Università di Pisa

Dipartimento di Informatica

Technical Report : TR-98-02

exists-universal termination of logic programs

S. Ruggieri

January 26, 1998

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

∃-Universal Termination of Logic Programs

Salvatore Ruggieri

Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy
Tel. +39-50-887243 Fax. +39-50-887226
e-mail: `ruggieri@di.unipi.it`

Abstract. We introduce the notion of ∃-universal termination of logic programs. A program P and a goal G ∃-universally terminate iff there exists a selection rule $\mathcal{S}$ such that every SLD-derivation of $P \cup \{G\}$ via $\mathcal{S}$ is finite. We claim that it is an essential concept for declarative programming, where a crucial point is to associate a terminating control strategy to programs and goals. We show that ∃-universal termination and universal termination via fair selection rules coincide. Then we offer a characterization of ∃-universal termination by defining *fair-bounded* programs and goals. They provide us with a correct and complete method of proving ∃-universal termination. We show other valuable properties of fair-bounded programs and goals, including persistency, modularity, ease of use in *paper & pencil* proofs, automatization of proofs.

Category and Subject Descriptors (ACM CR 1991): D.1.6 [**Programming Techniques**] Logic Programming, F.3.1 [**Logics and Meanings of Programs**] Specifying and Verifying and Reasoning about Programs.

Mathematical Subject Classification (MSC 1991): 68N17 Logic Programming, 68Q60 Specification and Verification of Programs.

Keywords and Phrases: Logic Programs, Universal Termination, Fair Selection Rules, Fair-bounded programs.

1 Introduction

Logic programming is advocated as an ideal support for declarative programming. The milestone work of Kowalski [8] foresaw a separation of concerns for the *logic* and the *control* components of programs. Logic is demanded to programmers, who write *specifications* that can be directly used as *programs*. The generation of a *complete control* is demanded to the underlying logic programming system. By *a complete control,* it is usually meant a selection rule $\mathcal{S}$ such that every logical consequence of a program has a refutation via $\mathcal{S}$. By the Strong Completeness Theorem of SLD-resolution (see Apt [1]), any selection rule is complete in this sense. However, a stronger form of completeness is usually intended, which takes into account termination as well.

Definition 1. By *a complete control for a program P and a goal G,* we mean any selection rule $\mathcal{S}$ such that every SLD-derivation of $P \cup \{G\}$ via $\mathcal{S}$ is finite. □

The ideal situation where logic and control are kept separated is usually contradicted by practical experience. For efficiency reasons, early systems such as Prolog adopted a fixed control, namely a left-to-right selection rule and a depth-first search strategy. Unfortunately, Prolog's control is not complete in the sense of Definition 1. This fact prevents writing programs declaratively, and, in practice, a left-to-right style is customary in any Prolog program. This and other problems with Prolog semantics (including the omission of the occur-check, the use of unsafe negation, the presence of extra-logical predicates, the undisciplined use of cut) motivated the design of more high level, expressive, efficient and practical logic programming systems.

Second generation logic languages adopt more flexible control primitives, which allow for addressing logic and control separately. Program clauses are intended to model the logic of programs, as usual. In addition, programs are augmented by *declarations* or *annotations* that (implicitly or explicitly) specify restrictions on the admissible selection rules. In this stream, we classify NU-Prolog [22], Gödel [7] and the recent Mercury [19] language, among the others. The trend in logic language design is then to achieve full declarativeness by forbidding the use of extra-logical predicates present in Prolog. Real world application problems are tackled providing facilities including type checking, meta-programming, higher order programming, declarative input/output, libraries and modules. At the same time, efficiency is improved by exploiting the large amount of techniques available for code optimizations, coroutining, parallel executions, deterministic executions, safe omission of the occur-check etc.

In this context, termination is a desirable property in its own, and it turns out be essential for compiler optimizations, in presence of program transformations, coroutining and parallel executions. However, existing termination methods (see [18] for a survey) are a step behind, since they mainly reason on the leftmost selection rule.

In this paper, we introduce and investigate ∃- *universal termination* of logic programs. We claim that ∃- *universal termination* is an essential concept for the concerns of separating the development of the *logic* part of programs from the problem of associating a complete control strategy to them.

Definition 2. A logic program P and a goal G ∃-universally terminate iff there exists a selection rule $\mathcal{S}$ such that every SLD-derivation of $P \cup \{G\}$ via $\mathcal{S}$ is finite. □

If P and G ∃-universally terminate then it is possible, at least in principle, to associate to them a complete control. On the other hand, if P and G does not ∃-universally terminate, then no logic programming system can be complete in the sense of Definition 1. Nevertheless, completeness can still be achieved by adopting transformational techniques, or loop checking mechanisms, or, more in general, by modifying the operational interpretation of P and G provided by SLD-resolution.

In this paper, we show that ∃-universal termination coincide with universal termination with respect to fair selection rules. Therefore, fair selection rules are a complete control for every logic program and goal, in the sense of Definition 1. Then we offer a characterization of ∃-universal termination by means of the notions of *fair-bounded* programs and goals, that provide us with a *correct* and *complete* method of proving ∃-universal termination. We show that the notion of *fair-boundedness* is persistent along SLD-derivations, i.e. if a program and a goal are fair-bounded then every SLD-resolvent of them is fair-bounded.

The definition of fair-boundedness is *purely declarative*, in the sense that neither any procedural notion is needed in order to prove a program fair-bounded nor the definition reflects some fixed ordering of the atoms. Proof obligations are *modular*, i.e. each program clause has to be considered in isolation. Moreover, we argue that the method is simple and intuitive in *paper & pencil* proofs. Finally, as a by-result of completeness of the method, we observe that any (possibly automatic) method for proving termination via some selection rule is a sufficient condition for proving a program fair-bounded. This allows us to reuse all existing automatic tools and termination proofs to the purpose of showing fair-boundedness.

Plan of the paper

After some preliminaries, we show in Section 2 that ∃-universal termination and universal termination via fair selection rules coincide. Then we introduce in Section 3 fair-bounded logic programs and goals. In Sections 4 and 5, we show persistency, correctness and completeness of the proof method. In Section 6, we extend the approach to reason on arithmetic built-in's declaratively. In Section 7, well-studied classes of terminating logic programs are related to fair-bounded programs. In Section 8, we discuss the problem of automatically inferring fair-boundedness. Finally, we report some recent related works dealing with termination via selection rules other than leftmost's and summarize the contribution of the paper.

Preliminaries

Throughout the paper we follow the standard notation of Apt [1]. A clause is a construct $A \leftarrow B_1, \ldots, B_n$, with $n \geq 0$, where A and $B_1, \ldots, B_n$ are atoms. A logic program is a set of clauses. A goal is a construct $\leftarrow B_1, \ldots, B_n$, where $B_1, \ldots, B_n$ are atoms. B_P the Herbrand base on L_P, the language generated by P. U_P is the set of ground terms on L_P, i.e. the Herbrand universe. A Herbrand interpretation I is a subset of B_P. We recall that for ground atoms $A_1, \ldots, A_n$, $I \models A_1, \ldots, A_n$ holds iff $\{A_1, \ldots, A_n\} \subseteq I$. Moreover, for (not necessarily ground) atoms $A_1, \ldots, A_n$, $I \models \exists\ A_1, \ldots, A_n$ holds iff there exists a ground instance $A'_1, \ldots, A'_n$ of $A_1, \ldots, A_n$ such that $I \models A'_1, \ldots, A'_n$. $ground(P)$ denotes the set of ground instances of clauses from P. Analogously, $ground(G)$ denotes the set of ground instances of a goal G. T_P is the *immediate consequence operator* defined as follows: $T_P(I) = \{A \mid A \leftarrow B_1, \ldots, B_n \in ground(P), I \models B_1, \ldots, B_n\}$. For an ordinal α, $T_P \uparrow \alpha$ and $T_P \downarrow \alpha$ are the upward and downward ordinal powers of T_P, respectively. A selection rule $\mathcal{S}$ is a function that assigns to a prefix of a SLD-derivation whose last goal is G an atom in G, called the *selected* atom. A selection rule $\mathcal{F}$ is *fair* if for every SLD-derivation ξ via $\mathcal{F}$ either ξ is finite or for every atom A in ξ, (some further instantiated version of) A is eventually selected. Finally, N is the set of natural numbers.

