On Greatest Fixpoint Semantics of Logic Programming

MATHIEU JAUME, *KIP6—SPI, Université Paris 6, 8 rue du Capitaine Scott, 75015 Paris, France.*
E-mail: Mathieu.Jaume@lip6.fr

Abstract

The study of fixpoints has long been at the heart of logic programming. However, whereas least fixpoint semantics works well for SLD-refutations (i.e. is sound and complete), there is no satisfactory (i.e. complete) fixpoint semantics for infinite derivations. In this paper, we focus on this problem. Standard approaches in this area consist in concentrating on infinite derivations that can be seen as computing, in the limit, some infinite object. This is usually done by extending the domain of computation with infinite elements and then defining the meaning of programs in terms of greatest fixpoints. The main drawback of these approaches is that the semantics defined is not complete. Hence, since defining a greatest fixpoint semantics for logic programs amounts to consider a program as a set of co-inductive definitions, we focus on this identification at a deeper level by considering infinite derivations as proof-terms in a co-inductive set. This reading leads into considering derivations as proofs rather than computations and allows one to show that for the subclass of infinite derivations over the domain of finite terms, a complete greatest fixpoint semantics can be obtained. Our main result is that the greatest fixpoint of the one-step-inference operator for the $\mathcal{C}$-semantics corresponds to atoms that have a non-failing fair derivation with the additional property that complete information over a variable is obtained after finitely many steps.

Keywords: Logic programming, infinite SLD-derivations, co-inductive definitions, fixpoint semantics.

1 Introduction — motivations

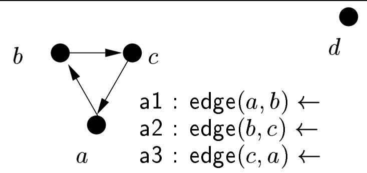
In computer science, termination of programs is a traditional requirement. Logic programming does not escape from this influence and there exist many works about termination of logic programs (see [8]). Hence, standard semantics of logic programs [4] is only concerned with refutations and then is strongly related to termination (infinite derivations are not taken into account). However, infinite behaviour of programs can be useful to model some situations, as some infinite processes or the computations of infinite objects, and this topic has received an increasing interest in the context of many programming paradigms: λ-calculus [25], rewrite systems [10], logic programming [1, 2, 13, 17, 26, 27, 31], constraint logic programming [19, 22], concurrent constraint programming [7, 30, 33, 34, 35] ... . In fact, the theory of programming languages often calls for infinite objects.

In the field of logic programming, infinite derivations are usually considered as useful to model the process of computation of an infinite object. As a typical example, with the program $P = \{\mathsf{LN}(x, [x|l]) \leftarrow \mathsf{LN}(S(x), l)\}$ we can obtain, from the query $\mathsf{LN}(k, l)$, the following infinite derivation:

$$\begin{array}{c}\mathsf{LN}(k,l) \rightarrow_P \mathsf{LN}(S(k), l_1) \rightarrow_P \mathsf{LN}(S^2(k), l_2) \rightarrow_P \cdots \\ \cdots \rightarrow_P \mathsf{LN}(S^{i-1}(k), l_{i-1}) \rightarrow_P \mathsf{LN}(S^i(k), l_i) \rightarrow_P \cdots\end{array}$$

At every step i of this derivation, a better approximation

$$\theta_i \cdots \theta_1 l = [k, S(k), ..., S^{i-1}(k)|l_i]$$



a1 : $\mathsf{edge}(a, b) \leftarrow$
a2 : $\mathsf{edge}(b, c) \leftarrow$
a3 : $\mathsf{edge}(c, a) \leftarrow$

c1 : $\mathsf{path}(x, x) \leftarrow$
c2 : $\mathsf{path}(x, z) \leftarrow \mathsf{edge}(x, y), \mathsf{path}(y, z)$

FIGURE 1. Connectivity in a directed graph

of the second argument is obtained (θ_i denotes the most general unifier used during the ith transition). The result 'computed at infinity' is the 'limit' of the sequence of approximations and corresponds to the infinite sequence of integers starting from k.

However, there exist logic programs from which infinite derivations that can be obtained do not compute any infinite object. This kind of program arises naturally when computers are used to model the execution of some processes. For example, the famous dining philosophers problem, introduced by Dijkstra as a model for resource sharing, can be described by a logic program from which every derivation is infinite, does not compute an infinite object, and just describes an infinite sequence of actions done by the philosophers. A more simple and typical example is the program testing connectivity in a directed graph. Since the directed graph considered here (see figure 1) is cyclic, there exists an infinite derivation from the query $\mathsf{path}(a, x)$ corresponding to the infinite path starting from a in the cycle:

$$\begin{aligned} \mathsf{path}(a, z) \quad & \overset{\mathsf{c2}}{\rightarrow}_P \mathsf{edge}(a, y_1), \mathsf{path}(y_1, z) \overset{\mathsf{a1}}{\rightarrow}_P \mathsf{path}(b, z) \\ & \overset{\mathsf{c2}}{\rightarrow}_P \mathsf{edge}(b, y_2), \mathsf{path}(y_2, z) \overset{\mathsf{a2}}{\rightarrow}_P \mathsf{path}(c, z) \\ & \overset{\mathsf{c2}}{\rightarrow}_P \mathsf{edge}(c, y_3), \mathsf{path}(y_3, z) \overset{\mathsf{a3}}{\rightarrow}_P \mathsf{path}(a, z) \rightarrow_P \cdots \end{aligned} \tag{1.1}$$

Clearly, this derivation does not compute any infinite object.

