}) \leftarrow Q$ is *well-moded* if (7) holds for all $i \in [1, n+1]$ and $K = 0$. A program is *well-moded* if all of its clauses are well-moded.

A query (clause, program) is *permutation well-moded* if it is well-moded modulo reordering of the atoms of the query (each clause body).

Well-modedness of $P$ and $Q$ is a persistent property. It ensures that atoms selected in a derivation of $P$ and $Q$ via the leftmost selection rule are ground in their input positions. Permutation well-modedness is a weakening implying groundness of terms in input positions of at least one atom in every query of any derivation of $P$ and $Q$.

**Lemma 6.6** *[2, 29] Let $P$ and $Q$ be a (permutation) well-moded program and query. Then:*

*(i) every SLD-resolvent of them is (permutation) well-moded;*

*(ii) every computed instance $Q'$ of them is ground.* □

*(ii)* reveals that the class of well-moded programs and queries is quite restricted, since they admit ground computed instances only.

If, in addition to (permutation) well-modedness, we assume a *moded* level mapping into naturals, we can conclude that the level of atoms in a refutation is bounded, which is precisely the sufficient condition we are looking for.

**Theorem 6.7** *Let $P$ be a (permutation) well-moded program, $|.| : B_L \to N$ a moded level mapping into naturals and $I$ a Herbrand interpretation.*

*If $P$ is well-bounded by $|.|$ and $I$ then $P$ and any (permutation) well-moded query $Q$ have bounded nondeterminism.*

*Proof.* We assume that $P$ and $Q$ are well-moded. The reasoning for permutation well-modedness follows in a similar way.

Let LD be the leftmost selection rule, and an LD-refutation an SLD-refutation via LD. Let $S$ be a subset of predicates of $L$. We denote by $P_S$ the clauses in $P$ defining the predicates in $S$. We show by induction on $S$ (with the subset ordering) that $P_S$ and any well-moded query $Q$ have finitely many LD-refutations. By Strong Completeness of SLD-resolution, this implies that $P$ (which is $P_S$ for $S$ including all predicates appearing in $P$) and any well-moded query have bounded nondeterminism.

*Base case.* Since $P_S = P_\emptyset = \emptyset$ the conclusion is immediate.

*Inductive step.* Let $Q = A_1 , \ldots , A_m$ . The proof is trivial for $m = 0$. Consider $m > 0$. Suppose for the moment that $P_S$ and (the well-moded atom) $A_1$ have finitely many LD-refutations. This means that $P_S$ and $Q$ have finitely many partial derivations via LD that end with a resolvent of the form $Q' = (A_2, \ldots , A_m)\theta$. By Lemma 6.6 *(i)* $Q'$ is well-moded. Therefore, we can apply inductive hypothesis (on the length of the query) to conclude that $P_S$ and $Q'$ have finitely many LD-refutations. A fortiori, $P_S$ and $Q$ have finitely many LD-refutations.

Let us show now that $P_S$ and a well-moded atom $A$ have finitely many LD-refutations (since $A$ is well-moded and $|.|$ is moded, with a little abuse of notation we write $|A|$ to denote the constant value $|A'|$ for any $A'$ ground instance of $A$.)

Consider a refutation $\xi = A, Q_1, \ldots , Q_n$ of $P_S$ and $A$ via a selection rule $s$ that always selects first atoms $B$ such that $rel(A) \simeq rel(B)$ and then behaves as the LD selection rule.

Let $\theta_1, \ldots , \theta_n$ be the mgu's used in the refutation, and $\theta = \theta_1 \circ \ldots \circ \theta_n$. By Lemma 6.6 *(ii)*, $A\theta$ and $Q_i\theta$, for $i \in [1, n]$, are ground. Moreover, by Soundness of SLD-resolution $I \models A\theta$ and $I \models Q_i\theta$. Since $P$ is well-bounded and $|.|$ is moded, the number of calls to predicates mutually recursive with $rel(A)$ are at most $|A\theta| = |A|$. As a consequence, there are finitely many $Q'$ such that $A, Q_1, \ldots , Q'$ is a prefix of a refutation of $P_S$ and $A$ via $s$, and $Q'$ contains only atoms $B$ with $rel(A) \neq rel(B)$. Since each $Q'$ is well-moded (by Lemma 6.6 *(i)*), we can apply the inductive hypothesis (on $S$) to conclude that there are finitely many LD-refutations for $P_{S\setminus\{rel(A)\}}$ and $Q'$. By definition of $s$, $P_S$ and $A$ have finitely many refutations via $s$. By Strong Completeness of SLD-resolution, $P_S$ and $A$ have finitely many LD-refutations. □

**Example 6.8** Let us consider again `PERMUTATION` and the symbolic constraint solving approach of Example 6.3. First, observe that `PERMUTATION` is well-moded with the moding:

$$\texttt{perm(}I\texttt{, }O\texttt{)} \quad \texttt{delete(}I\texttt{, }O\texttt{, }I\texttt{)}$$

given in Example 6.3.