2 $\exists$-universal termination and fair selection rules

First, we show that $\exists$-universal termination coincide with termination via fair selection rules. Therefore, any fair selection rule is a complete control for every logic program and goal.

Theorem 3. *A logic program P and goal G $\exists$-universally terminate iff every SLD-derivation of $P \cup \{G\}$ via any fair selection rule is finite.*

Proof. The *if* part is immediate. Conversely, suppose that there exists an infinite SLD-derivation ξ via a fair selection rule $\mathcal{F}$. Let $\mathcal{S}$ be any other selection rule. Since $\mathcal{F}$ is fair, by reasoning as in the Switching Lemma [10, Lemma 9.1], we can switch the order of selection of atoms in ξ accordingly to $\mathcal{S}$, thus obtaining an infinite SLD-derivation of $P \cup \{G\}$ via $\mathcal{S}$. $\square$

Starting from this characterization, we propose in the rest of this paper a proof method for $\exists$-universal termination which is indeed a method for proving universal termination with respect to fair selection rules. Correctness and completeness of the method are proved by exploiting well-known properties of fair selection rules. In particular, the following relation between finitely failed fair SLD-trees and downward ordinals of the *immediate consequence operator* will be useful.

Lemma 4. *Let P be a logic program, and $\leftarrow Q$ a goal. Then $T_P \downarrow i \not\models \exists\, Q$ for some $i \in N$ iff every SLD-derivation of $P \cup \{\leftarrow Q\}$ via any fair selection rule is failed.*

Proof. Consider the program $P' = P \cup \{\mathtt{p} \leftarrow Q\}$ where $\mathtt{p}$ is a fresh predicate symbol. By definition of T_P, for any $i \geq 0$, $T_P \downarrow i \not\models \exists\, Q$ iff $\mathtt{p} \notin T_{P'} \downarrow i+1$ iff $\mathtt{p} \notin T_{P'} \downarrow \omega$. We recall (see [1, Theorem 5.6]) that $\mathtt{p} \notin T_{P'} \downarrow \omega$ iff every SLD-derivation of $P' \cup \{\leftarrow \mathtt{p}\}$ via any fair selection rule is failed. Since $\mathtt{p}$ is a fresh symbol, we conclude that $T_P \downarrow i \not\models \exists\, Q$ for some $i \geq 0$ iff every SLD-derivation of $P \cup \{\leftarrow Q\}$ via any fair selection rule is failed. $\square$

3 Fair-bounded logic programs

In this section we define the class of fair-bounded logic programs and goals. First, we introduce the concept of level mappings.

Definition 5. Given a logic program P, a *level mapping* for P is a function $|\ | : B_P \to N^\infty$ of ground atoms to N^∞, where $N^\infty = N \cup \{\infty\}$. $\square$

Intuitively, level mappings play the role of termination functions. However, differently from more standard definitions (see [18]), we included ∞ in the codomain of level mappings, as a means to model non-termination and uninteresting instances of program clauses. We extend the $>$ order on naturals to a relation $\rhd$ on N^∞.

Definition 6. We define the relation $n \rhd m$ for $m, m \in N^\infty$ as follows:

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

We write $n \trianglerighteq m$ iff $n \rhd m$ or $n = m$. $\square$

Therefore, $\infty \rhd m$ for every $m \in N^\infty$. We are now in the position to introduce *fair-bounded* logic programs.

Definition 7. Let P be a logic program, I a Herbrand interpretation and $|\ |$ a level mapping for P. P is called *fair-bounded by* $|\ |$ *and* I iff I is a model of P such that for every $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$:

(a) $I \models B_1, \ldots, B_n$ *implies* for every $i \in [1, n]$ $|A| \rhd |B_i|$, and
(b) $I \not\models B_1, \ldots, B_n$ *implies* there exists $i \in [1, n]$ $I \not\models B_i \wedge |A| \rhd |B_i|$.
$\square$

Let us focus the attention on some valuable properties that follow directly from Definition 7. First, we observe that the proof obligations are *modular*, in the sense that program clauses are taken into consideration separately, one at a time. Second, the hypothesis of conditions *(a)* and *(b)* are *mutually exclusive*. Third, the notion of fair-boundedness is *purely declarative*, in the sense that neither any procedural notion is needed in order to prove a program fair-bounded nor the definition reflects some fixed ordering of the atoms.

Next definition extends fair-boundedness to goals.

Definition 8. Let G be goal, I a Herbrand interpretation and $|\ |$ a level mapping. G is called *fair-bounded by* $|\ |$ *and* I iff there exists $k \in N$ such that for every $\leftarrow A_1, \ldots, A_n \in ground(G)$:

(a) $I \models A_1, \ldots, A_n$ *implies* for every $i \in [1, n]$ $k \rhd |\, A_i\, |$, and
(b) $I \not\models A_1, \ldots, A_n$ *implies* there exists $i \in [1, n]$ $I \not\models A_i \wedge k \rhd |A_i|$.
$\square$

Example 1. The following program `ProdCons` abstracts a (concurrent) system composed of a producer and a consumer.

```
(s)    system(N) ← prod(Bs), cons(Bs, N).

(p1)   prod([s(0) | Bs])) ← prod(Bs).
(p2)   prod([s(s(0)) | Bs])) ← prod(Bs).
       prod([]).

(c)    cons([D | Bs], s(N)) ← cons(Bs, N), wait(D).
       cons([], 0).

       wait(0).
(w)    wait(s(D)) :- wait(D).
```

For notational convenience, we identify the term $\mathtt{s}^n\mathtt{(0)}$ with the natural number n. Intuitively, `prod` is the producer of a non-deterministic sequence of 1's and 2's, and `cons` the consumer of the sequence. The shared variable `Bs` in clause *(s)* plays the role of an unbounded buffer. Moreover, since it is realistic to assume that consumption depends on n, we model consumption by `wait`. The overall system $\leftarrow$ $\mathtt{system}(N)$ stops when `cons` has consumed N messages.

First of all, we observe that `ProdCons` and a goal $\leftarrow$ $\mathtt{system}(n)$ have infinite SLD-derivations via the leftmost and the rightmost selection rules. Actually, viewed as a concurrent system, the program needs the assumption of fairness in order to terminate.

Let us show that `ProCons` is fair-bounded. First, we define $size(t)$ as the number of function symbols (excluding constants) appearing in a ground term t. Moreover, we recall the *list-length* and *list-max* functions, that map ground terms into natural numbers as follows:

$$\begin{aligned} llen([x|xs]) &= \ llen(xs)+1, \\ llen(f(x_1,\ldots,x_n)) &= \ 0 \text{ if } f \neq \ [\,.\,|\,.\,]. \\ lmax([x|xs]) &= \ max\{lmax(xs), size(x)\}, \\ lmax(f(x_1,\ldots,x_n)) &= \ 0 \text{ if } f \neq \ [\,.\,|\,.\,]. \end{aligned}$$

Note that for a ground list xs, $llen(xs)$ equals its length and $lmax(xs)$ equals the maximum size of an element in xs. Then we define:

$$\begin{aligned} I = \quad & [\mathtt{system(N)}] \\ & \cup \{ \ \mathtt{prod}(bs) \mid bs \text{ list of 1's and 2's } \ \} \\ & \cup \{ \ \mathtt{cons}(bs,n) \mid llen(bs) = size(n) \ \} \\ & \cup [\mathtt{wait(X)}] \\ |\mathtt{system}(n)| &= size(n)+3 \\ |\mathtt{prod}(bs)| &= llen(bs) \\ |\mathtt{cons}(bs,\ n)| &= size(n)+lmax(bs) \ \text{ if } \mathtt{cons}(bs,\ n) \in I \\ |\mathtt{cons}(bs,\ n)| &= size(n) \ \text{ if } \mathtt{cons}(bs,\ n) \notin I \\ |\mathtt{wait}(t)| &= size(t). \end{aligned}$$

Let us show the proof obligations of Definition 7. Those for unit clauses are trivial. Let us concentrate on recursive clauses *(s)*, *(p1)*, *(p2)*, *(c)*, and *(w)*.

(w) I is obviously a model of *(w)*. In addition, $\mid \texttt{wait(s(}d\texttt{))} \mid = size(d) + 1 > size(d) = \mid \texttt{wait(}d\texttt{)} \mid$. This implies *(a, b)*.