The main approaches to assign some meaning to infinite derivations in 'pure' logic programming occurring in the literature [1, 2, 13, 17, 26, 27, 31] concentrate on the aspects related to the semantics of infinite objects and to the models for logic programs which take them into account. In this setting, only infinite derivations 'doing useful computations, in some sense' are considered: these derivations must compute the approximations of at least an infinite object to be 'useful'. This corresponds to the informally intended meaning of infinite derivations. Hence, the relevant notion is the one of 'computation in the limit' of an infinite object (such as streams or lazy lists). The sense of a 'useful' infinite derivation is given by the notion of *atom computed at infinity* (i.e. an infinite atom A such that there exists a finite atom from which there exists an infinite derivation which 'computes at infinity' A). However, these approaches are not satisfactory since most of them are defined by a greatest fixpoint construction which is unfortunately not always effective (i.e. there exist programs whose greatest fixpoint contains infinite elements that cannot be computed at infinity by a derivation). Indeed, the greatest fixpoint semantics gives a non-empty denotation not only to non-terminating programs computing at infinity infinite terms, but also to programs like the program defined in Figure 1 or the program

$$P = \{p(x) \leftarrow p(x)\} \tag{1.2}$$

called 'bad programs' in [31]. In fact, such programs have a non-empty denotation even if they do not allow to compute at infinity an infinite object. For example, with program (1.2), for every (finite or infinite) term t, $p(t)$ is in the greatest fixpoint of the classical one-step-inference operator associated with P and in particular, $p(f^\omega)$ is not computable at infinity by an infinite derivation. The construction of the greatest fixpoint does not reflect how the infinite terms are constructed during a derivation. Hence, either such semantics [13, 17, 27] are not complete, since there exist infinite atoms in the denotation of a program which are not computable at infinity, or the completeness is expressed as follows [1, 22]:

$$A \in \mathsf{Denotation}(P) \Leftrightarrow \begin{array}{l} A \text{ is either the root of a finite successful derivation} \\ \text{or the root of a fair infinite derivation} \end{array}$$

where A is a possibly infinite atom. Nevertheless, in this case, this 'completeness' result is obtained by allowing infinite terms in queries which start SLD-derivations. For example, with program (1.2), such an approach allows to consider the derivation:

$$p(f^\omega) \rightarrow_P \cdots \rightarrow_P p(f^\omega) \rightarrow_P \cdots \tag{1.3}$$

This requirement is clearly stated in [34]. However, this does not correspond to the standard operational semantics of logic programs as defined in [4] (in an implicit way, this implies the use of an adequate unification algorithm), and furthermore, this does not capture the notion of atom computed at infinity.

Since, defining a greatest fixpoint semantics for infinite derivations amounts to view at a logic program as a set of co-inductive definitions, in this paper, we investigate this '*logic programs as co-inductive definitions*' paradigm at a deeper level. As we will see, the notion of atoms computed at infinity cannot be captured by a greatest fixpoint construction and our approach is the exact opposite. We will focus on infinite derivations which do not compute at infinity any infinite object and for which a complete greatest fixpoint semantics can be obtained. Indeed, it is now well known that standard semantics of logic programs can be expressed by purely proof-theoretic methods [6, 9, 14, 15, 16, 18, 32]. The most immediate way to give such a semantics is to consider clauses as inference rules, rather than logic formulas, and then a logic program as an inference system. From this point of view, the denotation of a program is the set of theorems which can be derived in this system. Within this framework, (co-)inductive definitions are a natural way to define the denotation of logic programs. This will lead us to define a sound and complete semantics for the subclass of *infinite derivations over the domain of finite terms* (i.e. infinite derivations which do not compute at infinity any infinite term). One may question about the interest of such derivations which are often put aside as meaningless. However, as we said, such derivations are useful to model some infinite processes. Furthermore, it seems that incompleteness of the usual approaches comes from these derivations and this work can provide some illumination on previous work and complements the knowledge we have in this area.

The rest of the paper is organized as follows. First, Section 2 introduces the basic definitions and notations. Then Section 3 presents a brief survey of the standard approaches dealing with infinite derivations. Finally, Section 4 presents a sound and complete semantics for the subclass of infinite derivations over the domain of finite terms.

2 Background and notation

In the following, we assume familiarity with the standard notions of (co-)inductive definitions and logic programming (Herbrand semantics and $\mathcal{C}$-semantics) as introduced in [3, 4, 11, 12].

(Co-)inductive sets can be defined by some rules for generating elements of the set and by adding that an object is to be in the set only if it has been generated by applying these rules. Given a rule set Φ (a *rule* is written $e \leftarrow E$, where E is the set of premisses, and e is the conclusion) a set A is said to be Φ*-closed* (resp. Φ*-dense*) if each rule in Φ whose premisses are in A also has its conclusion in A (resp. if for every $a \in A$ there is a set $E \subseteq A$ such that $(a \leftarrow E) \in \Phi$). The set inductively (resp. co-inductively) defined by Φ, written $\mathsf{Ind}(\Phi)$ (resp. $\mathsf{CoInd}(\Phi)$), contains elements which have been obtained by an ending (resp. a possibly non-ending) method of construction, and is defined by:

$$\mathsf{Ind}(\Phi) = \bigcap\{A \mid A \text{ is } \Phi\text{-closed}\} \quad (\text{resp. } \mathsf{CoInd}(\Phi) = \bigcup\{A \mid A \text{ is } \Phi\text{-dense}\})$$