In order to show well-boundedness we start considering clauses defining `delete`. The symbolic constraints imposed by well-boundedness are the same of boundedness, i.e. *(c5-c7)* of Example 6.3. However, since the level mapping for `delete`-atoms cannot influence the one for `perm`-atoms, we can now solve *(c5-c7)* independently from the symbolic constraints for clauses defining `perm`. The same approach of Example 6.3, yields:

$$\begin{array}{rcl} & & |\texttt{delete(}x\texttt{, }xs\texttt{, }ys\texttt{)}| = |ys|, \\ I & = & \{\ \texttt{delete(}x\texttt{, }xs\texttt{, }ys\texttt{)} \mid 1 + |ys| \geq |xs|\ \}. \end{array}$$

Using this partial information, we can write down the symbolic constraints for clauses defining `perm`.

*(p1)* We have only to show that the symbolic interpretation is a model of it, i.e.

$c1.\ p_0' \ \geq\ 0.$

*(p2)* Consider a ground instance of *(p2)*:

$$\texttt{perm([}x\texttt{|}xs\texttt{], }ys\texttt{)} \ \leftarrow\ \ \texttt{delete(}x\texttt{, }ys\texttt{, }zs\texttt{), perm(}xs\texttt{, }zs\texttt{).}$$

The body is true in $I$ if $\phi'$ holds, where:

$\phi' \ \equiv\ 1 + |zs| - |ys| \ \geq\ 0 \ \wedge\ p_0' + p_1'|xs| - p_2'|zs| \ \geq\ 0,$

is now the equivalent of $\phi$ where parameters for `delete` are replaced with the partial solution computed so far. The decreasing of the level mapping is now required only from the head to the rightmost atom in the body:

$c2.\ \forall\ x, xs, ys, zs \quad \phi' \ \Rightarrow\ p_1 \ >\ 0.$

The requirement that the symbolic interpretation must be a model of the clause leads to:

*c*3. $\forall\ x, xs, ys, zs \quad \phi' \ \Rightarrow\ p'_1|xs| - p'_2|ys| + (p'_1 + p'_0)\ \geq\ 0.$

Solving *(c1-c3)* is now (computationally) simpler, and yields solutions including the one of Example 6.3:

$$\begin{aligned} & |\texttt{perm(}xs\texttt{, }ys\texttt{)}| = |xs| \\ I \ = \ & \{\ \texttt{perm(}xs\texttt{, }ys\texttt{)} \mid |xs| \geq |ys|\ \}. \end{aligned}$$

By Theorem 6.7, we conclude that `PERMUTATION` and any query perm($xs$, Ys), with $xs$ ground, have bounded nondeterminism. Due to the requirement of well-modedness, such a conclusion is weaker than the one of Example 6.3, where $xs$ can be any (not necessarily ground) term. □

Finally, observe that assuming level mapping into naturals is essential to achieve the result of Theorem 6.7.

**Example 6.9** The program below and the query `p` does not have bounded nondeterminism.

```
p ← q.
q ← q.
q.
```

However, the program and `p` are well-bounded by defining $|\texttt{p}| = 0$ and $|\texttt{q}| = \infty$. □

## 7 Conclusions

### 7.1 Related Work

#### Universal termination

A survey on termination of logic programs can be found in the paper by De Schreye and Decorte [10], covering both theoretical characterizations and automation issues. A more recent survey on the characterizations of classes of programs terminating with respect to different selection rules is due to Pedreschi et al. [25]. In particular, the survey covers bounded programs by citing some results from the conference version ([24]) of the present paper. [25] points out that acceptability proof obligations (that characterize left termination [3]) imply boundedness ones, and that fair-boundedness proof obligations (that characterize termination via fair selection rules [28]) imply boundedness ones as well.

#### Pruning SLD-derivations

The idea of pruning SLD-derivations is common to the research area of loop checking (see e.g., [5]). While a run-time analysis is potentially able to cut more unsuccessful branches, the evaluation of a pruning condition at run-time, such as for loop checks, involves a considerably high computational overhead. On the contrary, our approach combines both the advantages of a static analysis (termination) method, i.e. the analysis is conducted once and no run-time overhead is added, with those of pruning mechanisms, i.e. not being restricted to terminating derivations. In particular, observe that, by the Completeness Theorem 4.12, bounded programs and queries are the largest class such that a pruning mechanism can find out all refutations in a finite time.