(c) Consider a ground instance

$$\texttt{cons([}d\ \texttt{|}\ bs\texttt{], s(}n\texttt{))} \leftarrow \texttt{cons(}bs\texttt{, }n\texttt{), wait(}d\texttt{)}$$

of *(c)*. If $I \models \texttt{cons(}bs\texttt{, }n\texttt{), wait(}d\texttt{)}$ then $llen(bs) = size(n)$, and then:

$$llen(\texttt{[}d\ \texttt{|}\ bs\texttt{]}) = llen(bs) + 1 = size(n) + 1 = size(\texttt{s(}n\texttt{)}),$$

i.e. $I \models \texttt{cons([}d\ \texttt{|}\ bs\texttt{], s(}n\texttt{))}$. Therefore, I is a model of *(c)*. Let us show proof obligations *(a, b)* of Definition 7.

(a) Suppose that $I \models \texttt{cons(}bs\texttt{, }n\texttt{), wait(}d\texttt{)}$. We have showed that $I \models \texttt{cons([}d\ \texttt{|}\ bs\texttt{], s(}n\texttt{))}$. We calculate:

$$\begin{aligned} & |\texttt{cons([}d\ \texttt{|}\ bs\texttt{], s(}n\texttt{))}| \\ = {} & size(n) + 1 + max\{lmax(bs), size(d)\} \\ > {} & size(n) + lmax(bs) \\ = {} & |\texttt{cons(}bs\texttt{, }n\texttt{)}|. \end{aligned}$$

and

$$\begin{aligned} & |\texttt{cons([}d\ \texttt{|}\ bs\texttt{], s(}n\texttt{))}| \\ = {} & size(n) + 1 + max\{lmax(bs), size(d)\} \\ > {} & size(d) \\ = {} & |\texttt{wait(}d\texttt{)}|. \end{aligned}$$

These two inequalities show that *(a)* holds.

(b) If $I \not\models \texttt{cons(}bs\texttt{, }n\texttt{), wait(}d\texttt{)}$ then necessarily $I \not\models \texttt{cons(}bs\texttt{, }n\texttt{)}$. This and

$$\begin{aligned} & |\texttt{cons([}d\ \texttt{|}\ bs\texttt{], s(}n\texttt{))}| \\ \geq {} & size(n) + 1 \\ > {} & size(n) \\ = {} & \quad \{\ I \not\models \texttt{cons(}bs\texttt{, }n\texttt{)}\ \} \\ & |\texttt{cons(}bs\texttt{, }n\texttt{)}|. \end{aligned}$$

show *(b)*.

(p1) I is obviously a model of *(p1)*. Moreover:

$$|\texttt{prod([s(0) | }bs\texttt{])}| = llen(bs) + 1 \ > \ llen(bs) \ = \ |\texttt{prod(}bs\texttt{)}|.$$

implies *(a)* and *(b)*.

(p2) This case is analogous to the previous one.

(s) Consider a ground instance

$$\texttt{system}(n) \ \leftarrow \texttt{prod}(bs)\texttt{, cons}(bs\texttt{, } n\texttt{)}$$

of *(s)*. Obviously I is a model of *(s)*. Let us show *(a,b)*.

(a) Suppose that $I \models \texttt{prod}(bs)\texttt{, cons}(bs\texttt{, } n\texttt{)}$. Then bs is a list of 1's and 2's. This implies $lmax(bs) = 2$. Moreover, we have that $llen(bs) = size(n)$. We calculate:

$$\begin{aligned} |\texttt{system}(n)| &= size(n) + 3 \\ &> \quad \{\ llen(bs) = size(n)\ \} \\ &\quad llen(bs) \\ &= |\texttt{prod}(bs)| \end{aligned}$$

and

$$\begin{aligned} |\texttt{system}(n)| &= size(n) + 3 \\ &> \quad \{\ lmax(bs) = 2\ \} \\ &\quad |\texttt{cons}(bs\texttt{, } n)|. \end{aligned}$$

These two inequalities show *(a)*.

(b) Suppose that $I \not\models \texttt{prod}(bs)\texttt{, cons}(bs\texttt{, } n\texttt{)}$. We distinguish two cases. If $I \not\models \texttt{cons}(bs\texttt{, } n\texttt{)}$ then:

$$|\texttt{system}(n)| \ = \ size(n) + 3 \ > \ size(n) \ = \ |\texttt{cons}(bs\texttt{, } n)|.$$

If $I \not\models \texttt{prod}(bs)$ and $I \models \texttt{cons}(bs\texttt{, } n\texttt{)}$ then:

$$\begin{aligned} |\texttt{system}(n)| &= size(n) + 3 \\ &> \quad \{\ I \models \texttt{cons}(bs\texttt{, } n) \text{ implies } llen(bs) = size(n)\ \} \\ &\quad llen(bs) \\ &= |\texttt{prod}(bs)|. \end{aligned}$$

We conclude this example by noting that for every $n \in N$ the goal $\leftarrow \texttt{system}(n)$ is fair-bounded by $|\ |$ and I. □

We claim that proving that a program is fair-bounded is simple and practical in *paper & pencil* proofs. In fact, proof obligations restrict to consider ground instances of clauses and goals. Substitutions and non-ground terms have not to be taken into account, since the method automatically lifts up to non-ground goals G by considering every ground instance of G. Moreover, the various tools used for proving fair-boundedness have intuitive interpretations:

The *level mapping* $|\ |$ is a decreasing function that ensures termination.

∞ allows for excluding *ill-typed atoms* and *non-terminating atoms* from the termination analysis. In fact, if $|A| = \infty$ then *(a, b)* in Definition 7 are trivially satisfied. The need for reasoning on a subset of goals is motivated by the fact that logic programs are untyped, and then goals may have atoms that in the intended interpretation of the programmer are ill-typed. Another reason for introducing ∞ is the fact that a program can terminate for a strict subset of ground atoms only. As an example, consider the following simple program:

```
q ← q.
```

By defining

$$I = \emptyset, \quad |\mathtt{q}| = \infty, \quad |\mathtt{p}| = 0,$$

it is readily checked that the program and the goal $\leftarrow$ `p, q` are fair-bounded by I and $|\ |$. Later on, we will show that this implies that the program and the goal $\leftarrow$ `p, q` ∃-universally terminate. On the contrary, the program and the goal $\leftarrow$ `q` have only infinite SLD-derivations. In particular, we observe that they are not fair-bounded by any I and $|\ |$. In fact, by Definition 7 *(a, b)*, $|\mathtt{q}| \rhd |\mathtt{q}|$ must hold. Thus $|\mathtt{q}| = \infty$. This implies, by Definition 8 *(a, b)*, that $\leftarrow$ `q` cannot be fair-bounded by the same $|\ |$ and I. Summarizing, ∞ allows us to concentrate on interesting sets of goals by excluding *ill-typed atoms* and *non-terminating atoms* from the termination analysis.

Finally, let us discuss in more detail the meaning of proof obligations *(a)* and *(b)* in Definition 7. Consider a ground instance of a clause:

$$A \leftarrow B_1, \ldots, B_n$$

If the body $B_1, \ldots, B_n$ is true in the model I, then there might exist a SLD-refutation for it. *(a)* is then intended to bound the length of the refutation.

If the body is not true in the model I, then it cannot have a refutation. In this case, termination actually means that there is an atom in the body that has a finitely failed SLD-tree. *(b)* is then intended to bound the depth of the finitely failed SLD-tree.

Those intuitions about the meaning of *(a)* and *(b)* clarify why in Example 1 the level mapping for the `cons` atoms distinguishes two cases. When `cons(`bs`, `n`)` is in I, then we must bound the length of a possible SLD-refutation, while when it is not in I we must bound the depth of a finitely failed SLD-tree. However, since the proof method is purely declarative, in practice we do not reason about operational notions such as refutations and SLD-trees, but on models and level mappings.

Finally, the *model* I is a description of some property of the declarative interpretation of the program, namely the least Herbrand model. However, as a consequence of the intended meaning of proof obligation *(b)* in Definition 7, the complement of I is necessarily included in the finite failure set of the program. This intuition will be formally stated later in Lemma 10.