These sets can also be expressed by using monotone operators: if Φ is a rule set, we may define a monotone operator $\mathcal{T}_\Phi : 2^{\mathcal{B}} \to 2^{\mathcal{B}}$, where $\mathcal{B} = \cup_{e \leftarrow E \in \Phi}\{\{e\} \cup E\}$, by:

$$\mathcal{T}_\Phi(A) = \{e \in \mathcal{B} \mid \exists\ e \leftarrow E \in \Phi, \quad E \subseteq A\} \tag{2.1}$$

and then $\mathsf{Ind}(\Phi)$ and $\mathsf{CoInd}(\Phi)$ can be characterized in terms of least and greatest fixpoints of $\mathcal{T}_\Phi$:

$$\mathsf{Ind}(\Phi) = \bigcap_{\mathcal{T}_\Phi(A) \subseteq A} A = \mathsf{lfp}(\mathcal{T}_\Phi) \qquad \mathsf{CoInd}(\Phi) = \bigcup_{A \subseteq \mathcal{T}_\Phi(A)} A = \mathsf{gfp}(\mathcal{T}_\Phi).$$

Σ, Π and X denote respectively a set of function symbols, a set of predicate symbols, and a set of variable symbols. Elements of $T_\Sigma[X]$ (resp. $T_\Sigma^\infty[X]$) are finite (resp. possibly infinite) *terms* over $X \cup \Sigma$. A *substitution* θ is a mapping from X to terms such that $\{x \mid x \neq \theta x\} = dom(\theta)$ is finite. $range(\theta)$ denotes the set $\{var(\theta x) \mid x \in dom(\theta)\}$. We will write:

$$\theta = \begin{bmatrix} x_1 & \cdots & x_k \\ t_1 & \cdots & t_k \end{bmatrix}$$

the substitution whose domain is $\{x_1, \cdots, x_k\}$ and such that $\theta x_i = t_i$. Hence, the identity substitution will be denoted by $[\,]$. Composition of substitutions induces a preorder on substitutions ($\theta_1 \leq \theta_2 \Leftrightarrow \exists \mu, \mu\theta_1 = \theta_2$) and on expressions ($E_1 \leq E_2 \Leftrightarrow \exists \mu, \mu E_1 = E_2$). A renaming substitution is a mapping $\sigma : X \to X$ such that $\forall x, y \in dom(\sigma), x \neq y \Rightarrow \sigma(x) \neq \sigma(y)$. A mgu is a minimal idempotent unifier. The preorder $\leq$ induces an equivalence relation $\approx$ (called *variance*): $E_1 \approx E_2$ iff there exist two renaming substitutions θ_1 and θ_2 such that $\theta_1 E_1 = E_2$ and $\theta_2 E_2 = E_1$. $At_{\Sigma,\Pi}[X]$ (resp. $At_{\Sigma,\Pi}^\infty[X]$) denotes the set of finite (resp. possibly infinite) *atoms* over $X \cup \Sigma \cup \Pi$. Given a clause $C \in P$, we write C^+ for its head and C^- for its body. An *SLD-derivation* with a program P is a possibly infinite sequence of transitions:

$$\underbrace{A_1, \cdots, A_k, \cdots, A_n}_{R} \overset{C,\theta}{\rightarrow}_P \underbrace{\theta(A_1, \cdots, A_{k-1}, B_1, \cdots, B_q, A_{k+1}, \cdots, A_n)}_{\theta R[k \leftarrow C^-]}$$

where θ is a mgu of C^+ and A_k and where C is a variant of a clause in P, whose body is $B_1, \cdots, B_q$. In an SLD-derivation from a goal R_0, the sequence of clauses $C_1, C_2, \cdots$ is such that[1]:

$$\forall i \geq 1 \quad var(C_i) \cap (\cup_{j<i} var(C_j) \cup var(R_0)) = \emptyset. \tag{2.2}$$

As defined in [27], an SLD-derivation is *fair* if it is either failed or, for every atom B in the derivation, (some further instantiated version of) B is selected within a finite number of steps. Given a set E of terms, atoms or clauses, we use the following notation:

$$\begin{array}{lll} \lceil E \rceil & = \{\theta e \mid e \in E \ , \ \theta : X \to T_\Sigma[X]\} & \text{(finite instances)} \\ [E] & = \{\theta e \mid e \in E \ , \ \theta : X \to T_\Sigma[\emptyset]\} & \text{(finite ground instances)} \\ [\![E]\!] & = \{\theta e \mid e \in E \ , \ \theta : X \to T_\Sigma^\infty[\emptyset]\} & \text{(ground possibly infinite instances).} \end{array}$$

Furthermore, for the sake of simplicity, we will use the same notation to denote E and the quotient set of E with respect to the variance equivalence relation. Hence, E also denotes the set containing all the variants of terms, atoms or clauses in E.

In the framework of $\mathcal{C}$*-semantics*, interpretations are subsets of $At_{\Sigma,\Pi}[X]$ and, for an atom, the notion of $\mathcal{C}$-truth coincides with the one of being a member of. In order to avoid the situation where an atom A is $\mathcal{C}$-true with respect to an interpretation I which does not contain instances of A, interpretations are required to be upward closed (i.e. $(A \in I \wedge A \leq B) \Rightarrow B \in I$): upward closed subsets of $At_{\Sigma,\Pi}[X]$ are called $\mathcal{C}$*-interpretations*. Given a $\mathcal{C}$-interpretation I, a clause $A \leftarrow B_1, \cdots, B_q$ is $\mathcal{C}$*-true* in I iff for every substitution θ, if atoms θB_i $(1 \leq i \leq q)$ are $\mathcal{C}$-true in I, then θA is $\mathcal{C}$-true in I. A $\mathcal{C}$-interpretation I is a $\mathcal{C}$*-model* of a program P if every clause in P is $\mathcal{C}$-true in I. Intersection of $\mathcal{C}$-models of a program P is a $\mathcal{C}$-model of P and every program P has a least $\mathcal{C}$-model, written $\mathcal{M}_P^{\mathcal{C}}$, which gives the declarative meaning of P. Operational semantics is defined by:

$$S_P^{\mathcal{C}} = \{A \in At_{\Sigma,\Pi}[X] \mid A \overset{*,\theta}{\to}_P \Box \text{ and } \theta A = A\}$$

and is related to $\mathcal{M}_P^{\mathcal{C}}$ by considering the least fixpoint of the upward continuous operator, over $\mathcal{C}$-interpretations, defined by:

$$\begin{array}{ll} T_P^{\mathcal{C}}(I) = \{A \in At_{\Sigma,\Pi}[X] \mid & \exists A' \leftarrow A_1, \cdots, A_n \in P \ \exists \theta \colon X \to T_\Sigma[X] \\ & \theta A' = A \text{ and } \theta A_i \in I \quad (1 \leq i \leq n)\}. \end{array} \tag{2.3}$$

In fact, a $\mathcal{C}$-interpretation I is a $\mathcal{C}$-model of a program P iff $T_P^{\mathcal{C}}(I) \subseteq I$ and the $\mathcal{C}$-semantics is sound and complete for SLD-refutations [12]:

$$\mathcal{M}_P^{\mathcal{C}} = \mathsf{lfp}(T_P^{\mathcal{C}}) = T_P^{\mathcal{C}\uparrow\omega} = S_P^{\mathcal{C}}. \tag{2.4}$$

3 Objects computed at infinity

In this section, we review the main approaches dealing with infinite derivations in 'pure' logic programming. Since, all of them are based on the concept of 'infinite atoms being computable at infinity', the universe of the discourse considered in these approaches contains infinite elements. Furthermore, as we said, semantics for non-terminating derivations are generally defined in terms of greatest fixpoint. In this case, a supplementary motivation for requiring

[1]As illustrated in [24], this renaming process is crucial and has been explicitely considered in our proofs. All the derivations considered here satisfy this requirement.

infinite elements in the domain of computation comes from the fact that they all try to get the equality $\mathsf{gfp}(T_P) = T_P^{\downarrow\omega}$, where T_P denotes the classical one-step-inference operator as defined in [4], which generally does not hold in the Herbrand base containing only finite atoms. Note that programs satisfying this property, called canonical programs, have been studied in [21] and in [29], model-theoretic techniques have been considered in order to define a domain over which such a property holds. Therefore, the first step of these approaches consists in defining a completion of the Herbrand base. The most used continuous structures are complete metric spaces and complete partial orders which are both characterized by the presence of infinite elements viewed as the limits of infinite sequences of finite objects.

Metric approach
The more immediate approach to the completion of the Herbrand base, due to M.A. Nait Abdallah [1] and used by J.W. Lloyd [27], is the metric one. The Herbrand base is made a complete metric space by introducing a distance between two terms t_1 and t_2 as follows: $d(t_1, t_2) = 0$ if $t_1 = t_2$ and $d(t_1, t_2) = 2^{-\inf\{n,\, \tau_n(t_1) \neq \tau_n(t_2)\}}$, where $\tau_n(t)$ denotes the truncation at height n of the tree t, otherwise. Now by adding to $T_\Sigma[X]$ all the limits of Cauchy sequences of terms, we obtain the set $T_\Sigma^\infty[X]$ of finite and infinite terms and $(T_\Sigma^\infty[X], d)$ is a complete metric space (for more details, see [5]). The distance d can be extended to ground atoms and the new Herbrand base considered is the metric completion of $At_{\Sigma,\Pi}[\emptyset]$, written $At_{\Sigma,\Pi}^\infty[\emptyset]$. Now, the (extended) operator T_P (over completed Herbrand interpretations) is both upward and downward continuous and the main result from [1] is expressed as follows:

$$A \in At_{\Sigma,\Pi}^\infty[\emptyset] \text{ begins an SLD-refutation or a fair infinite derivation iff } A \in T_P^{\downarrow\omega}.$$

However, as we said in the introduction, such an assertion does not correspond to a completeness result since, in (standard) logic programming, queries cannot contain infinite terms. In [27], J.W. Lloyd defines the set C_P of atoms computable at infinity from P as atoms A such there exists a finite atom B and an infinite fair derivation from B with mgu's $\theta_1, \theta_2, \cdots$ such that $\lim_{n\to\infty} d(A, \theta_n \cdots \theta_1 B) = 0$. The soundness theorem obtained in [27] states $C_P \subseteq \mathsf{gfp}(T_P)$. However, the metric approach does not lead to a complete semantics: there exist atoms in $\mathsf{gfp}(T_P)$ which are not computable by an infinite derivation. As we said, if we consider program (1.2), we have $p(f^\omega) \in \mathsf{gfp}(T_P)$ but $p(f^\omega) \notin C_P$. Another problem to the metric approach comes from the fact that the notion of (metric) limit does not allow one to always distinguish between two sequences converging to the same limit but which are distinct at an operational level (for more details, see [33]). This problem can be solved by considering complete partial ordered sets (see next paragraph).