Martin and King [22] showed a transformation for Gödel programs, that shares with the transformation $Ter$, the idea of not following derivations longer than a certain length. However, compared with our approach, they rely on sufficient conditions for evaluating an upper bound on the length of refutations, namely termination via a class of selection rules called *semilocal*. Obviously, if we can show that a program and a query universally terminate via some selection rule $s$ then no refutation can be longer than the maximum derivation via $s$. Also, their transformation adds run-time overhead, since the maximum length is computed at run time.

**Estimating the number of computed instances**

Sufficient (semi-)automatic methods to approximate the number of computed instances by means of lower and upper bounds have been studied in the context of cost analysis of logic programs [11, 12] and of cardinality analysis of Prolog programs [6]. As an example, cost analysis is exploited in the Ciao-Prolog system [8]. Of course, if $\infty$ is a lower bound to the number of computed instances of $P$ and $Q$ then they cannot have bounded nondeterminism. Dually, if $n \in N$ is an upper bound then $P$ and $Q$ have bounded nondeterminism. In this case, however, we are still left with the problem of determining a level of the SLD-tree that includes all the refutations.

**Decidability and testing**

The class of bounded logic programs was investigated by Ruggieri [27] in the context of decidability of $\mathcal{M}, \mathcal{C}$ and $\mathcal{S}$-semantics. That paper shows that for a given program $P$ bounded by a given and computable level mapping, the semantics sets of $P$ (w.r.t. the $\mathcal{M}, \mathcal{C}$ and $\mathcal{S}$-semantics) are decidable. As a consequence, it is decidable whether a query is a correct or a computed instance of another one (in software engineering words, this is *testing* a program).

**Automatic inference**

Let us briefly discuss on the adaptation of the approach of Decorte and De Schreye [13] to infer acceptability. On the one hand, we have replaced the generation of proof obligations for acceptability with those for boundedness. On the other hand, their notion of acceptability reasons at non-ground level (i.e., considers not necessarily ground instances of clauses and queries). This implies a further proof obligation, known as *rigidity* of level mappings, that is tackled separately. In our approach, the equivalent of rigidity is represented by the requirements on queries, which are tackled in the same constraint satisfaction framework of the other requirements.

Among other possible approaches on proving termination that could be extended to infer boundedness, we mention the *TermiLog* system of Lindenstrauss et al. [15, 20, 21], the implementation of the static termination analysis algorithm of the Mercury system [30], and the LPTP system of Stärk [31] to prove both termination and partial correctness at a time.

Also, other works that could provide insights in automation concern inference of left-termination, i.e. finding a (as large as possible) set of queries that terminate with respect to a given program. In this field, Codish et al. [18] makes a link between backward analysis [19] and termination analysis, and Mesnard [23] presents a constraint-based Termination Inference (cTI) for left termination.

## 7.2 Conclusion

We have introduced the notion of bounded nondeterminism for logic programs and queries. On the one hand, bounded nondeterminism is an extension of the various notions of universal termination extensively studied in the literature. On the other hand, programs and queries that have bounded nondeterminism can be transformed, under determinate conditions, into programs and queries that universally terminate.

We have offered a declarative characterization of bounded nondeterminism in terms of bounded programs and queries. The characterization uses the well-known notions of level mapping and Herbrand model and it results simple and easy to apply in *paper & pencil* proofs. Our effort in adapting an existing automatic approach for left-termination to infer boundedness revealed a weakness of the definition of bounded programs, namely modularity. As a partial answer to such a weakness, we have presented a modular refinement of boundedness, called well-boundedness, that is sound for the class of well-moded programs.

Further work is needed in order to find larger classes for which a modular refinement exists. This is particularly relevant since automation is a prerequisite for the effective implementation of the approach based on the transformation $Ter()$. By adding a derivation length counter, $Ter()$ transforms a bounded program and query into a program and query that universally terminate via any selection rule, yet retaining the same set of refutations.

## References

[1] K. R. Apt. *From Logic Programming to Prolog.* CAR Hoare Series Editor. Prentice Hall, 1997.

[2] K. R. Apt and I. Luitjes. Verification of logic programs with delay declarations. In V.S. Alagar and M. Nivat, editors, *Proc. of AMAST'95*, volume 936 of *Lecture Notes in Computer Science*, pages 66–90, Berlin, 1995. Springer-Verlag, Berlin.

[3] K. R. Apt and D. Pedreschi. Reasoning about termination of pure prolog programs. *Information and Computation*, 106(1):109–157, 1993.

[4] M.A. Bezem. Strong Termination of Logic Programs. *Journal of Logic Programming*, 15(1 & 2):79–98, 1993.

[5] R. N. Bol, K. R. Apt, and J. W. Klop. An analysis of loop checking mechanism for logic programs. *Theoretical Computer Science*, 86(1):35–79, 1991.