4 Termination correctness

In this section, we show that if a program and a goal are fair-bounded then they ∃-universally terminate. Therefore, fair-boundedness provides us with a correct method for proving ∃-universal termination. First of all, we show that the notion of fair-boundedness is persistent along SLD-derivations.

Lemma 9 (Persistency). *Let P be a program and $\leftarrow Q$ a goal both fair-bounded by $|\ |$ and I. Every SLD-resolvent $\leftarrow Q'$ of $P \cup \{\leftarrow Q\}$ is fair-bounded by $|\ |$ and I.*

Proof. First of all, we observe that for any substitution θ, thanks to Definition 8, $\leftarrow Q\theta$ is fair-bounded by $|\ |$ and I by using some *bound* $\in N$. Let θ be now the mgu of the selected atom in Q and the input clause head. Assume that $\leftarrow Q\theta = \ \leftarrow A_1, \ldots, A_n$, and that $c : A_k \leftarrow B_1, \ldots, B_m$ is the instantiation by θ of the input clause. Then $\leftarrow Q'$ is

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

Let now $Q_{in} = A'_1, \ldots, A'_{k-1}, B'_1, \ldots, B'_m, A'_{k+1}, \ldots, A'_n$ be a ground instance of Q'. Then there exists

$$A'_1, \ldots, A'_n$$

ground instance of $Q\theta$, with $A'_k \leftarrow B'_1, \ldots, B'_m$ ground instance of c.

Let us show the proof obligations of Definition 8.

(a) Suppose that $I \models Q_{in}$. Then $I \models B'_1, \ldots, B'_m$ and, since I is a model of P, $I \models A'_1, \ldots, A'_n$. By Definition 8 *(a)*, *bound* $\rhd |A'_i|$ for every $i \in [1, n]$. Moreover, since $I \models B'_1, \ldots, B'_m$, then *bound* $\rhd |A'_k| \rhd |B'_i|$ for $i \in [1, m]$.

(b) Suppose that $I \not\models Q_{in}$. We distinguish two cases.

(b1) If $I \models A'_1, \ldots, A'_n$ then *bound* $\rhd |A'_k|$ by Definition 8 *(a)*. Moreover, $I \not\models B'_1, \ldots, B'_m$. By Definition 7 *(b)* there exists $i \in [1, m]$ such that $|A'_k| \rhd |B'_i|$ and $I \not\models B'_i$. We conclude that $I \not\models B_i$ and *bound* $\rhd |A'_k| \rhd |B'_i|$.

(b2) If $I \not\models A'_1, \ldots, A'_n$ then by Definition 8 *(b)*, there exists $i \in [1, n]$ such that $I \not\models A'_i$ and *bound* $\rhd |A'_i|$. We distinguish two cases.

If $i \neq k$ then we are finished, since A'_i is in Q_{in}.

Suppose, on the contrary, that $I \not\models A'_k \wedge$ *bound* $\rhd |A'_k|$. Since I is a model of P, $I \not\models A'_k$ implies $I \not\models B'_1, \ldots, B'_m$. By Definition 7 *(b)*, there exists $i \in [1, m]$ such that $|A'_k| \rhd |B'_i|$ and $I \not\models B'_i$. We then conclude that $I \not\models B_i$ and *bound* $\rhd |A'_k| \rhd |B'_i|$. □

The following is a property of fair-bounded programs that will be useful later on.

Lemma 10. *Let P be a program fair-bounded by $|\ |$ and I. Then for every $A \notin I$, if $|A| \neq \infty$ then $A \notin T_P \downarrow |A| + 1$.*

Proof. Suppose that $A \notin I$ and $|A| \neq \infty$. The proof proceeds by induction

($|A| = 0$) We claim that no $A \leftarrow B_1, \ldots, B_n$ is in *ground*(P). Otherwise, since I is a model of P and $A \notin I$ then $I \not\models B_1, \ldots, B_n$. By Definition 7 *(b)* there exists $i \in [1, n]$ such that $0 \rhd |B_i|$, which is impossible. By definition of T_P, we conclude that $A \notin T_P \downarrow 1$.

($|A| > 0$) We distinguish two cases.

Suppose that there is no $A \leftarrow B_1, \ldots, B_n$ in *ground*(P). By definition of T_P, we have that $A \notin T_P \downarrow 1$. By monotonicity of T_P, we conclude $A \notin T_P \downarrow |A|+1$.

Suppose, on the contrary, that there exists $A \leftarrow B_1, \ldots, B_n$ in *ground*(P). Since I is a model of P and $A \notin I$ then $I \not\models B_1, \ldots, B_n$. By Definition 7 *(b)* there exists $i \in [1,n]$ such that $I \not\models B_i$ and $|A| \rhd |B_i|$. By induction hypothesis, we have that $B_i \notin T_P \downarrow |B_i|+1$. By monotonicity of T_P, we observe that $B_i \notin T_P \downarrow |A|$. By definition of T_P, we conclude that $A \notin T_P \downarrow |A|+1$.

□

In what follows we shall use the multiset ordering. A *multiset*, sometimes called *bag*, is an unordered sequence. We denote a multiset consisting of elements $a_1, \ldots, a_n$ by *bag* $(a_1, \ldots, a_n)$. Given a (non-reflexive) ordering $<$ on a set W, the *multiset ordering over* $(W, <)$ is an ordering on finite multisets of the set W. It is defined as the transitive closure of the relation in which X is smaller than Y if X can be obtained from Y by replacing an element a of Y by a finite (possibly zero) number of elements each of which is smaller than a in the ordering $<$. Finally, we write $X \succeq_m Y$ iff $X \succ_m Y$ or $X = Y$.

It is well-known (see e.g. Dershowitz [5]) that multiset ordering over a well-founded ordering is again well-founded. In particular, the multiset ordering over the set of natural numbers with their usual ordering is well-founded. Next we associate a finite multiset over N to fair-bounded goals.

Definition 11. Let $G = \leftarrow A_1, \ldots, A_n$ be a goal fair-bounded by $|\ |$ and I. We define the sets $|G|_i^I$ for $i \in [1,n]$ as follows:

$$|G|_i^I = \{|A'_i| \mid \leftarrow A'_1, \ldots, A'_n \in \mathit{ground}(G) \wedge I \models A'_1, \ldots, A'_n \}.$$

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

$$|G|^I = \mathit{bag}(\mathit{max}|G|_1^I, \ldots, \mathit{max}|G|_n^I),$$

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

We observe that the definition is well-formed. By Definition 8, the sets $|G|_i^I$ for $i \in [1,n]$ are finite, and then there exists the maximum (which is 0 in case of empty sets). The following lemma shows a crucial relation between a goal and its SLD-resolvents.

Lemma 12. *Let P be a program and $\leftarrow Q$ a goal both fair-bounded by $|\ |$ and I. For every SLD-resolvent $\leftarrow Q'$ of $P \cup \{\leftarrow Q\}$, we have that*

(i) $|\leftarrow Q|^I \succeq_m |\leftarrow Q'|^I$, *and*
(ii) *if $I \models \exists\, Q'$ then* $|\leftarrow Q|^I \succ_m |\leftarrow Q'|^I$.

Proof. First of all, we observe that for every substitution θ,

$$|\leftarrow Q|^I \succeq_m |\leftarrow Q\theta|^I. \tag{1}$$

In fact, by Definition 11 $|\leftarrow Q|_i^I \supseteq |\leftarrow Q\theta|_i^I$ holds for $i \in [1,n]$, where n is the number of atoms in Q.