Partial ordering approach
Another approach, due to W.G. Golson [13], is an order-theoretic one and is based on the completion by ideals of atoms. Given a partial ordered set E, an ideal $\mathcal{I}$ is a directed (every pair of elements has a least upper bound in $\mathcal{I}$) and downward closed (if $x \in E$, $y \in \mathcal{I}$ and $x \leq y$, then $x \in \mathcal{I}$) subset of E. Since the set of all the ideals ordered by set inclusion is a complete partial order, every chain of ideals has a least upper bound which is again an ideal, representing its limits. In [13], ideals of $At_{\Sigma,\Pi}[X]$, called objects, are defined as the sets $\mathcal{A}\Theta = \{\mathcal{A}' \mid \exists\sigma \in \Theta,\ \mathcal{A}' \leq \sigma\mathcal{A}\}$ where $\mathcal{A}$ is a set of finite atoms and Θ is a directed set of substitutions. Such an object is infinite if the cardinality of Θ (modulo renaming) is infinite. Interpretations are upward closed sets of ideals with respect to set inclusion (i.e. if $\alpha_1 \in \mathcal{I}$ and $\alpha_1 \subseteq \alpha_2$ then $\alpha_2 \in \mathcal{I}$) and given an interpretation I, $\min(I)$ denotes the set $\{\alpha \in I \mid \forall\beta \in I,\ \beta \subseteq \alpha \Rightarrow \alpha = \beta\}$. The considered one-step-inference operator T_P is

defined by:

$$\begin{array}{lll} T_P(I) = \{ & \alpha \text{ (object)} \mid & \exists\{A_i \leftarrow \mathcal{A}'_i\} \subseteq P \;\; \exists\Theta \quad \text{(directed set of substitutions)} \\ & & \alpha = \mathcal{A}\Theta \quad (\mathcal{A} = \cup\{A_i\}) \quad \text{and} \quad \mathcal{A}'\Theta \in I \quad (\mathcal{A}' = \cup\mathcal{A}'_i)\} \end{array}$$

and is shown downward continuous by considering only programs whose clauses C are such that $var(C^-) \subseteq var(C^+)$. The main result expressed in [13] is stated as follows:

> $\alpha \in \min(\mathsf{gfp}(T_P))$ iff there exists a fair derivation from $\mathcal{A}$ such that $\mathcal{A}\{\cup_j\{\sigma_j\}\} = \alpha$ where $\mathcal{A}$ is a collection of distinct rule head variants of P and where σ_j is the mgu used during the jth resolution step.

For example, with program (1.2), we have $\min(\mathsf{gfp}(T_P)) = \{p(z)\}\{[\,]\}$ which is not an infinite object while with the program:

$$P = \{p(f(x)) \leftarrow p(x)\} \tag{3.1}$$

we have $\min(\mathsf{gfp}(T_P)) = \{p(z)\}\left\{\left[\begin{array}{c} z \\ f^i(x_i) \end{array}\right]\right\}_{i\geq 0}$ which is an infinite object computed by the following fair infinite derivation:

$$p(z) \overset{\left[\begin{array}{c} z \\ f(x_1) \end{array}\right]}{\longrightarrow_P} p(x_1) \overset{\left[\begin{array}{c} x_1 \\ f(x_2) \end{array}\right]}{\longrightarrow_P} \cdots \overset{\left[\begin{array}{c} x_{i-1} \\ f(x_i) \end{array}\right]}{\longrightarrow_P} p(x_i) \overset{\left[\begin{array}{c} x_i \\ f(x_{i+1}) \end{array}\right]}{\longrightarrow_P} \cdots \tag{3.2}$$

However, infinite SLD-derivations are not completely characterized: T_P is shown downward continuous by considering a subclass of logic programs and furthermore, only a subclass of the finite and infinite elements constructible by non-terminating derivations of a logic program (called 'minimal' objects) are characterized by a proper subset of $\mathsf{gfp}(T_P)$. For example, with the program $P = \{p(f(x), y) \leftarrow p(x, y)\}$, the infinite derivation starting from $p(x, y)$ computes a minimal object while from $p(x, g(y))$, no minimal object is computed.

Others approaches
For the sake of 'completeness' we briefly cite here some others approaches dealing with infinite derivations in 'pure' logic programming. Since greatest fixpoint semantics is sound but not complete, in [17] J. Hein has proposed to characterize those logic programs for which the greatest fixpoint semantics is complete and on the other hand, in order to solve the completeness problem for a program P, he has proposed a completion method to obtain a program Q (containing P) such that every atom in $\mathsf{gfp}(T_P)$ corresponding to an infinite ground instance of the head of a unit clause in P can be computed at infinity by Q. However, also in this work, full correspondence is not achieved. A different approach, based on adding to the program some suitable clauses containing indefinite terms and then applying a least fixpoint construction, has been developed by G. Levi and C. Palamidessi in [31] and revisited in [26]. In an order-theoretic framework (involving algebraic complete partial order), they consider the 'final result' of an infinite derivation as the limit of a sequence of approximations, characterized by a least fixpoint semantics based on a modified version of the programs (some suitable unit clauses are added and used as the starting point of the construction of a sequence of non-empty interpretations). Then, infinite objects in the denotation of a program are characterized by the topological closure of $\mathsf{lfp}(T_{P\cup C(P)})$ (where $C(P)$ is the set of added clauses): each infinite element is the least upper bound of a directed set (of finite elements which are its partial approximations) included in $\mathsf{lfp}(T_{P\cup C(P)})$. However, also in

this case, the construction proposed has not been able to reach a full correspondence with the operational semantics.

Of course, we focus here on ‘pure’ logic programming and the notion of ‘atom computed at infinity’ was an earlier approach to give a semantics to infinite processes expressed as logic programs which has been abandoned. Indeed, there exist now many works on infinite computations within more general frameworks [7, 19, 22, 30, 33, 34, 35] (constraint logic programming or concurrent constraint programming). In particular, in the area of constraint logic programming, some results are closely related to our main results. Nevertheless, in this paper, our aim is to clarify the relationship between (standard) operational semantics and greatest fixpoint semantics of (pure) logic programs from a proof-theoretic point of view.