[6] C. Braema, B. Le Charlier, S. Modart, and P. Van Hentenryck. Cardinality analysis of prolog. In M. Bruynooghe, editor, *Proc. of the International Logic Programming Symposium*, pages 457—471. MIT Press, 1994.

[7] M. Bruynooghe, H. Vandecasteele, D. A. de Waal, and M. Denecker. Detecting Unsolvable Queries for Definite Logic Programs. *Journal of Functional and Logic Programming*, Special issue 2, 1999.

[8] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The Ciao Prolog system: Reference manual, 2002. System and on-line version of the manual available at `http://clip.dia.fi.upm.es/Software/Ciao/`.

[9] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, second edition, 2002.

[10] D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *Journal of Logic Programming*, 19-20:199–260, 1994.

[11] S. K. Debray and N. W. Lin. Cost analysis of logic programs. *ACM Trans. on Programming Languages and Systems*, 15(5):826–875, 1993.

[12] S.K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin. Estimating the computational cost of logic programs. In B. Le Charlier, editor, *Proc. of Static Analysis Symposium*, number 864 in Lecture Notes in Computer Science, pages 255–265, Berlin, 1994. Springer-Verlag, Berlin.

[13] S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint-based automatic termination analysis for logic programs. *ACM Trans. on Programming Languages and Systems*, 21(6):1136–1195, 1999.

[14] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 8:69–116, 1987.

[15] N. Dershowitz, N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. A general framework for automatic termination analysis of logic programs. *Applicable Algebra in Engineering,Communication and Computing*, 12(1/2):117–156, 2001.

[16] P. Devienne, P. Lebègue, and J.C. Routier. Halting Problem of One Binary Horn Clause is Undecidable. In *STACS '93*, volume 665 of *Lecture Notes in Computer Science*, pages 48–57. Springer-Verlag, Berlin, 1993.

[17] S. Etalle, A. Bossi, and N. Cocco. Termination of well-moded programs. *Journal of Logic Programming*, 38(2):243–257, 1999.

[18] S. Genaim and M. Codish. Inferring termination condition for logic programs using backwards analysis. In *Proc. of Logic for Programming, Artificial Intelligence and Reasoning*, volume 2250 of *Lecture Notes in Computer Science*, pages 685–694, Berlin, 2001. Springer-Verlag, Berlin.

[19] A. King and L. Lu. A backward analysis for constraint logic programs. *Theory and Practice of Logic Programming*, 2(4-5):517–547, 2002.

[20] N. Lindenstrauss. TermiLog: a system for checking termination of queries to logic programs, 1997. `http://www.cs.huji.ac.il/~naomil`.

[21] N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In L. Naish, editor, *Proc. of the International Conference on Logic Programming*, pages 63–77. The MIT Press, 1997.

[22] J. Martin and A. King. Generating efficient, terminating logic programs. In *Proc. of TAPSOFT '97*, number 1214 in Lecture Notes in Computer Science, pages 273–284. Springer-Verlag, Berlin, 1997.

[23] F. Mesnard and U. Neumerkel. Applying static analysis techniques for inferring termination conditions of logic programs. In P. Cousot, editor, *Static Analysis Symposium*, volume 2126 of *Lecture Notes in Computer Science*, pages 93–110, Berlin, 2001. Springer-Verlag, Berlin.

[24] D. Pedreschi and S. Ruggieri. Bounded nondeterminism of logic programs. In D. De Schreye, editor, *Proc. of the International Conference on Logic Programming*, pages 350–364. The MIT Press, 1999.

[25] D. Pedreschi, S. Ruggieri, and J.G. Smaus. Classes of terminating logic programs. *Theory and Practice of Logic Programming*, 2(3):369–418, 2002.

[26] S. Ruggieri. Termination of constraint logic programs. In *Proc. of International Colloquium on Automata Languages and Programming (ICALP '97)*, number 1256 in Lecture Notes in Computer Science, pages 838–848, 1997.

[27] S. Ruggieri. Decidability of Logic Program Semantics and Applications to Testing. *Journal of Logic Programming*, 46(1,2):103–137, 2000.

[28] S. Ruggieri. $\exists$-universal termination of logic programs. *Theoretical Computer Science*, 254(1-2):273–296, 2001.

[29] J.-G. Smaus. *Modes and Types in Logic Programming.* PhD thesis, University of Kent at Canterbury, 1999.

[30] C. Speirs, Z. Somogyi, and H. Søndergaard. Termination Analysis for Mercury. In P. van Hentenryck, editor, *Proc. of Static Analysis Symposium (SAS '97)*, volume 1302 of *Lecture Notes in Computer Science*, pages 160–171. Springer-Verlag, Berlin, 1997.

[31] R.F. Stärk. The theoretical foundations of LPTP (A Logic Program Theorem Prover). *Journal of Logic Programming*, 36(3):241–269, 1998.