Let θ be now the mgu of the selected atom in Q and the input clause head. Assume that $\leftarrow Q\theta = \leftarrow A_1, \ldots, A_n$, and that $c : A_k \leftarrow B_1, \ldots, B_m$ is the instantiation by θ of the input clause. Then $\leftarrow Q'$ is

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

First, suppose that $I \not\models \exists\, Q'$. Then we have to show only *(i)*, which is immediate by observing that $|\leftarrow Q|^I \succeq_m bag() = |\leftarrow Q'|^I$.
On the other hand, assume now that $I \models \exists\, Q'$. Then for every $i \in [1, n+m-1]$:

$$|\leftarrow Q'|_i^I \neq \emptyset. \tag{2}$$

Let now $Q_{in} = A'_1, \ldots, A'_{k-1}, B'_1, \ldots, B'_m, A'_{k+1}, \ldots, A'_n$ be a ground instance of Q' such that $I \models Q_{in}$. Then there exists

$$A'_1, \ldots, A'_n$$

ground instance of $Q\theta$, with $A'_k \leftarrow B'_1, \ldots, B'_m$ ground instance of c. As a consequence for every $i \in [1,n], i \neq k$, we have $|A'_i| \in |\leftarrow Q\theta|_i^I$. This implies:

$$\text{for } i \in [1, k-1] \;\; max|\leftarrow Q'|_i^I \leq max|\leftarrow Q\theta|_i^I, \tag{3}$$
$$\text{for } i \in [k+1, n] \;\; max|\leftarrow Q'|_{i+m-1}^I \leq max|\leftarrow Q\theta|_i^I. \tag{4}$$

Moreover, since $I \models B'_1, \ldots, B'_m$, by Definition 7 *(a)*, we have that $|A'_k| \rhd |B'_i|$ for every $i \in [1,m]$. By the assumption that $\leftarrow Q$ is fair-bounded, we have that $|A'_k| \in N$ and then $|A'_k| > |B'_i|$ for every $i \in [1,m]$. Summarizing:

$$\text{for } i \in [1,m] \;\; \forall\, x \in |\leftarrow Q'|_{k+i-1}^I \; \exists\, y \in |\leftarrow Q\theta|_k^I \; y > x. \tag{5}$$

In conclusion, we calculate:

$$
\begin{array}{ll}
 & |\leftarrow Q|^I \\
\succeq_m & \quad \{\ (1)\ \} \\
 & |\leftarrow Q\theta|^I \\
= & \quad \{\ \text{Definition 11}\ \} \\
 & bag(max|\leftarrow Q\theta|_1^I, \ldots, max|\leftarrow Q\theta|_k^I, \ldots, max|\leftarrow Q\theta|_n^I) \\
\succ_m & \quad \{\ (2-5) \text{ and the fact that} \\
 & \quad\quad \forall\, x \in S \neq \emptyset\; \exists y \in R.\ y > x \textit{ implies } max\ R > max\ S\ \} \\
 & bag(max|\leftarrow Q'|_1^I, \ldots, max|\leftarrow Q'|_{n+m-1}^I) \\
= & \quad \{\text{Definition 11}\} \\
 & |\leftarrow Q'|^I
\end{array}
$$

which implies *(i-ii)*. □

Next Theorem shows that no SLD-derivation of fair-bounded programs and goals is infinite via a fair selection rule.

Theorem 13. *Let P be a program and $\leftarrow Q$ a goal both fair-bounded by $|\ |$ and I. Then every SLD-derivation of $P \cup \{\leftarrow Q\}$ via a fair selection rule is finite.*

Proof. Suppose that there exists an infinite SLD-derivation $\leftarrow Q_1, \ldots, \leftarrow Q_i, \ldots$ of $P \cup \{\leftarrow Q\}$ via a fair selection rule. By Lemma 9, every $\leftarrow Q_i$ is fair-bounded by $|\ |$ and I. We distinguish two cases depending whether or not for every $i \geq 1, I \models \exists\, Q_i$.

Suppose that $I \not\models \exists Q_i$ for some $i \geq 1$. By Definition 8 *(b)*, there exists $k \in N$ such that for every $A_1, \ldots, A_n$ ground instance of Q_i, there exists $j \in [1, n]$ such that $I \not\models A_j$ and $k \rhd |A_j|$. By Lemma 10, $A_j \notin T_P \downarrow |A_j| + 1$. By monotonicity of T_P, $A_j \notin T_P \downarrow k$. Summarizing, $T_P \downarrow k \not\models \exists\, Q_i$. By Lemma 4, every fair SLD-derivation of $P \cup \{\leftarrow Q_i\}$ is failed, hence finite. This contradicts the assumption that there exists an infinite fair SLD-derivation.

Suppose now that for every $i \geq 1, I \models \exists\, Q_i$.

By Lemma 12 *(ii)*, $|\leftarrow Q_1|^I \succ_m \ldots \succ_m |\leftarrow Q_i|^I \succ_m \ldots$ is an infinite descending chain of bags over naturals. This is impossible since the finite multiset ordering over naturals is well-founded. □

We are in the position to state correctness of the proposed proof method.

Theorem 14 (Termination Correctness). *Let P be a program and G a goal both fair-bounded by $|\ |$ and I. Then P and G $\exists$-universally terminate.*

Proof. By Theorem 13, every SLD-derivation of $P \cup \{G\}$ via a fair selection rule is finite. By Theorem 3, we conclude that P and G ∃-universally terminate. □

Example 2. Let us consider Example 1. We showed that `ProdCons` and the goal $\leftarrow$ `system(`n`)` are both fair-bounded by $|\ |$ and I, where $n \in N$. By Theorem 13, we conclude that every fair SLD-derivation of `ProdCons` $\cup \{\leftarrow$ `system(`n`)`$\}$ is finite. □

We conclude this section with a corollary showing that upward and downward ordinal closures of fair-bounded programs coincide on $\{A \mid |A| \neq \infty\}$.

Corollary 15. *Let P be a program fair-bounded by $|\ |$ and I. For every $A \in B_P$ such that $|A| \neq \infty$, we have that*

$$A \in T_P \downarrow \omega \quad \textit{iff} \quad A \in T_P \uparrow \omega.$$

Proof. Since $|A| \in N$, we have that the goal $\leftarrow A$ is fair-bounded. By Theorem 13, every SLD-derivation of $P \cup \{\leftarrow A\}$ via a fair selection rule $\mathcal{S}$ is finite. Then either there exists a SLD-refutation or every SLD-derivation via $\mathcal{S}$ is failed. We recall that there exists a refutation of $P \cup \{\leftarrow A\}$ iff $A \in T_P \uparrow \omega$, and that every SLD-derivation via $\mathcal{S}$ is failed iff $A \notin T_P \downarrow \omega$ (see Lemma 4). This implies that $A \in T_P \downarrow \omega$ iff $A \in T_P \uparrow \omega$. □

5 Termination completeness

In this section, we show that if P and G $\exists$-universally terminate then they are fair-bounded by some $|\ |$ and I. Therefore, fair-boundness is a correct and complete proof method for $\exists$-universal termination. We start with a simple observation.

Lemma 16. *For every logic program P, $T_P \downarrow \omega$ is a model of P.*

Proof. By monotonicity of T_P, we have that for every $i \geq 0$, $T_P \downarrow \omega \subseteq T_P \downarrow i$. As a consequence, $T_P(T_P \downarrow \omega) \subseteq T_P \downarrow i+1$, for every $i \geq 0$. This implies $T_P(T_P \downarrow \omega) \subseteq T_P \downarrow \omega$, i.e. $T_P \downarrow \omega$ is a model of P. $\square$

Next, we introduce two further definitions.

Definition 17. For a program P and a goal G, we define $length_{\mathcal{S}}^{P}(G)$ as ∞ if there exists an infinite SLD-derivation of $P \cup \{G\}$ via the selection rule $\mathcal{S}$, and as the maximum length of a SLD-derivation of $P \cup \{G\}$ via $\mathcal{S}$ if no infinite SLD-derivation via $\mathcal{S}$ exists.

We say that a selection rule $\mathcal{S}$ is *closed under substitution* if whenever $\mathcal{S}$ selects the i^{th} atom in a goal G then $\mathcal{S}$ selects the i^{th} atom in every instance of G. $\square$

We observe that the definition of $length_{\mathcal{S}}^{P}$ is well-formed. In fact, since SLD-trees are finitely branching, by König's Lemma if there is no infinite SLD- derivation of $P \cup \{G\}$ via $\mathcal{S}$ then there are finitely many SLD-derivations of $P \cup \{G\}$, hence the maximum length exists. Another useful property of $length_{\mathcal{S}}^{P}$ is the following.

Lemma 18. *Let P be a logic program, $\leftarrow Q$ a goal and $\mathcal{S}$ a selection rule closed under substitution. Then*

(i) for every $\leftarrow Q'$ SLD-resolvent of $P \cup \{\leftarrow Q\}$ via $\mathcal{S}$

$$length_{\mathcal{S}}^{P}(\leftarrow Q) \rhd length_{\mathcal{S}}^{P}(\leftarrow Q'),$$

(ii) for every Q' instance of Q, $length_{\mathcal{S}}^{P}(\leftarrow Q) \trianglerighteq length_{\mathcal{S}}^{P}(\leftarrow Q')$.