4 Infinite derivations over the domain of finite terms

4.1 Proof-theoretic reading of logic programming

The technical developments of the next subsections are based on a proof-theoretic reading of logic programming. Hence, in this subsection, we first present a brief discussion about the ‘logic programs as (co-)inductive definitions’ paradigm. Indeed, an advantage of logic programming is the justification of results in terms of logical implications, often considered as a much more intuitive notion than fixpoints. But as a conceptually equivalent notion, fixpoints play a crucial role in the semantics of logic programming. Indeed, it is a common pratice to think of a logic program as defining a continuous mapping on the lattice of Herbrand interpretations and, by using another terminology, this means that programs are inference systems. In others words, as claimed in [9, 18, 32], logic programs are (co-)inductive definitions. In this way, a clause $A \leftarrow A_1, \cdots, A_n$ can be viewed as a rule used to prove A from the proofs of $A_1, \cdots, A_n$. Hence, this leads to consider a program P as a schematic rule which abbreviates an infinite set of rules. By following this approach, clauses should not be viewed as assertions in first-order logic as in the ‘orthodox’ reading of logic programming, but as rules generating a set. The fixpoint semantics has long been used as a technical device, however, it corresponds to the ‘logic programs as (co-)inductive definitions’ paradigm which can be considered as the logic program’s intrinsic declarative content.

On least fixpoint semantics of logic programming
Let us see how standard results of $\mathcal{C}$-semantics can be expressed in terms of inductive definitions. Indeed, many properties of logic programs are similar to these enjoyed by inductive definitions. In fact, operational semantics and least fixpoint semantics are just two equivalent ways to prove an atom: such proofs are (linear) SLD-refutations with operational semantics while they correspond to proof trees (see Definition 4.2) with fixpoint semantics (in the finite case, the correspondence is obvious). Let us consider now the model-theoretic semantics. Recall that, the model intersection property allows to define the least $\mathcal{C}$-model $\mathcal{M}_P^{\mathcal{C}}$ of a program P as the intersection of all $\mathcal{C}$-models of P, which are $\mathcal{C}$-interpretations I such that $T_P^{\mathcal{C}}(I) \subseteq I$ where $T_P^{\mathcal{C}}$ is the operator defined by (2.3). Now, since $T_P^{\mathcal{C}}$ is exactly the operator $\mathcal{T}_{\lceil P \rceil}$ obtained from the rule set $\lceil P \rceil$, as described by (2.1), and since the inductive set $\mathsf{Ind}(\lceil P \rceil)$ is defined as the intersection of all $\mathcal{T}_{\lceil P \rceil}$-closed sets (i.e. sets E such that $\mathcal{T}_{\lceil P \rceil}(E) \subseteq E$), it follows the well-known result $\mathcal{M}_P^{\mathcal{C}} = \mathsf{Ind}(\lceil P \rceil) = \mathsf{lfp}(\mathcal{T}_{\lceil P \rceil})$. Furthermore, since the body of each clause contains a finite number of atoms, the rule set $\lceil P \rceil$ is finitary and therefore $\mathcal{T}_{\lceil P \rceil}$ is upward continuous and we obtain the equality (2.4) which only follows from properties of inductive definitions: $\mathcal{M}_P^{\mathcal{C}}$ can be directly expressed by an inductive definition from the

rule set $\lceil P \rceil$. This 'logic programs as inductive definitions' paradigm has been used both to explain the foundations of logic programming [6, 14] and to extend logic programming languages in order to increase the power of 'pure' declarative programming [15, 16, 32].

Atoms computed at infinity and greatest fixpoints
Unfortunately, the identification of a logic program with a set of co-inductive definitions is not fruitful when we consider infinite derivations. In fact, in this setting, the denotation of a program P is defined as the set of theorems which are the results of a possibly infinite number of applications of instances of clauses in P (viewed as an inference system). So, when the domain of computations is extended with infinite terms, the denotation of P, defined in terms of greatest fixpoint, corresponds to the co-inductive set $\mathsf{CoInd}(\llbracket P \rrbracket)$ (where $\llbracket P \rrbracket$ denotes all the ground possibly infinite instances of clauses in P). However, as we said, if we assign some meaning to infinite derivations by considering the set C_P containing all the (ground) infinite atoms that can be computed at infinity, then a greatest fixpoint construction does not lead to a complete semantics. Indeed, whereas for programs, like program (3.1), such an approach is sound and complete since we have $C_P = \mathsf{CoInd}(\llbracket P \rrbracket)$ ($p(f^\omega)$ is both in the greatest fixpoint of the operator $\mathcal{T}_{\llbracket P \rrbracket}$ and computable by the infinite derivation (3.2)), for programs, like program (1.2), such an approach is not complete. Clearly, with program (1.2), $p(f^\omega)$ is not computable by an infinite derivation but we have $p(f^\omega) \in \mathsf{CoInd}(\llbracket P \rrbracket)$ since the clause $p(f^\omega) \leftarrow p(f^\omega)$ is in $\llbracket P \rrbracket$ and we can apply it at infinity. Therefore $\{p(f^\omega)\}$ is $\llbracket P \rrbracket$-dense (i.e. $\mathcal{T}_{\llbracket P \rrbracket}$-dense). The incompleteness comes from the fact that clauses of $\llbracket P \rrbracket$ are expressed over a language richer than the language of clauses of P and the language of queries: by allowing infinite elements in queries and programs, such an approach becomes complete. For example, with program (1.2) we can obtain derivation (1.3) from the query $p(f^\omega)$. Therefore, in the following subsections we investigate infinite derivations over the domain of finite terms. As we will see, there exists a complete greatest fixpoint semantics for these derivations.