Proof. *(i)* is immediate by definition of $length_{\mathcal{S}}^{P}$. Let us show *(ii)*. Consider a SLD-resolvent $\leftarrow Q_1'$ of $P \cup \{\leftarrow Q'\}$ via $\mathcal{S}$. Since $\leftarrow Q'$ is an instance of $\leftarrow Q$ and $\mathcal{S}$ is closed under substitution, there exists a SLD-resolvent $\leftarrow Q_1$ of $P \cup \{\leftarrow Q\}$ via $\mathcal{S}$ such that $\leftarrow Q_1'$ is an instance of $\leftarrow Q_1$. In general, for every SLD-derivation $\leftarrow Q', \leftarrow Q_1', \ldots, \leftarrow Q_n', \ldots$ via $\mathcal{S}$ there exists

$$\leftarrow Q, \leftarrow Q_1, \ldots, \leftarrow Q_n, \ldots$$

(prefix of a) SLD-derivation via $\mathcal{S}$ such that $\leftarrow Q_i'$ is an instance of $\leftarrow Q_i$ for $i \geq 1$. Therefore, $length_{\mathcal{S}}^{P}(\leftarrow Q) \trianglerighteq length_{\mathcal{S}}^{P}(\leftarrow Q')$. $\square$

We are now in the position to show that fair-boundedness is a complete termination proof method with respect to fair selection rules.

Lemma 19. *Let P be a logic program and $\mathcal{F}$ a fair selection rule. Then there exist $|\ |$ and I such that*

(i) *P is fair-bounded by $|\ |$ and I, and*
(ii) *for every $A \in B_P$, $|A| \in N$ iff every SLD-derivation of $P \cup \{\leftarrow A\}$ via $\mathcal{F}$ is finite.*

Proof. We define $I = T_P \downarrow \omega$ and $|A| = length^P_{\mathcal{F}}(\leftarrow A)$.

First, we observe that *(ii)* is immediate by definition of $|\ |$. Moreover, we can assume without loss of generality that $\mathcal{F}$ is closed under substitution. In fact, by Theorem 3 every SLD-derivation of $P \cup \{\leftarrow A\}$ via $\mathcal{F}$ is finite iff every SLD-derivation of $P \cup \{\leftarrow A\}$ via a fair selection rule closed under substitution is finite.

Let us now consider *(i)*. We show the proof obligations of Definition 7. By Lemma 16, I is a model of P. Consider now $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$.

(a) Suppose that $I \models B_1, \ldots, B_n$. By definition of I, this implies that for every $k \geq 0, T_P \downarrow k \models B_1, \ldots, B_n$. By Lemma 4, we conclude that there exists at least one non-failed SLD-derivation ξ of $P \cup \{\leftarrow B_1, \ldots, B_n\}$ via $\mathcal{F}$. We claim that for $i \in [1, n]$:

$$length^P_{\mathcal{F}}(\leftarrow B_1, \ldots, B_n) \trianglerighteq length^P_{\mathcal{F}}(\leftarrow B_i). \tag{6}$$

In fact, consider a SLD-derivation ξ' of $P \cup \{\leftarrow B_i\}$ via $\mathcal{F}$. Since $\mathcal{F}$ is fair and ξ is non-failed, there exists a SLD-derivation of $P \cup \{\leftarrow B_1, \ldots, B_n\}$ where all the atoms in ξ' are eventually selected, and the other selections are made accordingly to ξ. Thus, we obtain a SLD-derivation of $P \cup \{\leftarrow B_1, \ldots, B_n\}$ via $\mathcal{F}$ whose length is greater or equal than the length of ξ'. In conclusion, (6) holds. Observing that $\leftarrow B_1, \ldots, B_n$ is an instance of a SLD-resolvent G of $P \cup \{\leftarrow A\}$, we now calculate for $i \in [1, n]$:

$$\begin{array}{rl}
|A| = & length^P_{\mathcal{F}}(\leftarrow A) \\
\triangleright & \{ \text{ Lemma 18 } (i) \ \} \\
& length^P_{\mathcal{F}}(G) \\
\trianglerighteq & \{ \text{ Lemma 18 } (ii) \ \} \\
& length^P_{\mathcal{F}}(\leftarrow B_1, \ldots, B_n) \\
\trianglerighteq & \{ \ (6) \ \} \\
& length^P_{\mathcal{F}}(\leftarrow B_i) = |B_i|
\end{array}$$

(b) Suppose now that $I \not\models B_1, \ldots, B_n$. By definition of I, we have that for some $i \in [1, n]$ there exists $k_i \geq 0$ such that $T_P \downarrow k_i \not\models B_i$. By Lemma 4, we conclude that every SLD-derivation of $P \cup \{\leftarrow B_i\}$ via $\mathcal{F}$ is failed.

Consider now B_h such that $T_P \downarrow k_h \not\models B_h$ and $length^P_{\mathcal{F}}(\leftarrow B_h)$ is minimum. By Lemma 4, we have that for $i \neq h$ there exist a SLD-derivation ξ_i of

$P \cup \{ \leftarrow B_i\}$ via $\mathcal{F}$ such that ξ_i is either successful or has length greater or equal than $length^P_{\mathcal{F}}(\leftarrow B_h)$. We claim that:

$$length^P_{\mathcal{F}}(\leftarrow B_1 \,, \ldots \,, B_n \,) \trianglerighteq length^P_{\mathcal{F}}(\leftarrow B_h). \tag{7}$$

In fact, let ξ be a SLD-derivation of $P \cup \{ \leftarrow B_h\}$ with length $length^P_{\mathcal{F}}(\leftarrow B_h)$. Consider now a SLD-derivation ξ' of $P \cup \{ \leftarrow B_1 \,, \ldots \,, B_n \,\}$ via $\mathcal{F}$ where the atoms are selected accordingly to ξ and to ξ_i for $i \neq h$. We observe that the length of ξ' is at least $length^P_{\mathcal{F}}(\leftarrow B_h)$, since ξ_i for $i \neq h$ is either successful or longer than ξ. In conclusion, (7) holds. Observing that $\leftarrow B_1, \ldots, B_n$ is an instance of a SLD-resolvent G of $P \cup \{ \leftarrow A\}$, we calculate:

$$\begin{array}{rcl} |A| & = & length^P_{\mathcal{F}}(\leftarrow A) \\ & \triangleright & \quad \{ \text{ Lemma 18 } \textit{(i)} \} \\ & & length^P_{\mathcal{F}}(G) \\ & \trianglerighteq & \quad \{ \text{ Lemma 18 } \textit{(ii)} \} \\ & & length^P_{\mathcal{F}}(\leftarrow B_1, \ldots, B_n) \\ & \trianglerighteq & \quad \{ \text{ (7) } \} \\ & & length^P_{\mathcal{F}}(\leftarrow B_h) = |B_h| \end{array}$$

and then we conclude that $I \not\models B_h$ and $|A| \triangleright |B_h|$. □

Finally, we state completeness of the proof method.

Theorem 20 (Termination Completeness). *Assume that a program P and a goal G $\exists$-universally terminate. Then there exist $|\ |$ and I such that P and G are both fair-bounded by $|\ |$ and I.* □

Proof. Assume that $G = \ \leftarrow Q$. By Theorem 3, every SLD-derivation of $P \cup \{G\}$ via any fair selection rule $\mathcal{F}$ is finite. Consider the program $P' = P \cup \{ \ \mathtt{new} \leftarrow Q \ \}$, where `new` is a fresh predicate symbol. By Lemma 19 *(i)*, P' is fair-bounded by some $|\ |$ and I. Since `new` is a fresh symbol, the assumption of the Theorem implies that every SLD-derivation of $P' \cup \{ \leftarrow \mathtt{new}\}$ via $\mathcal{F}$ is finite. By Lemma 19 *(ii)*, we conclude that $|\mathtt{new}| \in N$. Consider now the restrictions of $|\ |$ and I to B_P, i.e. not including `new`. It is immediate to see that P is fair-bounded by the restrictions of $|\ |$ and I.
Turning the attention on G, since $\mathtt{new} \leftarrow Q$ is fair-bounded by $|\ |$ and I, we have that for every ground instance $\leftarrow A_1 \,, \ldots \,, A_n$ of G:

(a) if $I \models A_1 \,, \ldots \,, A_n$ then for $i \in [1,n]$, $|\mathtt{new}| \triangleright |A_i|$, and
(b) if $I \not\models A_1 \,, \ldots \,, A_n$ then there exists $i \in [1,n]$ such that $I \not\models A_i$ and $|\mathtt{new}| \triangleright |A_i|$.

In conclusion G is fair-bounded by the restrictions of $|\ |$ and I, by fixing $k = |\mathtt{new}|$ in Definition 8. □

Summarizing, in the last two sections we showed that the class of fair-bounded logic programs and goals precisely characterize the notion of $\exists$-universal termination, i.e. the class of logic programs and goals for which a complete control exists in the sense of Definition 1.

6 Arithmetic built-in's

A trend in logic programming language design is to forbid the use of extra-logical predicates present in early systems, such as Prolog, or, alternatively, to provide a declarative interpretation for them. In this section, we concentrate on a largely employed class of extra-logical predicates, namely arithmetic built-in's. We show that fair-boundedness naturally extends to programs containing those predicates. Accordingly to Sterling and Shapiro [21], we declaratively model the built-in `>` by implicitly augmenting the program with the (infinite) set of unit clauses

$$n \texttt{ > } m. \tag{8}$$

for every n, m ground arithmetic expressions (gae's for short) such that $val(n) > val(m)$, where $val(n)$ is the integer denoted by n, and analogously for m. We model `<, <=, =:=, =/=, >=` analogously. For the built-in `is`, instead, we consider the clauses n `is` m. where m is a gae and n is the arithmetic constant, i.e. `0, 1, -1, ...` etc., such that $val(n) = val(m)$. Since the resulting SLD-trees are not finitely branching, a further assumption is necessary.

Definition 21. We say that a SLD-derivation *ends with a run-time error* if an atom n `>` m is selected with n or m not gae's. □

Analogous definitions should be considered for `<, <=, =:=, =/=, >=`. In the case of the atom m `is` n, it is sufficient to assume that m is a gae. Prolog systems stop the computation when a SLD-derivation (via the leftmost selection rule) ends with a run-time error. In recent systems, such as Gödel and Mercury, the selection of insufficiently instantiated atoms is delayed until possible. In particular, Mercury statically reorders clause bodies in such a way that its type system can ensure that no insufficiently instantiated atom will be selected via the leftmost selection rule. Gödel, instead, checks that condition at run-time.

Let us consider now fair-boundedness in presence of the arithmetic built-in `>`. We observe that clauses from (8) are clearly fair-bounded by any $|\ |$ and I, provided that I is a model of (8). Therefore, the proof obligations of Definition 7 have not to be explicitly shown for arithmetic built-in's. Next Theorem extends the Correctness and Completeness Theorems 14 and 20. We recall that a program P implicitly contains clauses such as (8).

Theorem 22. *Let P be a program containing arithmetic built-in's and G a goal both fair-bounded by $|\ |$ and I. Then P and G ∃-universally terminate (possibly ending with a run-time error).*

Conversely, assume that there exists a selection rule $\mathcal{S}$ such that every SLD-derivation of $P \cup \{G\}$ via $\mathcal{S}$ is finite and not ending with a run-time error. Then there exist $|\ |$ and I such that P and G are both fair-bounded by $|\ |$ and I.

Proof. The same of Theorems 13 and 19 by noting that SLD-trees are still finitely branching. □

Example 3. Let us consider the `Partition` program.

```
part(X, [Y | Ys], [Y | Ls], Bs) ←
   X > Y, part(X, Ys, Ls, Bs).
part(X, [Y | Ys], Ls, [Y | Bs]) ←
   X ≤ Y, part(X, Ys, Ls, Bs).
part(X, [], [], []).
```

`Partition` is fair-bounded by $|\ |$ and any model I of it, where:

$$|\mathtt{part}(x,xs,ls,bs)| = llen(xs),$$
$$|s > t| = 0 \text{ and } |s \leq t| = 0.$$

The proof obligations for the unit clause are trivial. Let us consider now a ground instance of the first clause (for the second clause we reason simmetrically):

$$\mathtt{part}(x,\ [y\ |\ ys],\ [y\ |\ ls],\ bs) \leftarrow x > y,\ \mathtt{part}(x,\ ys,\ ls,\ bs).$$

We have that

$$\begin{aligned} &|\mathtt{part}(x,\ [y\ |\ ys],\ [y\ |\ ls],\ bs)| \\ &= llen(ys) + 1 \\ &> llen(ys) \\ &= |\mathtt{part}(x,\ ys,\ ls,\ bs)| \end{aligned}$$

and $|\mathtt{part}(x,\ [y\ |\ ys],\ [y\ |\ ls],\ bs)| > 0 = |x > y|$. Suppose now that the body of the clause instance is true in I. Then the two inequalities above show condition *(a)* of Definition 7. On the contrary, suppose that the body is not true in I. Then one of the two body atoms in not in I. Since the two inequalities above show the decreasing of level mapping for both body atoms, we have that there exists a body atom not in I for which the level mapping decreases, i.e. condition *(a)* of Definition 7 holds.

Summarizing, we have that `Partition` is fair-bounded by $|\ |$ and I. Moreover the goal $\leftarrow \mathtt{part}(x,\ ys,\ \mathtt{Ls},\ \mathtt{Bs})$ is fair-bounded by the same $|\ |$ and I, when ys is a list. Finally, by Theorem 22 we conclude that every SLD-derivation of $\mathtt{Partition} \cup \{\leftarrow \mathtt{part}(xs,\ ys,\ \mathtt{Ls},\ \mathtt{Bs})\}$ via a fair selection rule is finite (possibly ending with a run-time error). □

7 Classes of terminating programs

As a by-result of the Termination Completeness Theorem 20, any class of programs and goals that universally terminate via some selection rule is included in the class of fair-bounded programs and goals. In this section, we relate fair-bounded logic program to the well-known classes of recurrent, acceptable and bounded programs.

Recurrent programs

Recurrent programs were introduced by Bezem [4]. They coincide with the class of programs such that every SLD-derivation for every ground goal via *every* selection rule is finite.

Definition 23. Let P be a logic program and $|\ | : B_P \to N$ a level mapping for P. P is called *recurrent by* $|\ |$ iff for every $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$, for $i \in [1, n]$ $\quad |A| > |B_i|$. □

The following result states that proving that a program is recurrent is a sufficient condition for proving the program fair-bounded. For instance, the proof that `Partition` program is fair-bounded (see Example 3) is in practice a proof that `Partition` is recurrent.

Theorem 24. *If a logic program P is recurrent by $|\ |$ then P is fair-bounded by $|\ |$ and any I.*

Proof. Immediate from Definitions 7 and 23. □

Acceptable programs

Acceptable programs were introduced by Apt and Pedreschi [2]. They coincide with the class of programs such that every SLD-derivation for every ground goal via the leftmost selection rule is finite.

Definition 25. Let P be a logic program, I a Herbrand interpretation and $|\ | : B_P \to N$ a level mapping for P. P is called *acceptable by* $|\ |$ *and* I iff I is a model of P and for every $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$, for $i \in [1, n]$:

$$I \models B_1, \ldots, B_{i-1} \quad \textit{implies} \quad |A| > |B_i|.$$ □

A large number of Prolog programs have been shown to be acceptable [2]. As in the case of recurrent programs, acceptability is a sufficient condition for proving fair-boundedness.

Theorem 26. *Let P a logic program acceptable by $|\ |$ and I. Then P is fair-bounded by $|\ |$ and any I.*

Proof. Consider $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$:

(a) if $I \models B_1, \ldots, B_n$ then by Definition 25 we conclude that for every $i \in [1, n]$ $|A| > |B_i|$.

(b) if $I \not\models B_1, \ldots, B_n$ then let $k \in [1, n]$ such that $I \models B_1, \ldots, B_{k-1}$ and $I \not\models B_k$. By Definition 25, we conclude $|A| > |B_k|$. Summarizing, $I \not\models B_k$ and $|A| > |B_k|$. □

Bounded programs.