4.2 *Proof trees and fair derivations*

Recall that one of the underlying ideas of logic programming is to consider a computation as the extraction of a result from a proof: successful derivations and fair infinite derivations are proofs. First, we define SLD-proofs (for the operational semantics based on SLD-resolution) and the classical notion of proof trees (for the declarative semantics based on fixpoint methods).

DEFINITION 4.1
An **SLD-proof** is either an SLD-refutation or a fair infinite SLD-derivation.

DEFINITION 4.2
Given a rule set Φ over $\mathcal{B}$, a **proof tree** of $x \in \mathcal{B}$ for Φ is a possibly infinite tree T such that x is the root of T, and for every node z occurring in T with $z_1, \cdots, z_n$ as children, there exists a rule $z \leftarrow z_1, \cdots, z_n \in \Phi$ (in particular, if z is a leaf, there exists a rule $z \leftarrow \in \Phi$).

In the following, we say that T is a *partial proof tree* if T is a proof tree whose leaves do not necessarily correspond to a (unit) rule. We have the following well-known lemma.

LEMMA 4.3
$x \in \mathsf{CoInd}(\Phi)$ iff x is the root of a proof tree for Φ (furthermore, the proof tree is finite iff

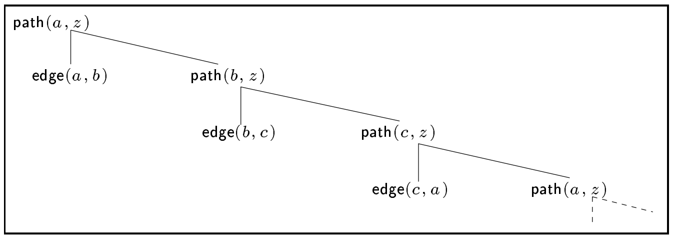


FIGURE 2. Proof tree of $\mathsf{path}(a,z)$

$x \in \mathsf{Ind}(\Phi)$ for finitary[2] Φ).

As we will see, it is easy to obtain a proof tree from a derivation and the soundness result will be obtained in a direct way. Conversely, in order to prove the completeness of our approach, we will need to be able to 'translate' a proof tree into an SLD-derivation. Proposition 4.5 shows (in a constructive way) how this translation can be done. Its proof requires Lemma 4.4 expressing properties about variables renaming. In fact, it is important to note that it is necessary to take into account the renaming process used in an SLD-derivation, often considered as a 'minor' detail, in more informal presentations. Of course, proofs are getting a bit complicated but this avoids some confusions usually due to the fact that the meaning of 'renaming' is often assumed to be simpler than its formal definition implies (see [24]). Hence, each result presented here has been proved by considering in a explicit way the renaming process. Most of proofs related to this process are incomplete or omitted here. They can be found in a research report [23].

Roughly speaking, the proof of Proposition 4.5 consists in showing how to obtain a derivation during a breadth-first traversal of a proof tree. The main difficulty comes from the fact that no properties on the variables renaming are required in the proof tree considered. Hence, in order to be able to obtain a derivation satisfying property (2.2), we have to define a renaming process during the traversal of the proof tree. For example, let us consider the rule set $\lceil P \rceil$ obtained from the program defined in Figure 1. The following instances of clauses :

$$\begin{array}{llll} c_2^1: & \mathsf{path}(a,z) \leftarrow \mathsf{edge}(a,b),\mathsf{path}(b,z) & c_2^2: & \mathsf{path}(b,z) \leftarrow \mathsf{edge}(b,c),\mathsf{path}(c,z) \\ c_2^3: & \mathsf{path}(c,z) \leftarrow \mathsf{edge}(c,a),\mathsf{path}(a,z) & & \end{array}$$

belong to $\lceil P \rceil$ and we can obtain a proof tree of $\mathsf{path}(a,z)$ for the rule set $\lceil P \rceil$ (see Figure 2). However, according to (2.2), the variable z of the clauses defining the predicate path must be renamed at each transition of the derivation starting from $\mathsf{path}(a,z)$ with program $\lceil P \rceil$:

$$\begin{aligned} \mathsf{path}(a,z) \quad & \xrightarrow{\left[\begin{smallmatrix} z_1 \\ z \end{smallmatrix}\right], c_2^1}_{\lceil P \rceil} \mathsf{edge}(a,b),\mathsf{path}(b,z) \xrightarrow{[\,],\mathsf{a1}}_{\lceil P \rceil} \mathsf{path}(b,z) \\ & \xrightarrow{\left[\begin{smallmatrix} z_2 \\ z \end{smallmatrix}\right], c_2^2}_{\lceil P \rceil} \mathsf{edge}(b,c),\mathsf{path}(c,z) \xrightarrow{[\,],\mathsf{a2}}_{\lceil P \rceil} \mathsf{path}(c,z) \\ & \xrightarrow{\left[\begin{smallmatrix} z_3 \\ z \end{smallmatrix}\right], c_2^3}_{\lceil P \rceil} \mathsf{edge}(c,a),\mathsf{path}(a,z) \xrightarrow{[\,],\mathsf{a3}}_{\lceil P \rceil} \mathsf{path}(a,z) \to_{\lceil P \rceil} \cdots . \end{aligned} \tag{4.1}$$

[2]In the sense that each rule has only finitely many premisses.