Bounded programs were introduced by Ruggieri [17]. They coincide with the class of programs such that there are finitely many SLD-refutations for every ground goal via any selection rule.

Definition 27. Let P be a logic program, I a Herbrand interpretation and $|\ |: B_P \rightarrow N$ a level mapping for P. P is called *bounded by* $|\ |$ *and* I iff I is a model of P such that for every $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$:

$$I \models B_1, \ldots, B_n \quad implies \quad \text{for every } i \in [1, n] \quad |A| > |B_i|. \qquad \square$$

Bounded programs are defined by discarding the proof obligation *(b)* in Definition 7. While recurrent and acceptable programs are included in the class of fair-bounded programs, we have that fair-bounded programs are a special case of bounded programs, apart from the fact that level mappings $|\ |: B_P \rightarrow N$ into naturals are considered in [17].

8 Inferring fair-boundedness

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

Theorem 28. *It is undecidable whether there exist* $|\ |$ *and* I *such that a program* P *and a goal* G *are both fair-bounded by* $|\ |$ *and* I.

Proof. By Correctness and Completeness Theorems 14 and 20, the problem is decidable iff it is decidable whether P and G $\exists$-universally terminate. In [6, Theorem 8], it is shown that it is undecidable whether given a program consisting of one clause of the form

$$p(t_1, \ldots, t_n) \leftarrow p(s_1, \ldots, s_n) \tag{9}$$

and a goal $\leftarrow p(v_1, \ldots, v_n)$, the resolution stops. The form (9) implies that there is only one SLD-derivation for the program and the goal. As a consequence, it is undecidable whether they $\exists$-universally terminate, and *a fortiori* whether they are fair-bounded. $\square$

On a practical level, however, the undecidability result is not a major problem.

In general, by the Termination Completeness Theorem 20, fair-bounded programs and goals include every class of programs and goals that universally terminate via some selection rule. Therefore, any existing automatic tool for proving universal termination of P and G is sufficient for proving that P and G are fair-bounded. This fact allows us to reuse all existing automatic tools and termination proofs to the purpose of showing fair-boundedness.

Apart from this consideration, we argue that existing tools can be adapted for proving the proof obligations of Definition 7 directly. In fact, those proof obligations are variations of proof obligations of well-known methods, such as recurrency and acceptability, for which automatic tools already exists.

9 Conclusions

Related works

Termination of logic programs has been studied for a long time. We refer the reader to [18] for a survey on termination proof methods for logic programs, including definitions, relations, properties and tools for automatic checking and inference. Until recently, however, termination analysis has been restricted to fixed selection rules, and in particular Prolog's leftmost selection rule. In the following, we recall some recent proposals dealing with selection rules other than leftmost's.

Speirs et al. [20] present the static termination analysis algorithm of Mercury, which ensures termination with respect to the leftmost selection rule for a program obtained by permuting the body atoms of the original program. Martin and King [13] present a transformational approach for termination of Gödel programs which relies on an run-time analysis of the depth of refutations. The basic idea is not to proceed searching along a derivation longer than the longest refutation. Marchiori and Teusink [12] propose a sufficient termination method for so-called *local* selection rules, i.e. rules that resolve completely an atom in a goal before starting resolution of the other atoms. Krishna Rao et al. [11] propose a method for proving termination of GHC programs by transforming a GHC program into a term rewriting systems, and then applying well-known techniques for that paradigm. GHC is a concurrent logic language where an atom and a clause can be resolved only if the atom is an instance of the clause's head. Also, Plümer [16] studied termination of GHC programs by relating GHC derivations to the leftmost selection rule, and the using the techniques proposed in [15].

Apart from the proposal of Martin and King, all the cited approaches impose hypothesis that prevent coroutining executions, such as in the producer-consumer program `ProdCons` of Example 1. On the contrary, we observe that fair selection rules allow for coroutining and parallel executions. We refer the reader to Naish [14] for a discussion about subtles and problems of termination in presence of coroutining executions. He proposes how terminating programs can be composed by disjunctions, conjunction and recursions to form terminating programs in presence of couroutining.

We conclude this brief overview by mentioning a few fundamental studies on properties of fair selection rules in SLD-resolution due to Apt and van Emden [3], Lassez and Maher [9] and van Emden and Nait Abdallah [23].

Conclusion

In this paper, we introduced the notion of ∃-universal termination of logic programs. We claimed that it is an essential concept for declarative programming, where a crucial point is to associate a complete control strategy to programs. We have shown that ∃-universal termination and universal termination via fair selection rules coincide.

Then we introduced a characterization of $\exists$-universal termination by means of the notion of *fair-bounded programs* that provide us with a correct and complete method of proving termination. The method is purely declarative, modular, simple and practical in *paper & pencil* proofs. Moreover, by the Termination Completeness Theorem 20 any existing (possibly automatic) method for proving universal termination is a sufficient method for proving fair-boundedness.

Acknowledgments. I am grateful to F. Mesnard and D. Pedreschi for having stimulated the development of this paper. Also, thanks are owing to S. Contiero and M. Gabbrielli for comments on a draft.

References

1. K.R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 493–574. Elsevier, 1990.
2. K.R. Apt and D. Pedreschi. Reasoning about termination of pure prolog programs. *Information and computation*, 106(1):109–157, 1993.
3. K.R. Apt and M.H. van Emden. Contributions to the Theory of Logic Programming. *Journal of the ACM*, 29(3):841–862, 1982.
4. M.A. Bezem. Strong Termination of Logic Programs. *Journal of Logic Programming*, 15(1 & 2):79–98, 1993.
5. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 8:69–116, 1987.
6. 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.
7. P. M. Hill and J. W. Lloyd. *The Gödel Programming Language*. The MIT Press, 1994.
8. R. A. Kowalski. Algorithms = Logic + Control. *Comm. of the ACM*, 22(7):424–436, 1979.
9. J.L. Lassez and M.J. Maher. Closure and fairness in the semantics of Logic Programming. *Theoretical Computer Science*, 29:167–184, 1984.
10. J.W. Lloyd. *Foundations of logic programming.* Springer-Verlag, Berlin, Berlin, second edition, 1987.
11. R. K. Shyamasundar M. R. K. Krishna Rao, D. Kapur. Proving termination of GHC programs. *New Generation Computing*, 15(3):293–338, 1997.
12. E. Marchiori and F. Teusink. On Termination of Logic Programs with Delay Declarations. In *Proc. of the 1995 International Symposium on Logic Programming*, pages 447–461. The MIT Press, 1995.
13. J. Martin and A. King. Generating Efficient, Terminating Logic Programs. In *Proceedings of TAPSOFT'97*, number 1214 in Lecture Notes in Computer Science, pages 273–284, 1997.
14. L. Naish. Coroutining and the construction of terminating logic programs. Technical Report 92/5, Department of Computer Science, University of Melbourne, 1992.
15. L. Plümer. *Termination Proofs for Logic Programs.* Lecture Notes in Artificial Intelligence 446, Springer-Verlag, Berlin, 1990.

16. L. Plümer. Automatic verification of parallel logic programs: Termination. In C. Beierle and L. Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, chapter 3, pages 91–119. Elsevier, 1995.
17. S. Ruggieri. Decidability of Logic Program Semantics and Applications to Testing. In *Proc. of PLILP'96*, volume 1140 of *Lecture Notes in Computer Science*, pages 347–362. Springer-Verlag, Berlin, 1996.
18. D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *Journal of Logic Programming*, 19-20:199–260, 1994.
19. Z. Somogyi, F. Henderson, and T. Conway. The execution algorithm of Mercury, an efficient purely declarative logic programming language. *Journal of Logic Programming*, 29(1–3):17–64, 1996.
20. C. Speirs, Z. Somogyi, and H. Søndergaard. Termination Analysis for Mercury. In P. van Hentenryck, editor, *Proc. of SAS'97*, volume 1302 of *Lecture Notes in Computer Science*, pages 160–171. Springer-Verlag, Berlin, 1997.
21. L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, second edition, 1994.
22. J. Thom and J. Zobel. NU-Prolog reference manual, version 1.3. Technical report, Department of Computer Science, University of Melbourne, Australia, 1988.
23. M. H. van Emden and M. A. Nait Abdallah. Top-down semantics of fair computations of logic programs. *Journal of Logic Programming*, 2(1):67–76, 1985.