The variants of the clauses c_2^1, c_2^2 and c_2^3 used during this derivation are obtained with the following renaming substitutions:

$$r_1 = \begin{bmatrix} z \\ z_1 \end{bmatrix} \quad r_2 = \begin{bmatrix} z \\ z_2 \end{bmatrix} \quad r_3 = \begin{bmatrix} z \\ z_3 \end{bmatrix} \quad \cdots .$$

LEMMA 4.4 ([23])
Given a set of variables Z, a clause C_T, and a renaming substitution r_0 such that $range(r_0) \cap var(C_T) = \emptyset$, there exists a resolution step

$$r_0 C_T^+ \overset{C,\theta}{\to} \theta C^-$$

where C is a clause satisfying $var(C) \cap (var(r_0 C_T) \cup Z) = \emptyset$ and where θ is an idempotent renaming substitution such that $dom(\theta) = var(C^+)$. Furthermore, there exists a renaming substitution r such that $range(r) = var(C^-) \backslash var(C^+)$ and $rr_0 C_T^- = \theta C^-$.

We are now in position to prove the main result of this subsection.

PROPOSITION 4.5
Given a rule set Φ and an atom $A \in \mathsf{CoInd}(\Phi)$, there exists an SLD-proof from A with Φ as program such that, at each resolution step of the SLD-proof, the mgu used is a renaming substitution whose domain coincides with the variables occurring in the head of the rule (i.e. the clause) used.

PROOF. If $A \in \mathsf{CoInd}(\Phi)$, then, by Lemma 4.3, A is root of a proof tree T for Φ. Number the arcs emanating from each node from left to right, starting with 1. Each node can be designated (indexed) by the word obtained by concatenating the numbers of the arcs of the path leading from the root to the node (ε is the empty word). The breadth-first traversal of T produces a list $\mathcal{L}$. Since T is a proof tree for Φ, for each node $A_{\vec\imath}$ in T, there exists a rule $C_{T,\vec\imath} \in \Phi$ which can be written $A_{\vec\imath} \leftarrow A_{\vec\imath 1}, \cdots, A_{\vec\imath n_{\vec\imath}}$. We write $\prec_\ell$ the lexical order over $\mathbb{N}^\star$ and $|\vec\imath|$ the length of $\vec\imath \in \mathbb{N}^\star$. Indexes of T can also be ordered by $\prec$ as follows: $\vec\imath \prec \vec\jmath$ iff $A_{\vec\imath}$ occurs before $A_{\vec\jmath}$ in $\mathcal{L}$ (i.e. iff $((|\vec\imath| < |\vec\jmath|) \vee (|\vec\imath| = |\vec\jmath| \wedge \vec\imath \prec_\ell \vec\jmath)))$. $Z_T = \cup var(C_{T,\vec\imath})$ is the possibly infinite set of variables occurring in T. By Lemma 4.4, given a clause $C_{T,\vec\imath} \in \Phi$, a renaming substitution $r_0^{\vec\imath}$, such that $range(r_0^{\vec\imath}) \cap var(C_{T,\vec\imath}) = \emptyset$, and a set of variables $Z_{\vec\imath}$, there exists a substitution $\theta_{\vec\imath}$, a clause $C_{\vec\imath}$ and a renaming substitution $r_1^{\vec\imath} = r^{\vec\imath} r_0^{\vec\imath}$ such that:

$$\begin{array}{cc} var(C_{\vec\imath}) \cap (var(r_0^{\vec\imath} C_{T,\vec\imath}) \cup Z_{\vec\imath}) = \emptyset & r_1^{\vec\imath} C_{T,\vec\imath}^- = \theta_{\vec\imath} C_{\vec\imath}^- \\ dom(\theta_{\vec\imath}) = var(C_{\vec\imath}^+) & range(r^{\vec\imath}) = var(C_{\vec\imath}^-) \backslash var(C_{\vec\imath}^+) \end{array}$$

where $\theta_{\vec\imath}$ is an idempotent renaming substitution which is a mgu of $C_{\vec\imath}^+$ and $r_0^{\vec\imath} C_{T,\vec\imath}^+$. From $\mathcal{L}$, we can define the following sequence of resolution steps:

$$t_\varepsilon = \mathcal{S}(C_{T,\varepsilon}, [\,], Z_T), \qquad t_{\vec\imath k} = \mathcal{S}\left(C_{T,\vec\imath k}, r_0^{\vec\imath k}, Z_{\vec\imath k}\right) \quad \begin{cases} 1 \leq k \leq n_{\vec\imath} \\ r_0^{\vec\imath k} = r_1^{\vec\imath} \\ Z_{\vec\imath k} = Z_T \cup \bigcup\limits_{\vec\jmath \prec \vec\imath k} var(C_{\vec\jmath}) \end{cases}$$

where $\mathcal{S}(C_{T,\vec\imath}, r_0^{\vec\imath}, Z_{\vec\imath})$ denotes the resolution step $r_0^{\vec\imath} C_{T,\vec\imath}^+ \xrightarrow{C_{\vec\imath},\theta_{\vec\imath},Z_{\vec\imath}}_{\Phi} \theta_{\vec\imath} C_{\vec\imath}^-$ ($Z_{\vec\imath}$ denotes the variables used for the renaming during the previous transitions). This definition is sound since we can prove (see [23]) by induction over $\vec\imath$ that $\forall \vec\imath \; range(r_0^{\vec\imath}) \cap Z_T = \emptyset$. Furthermore, it can be proved [23] that this sequence defines an SLD-proof satisfying the desired properties (fairness follows from the breadth-first traversal of T). ∎

In this subsection, proof trees for a rule set Φ have been related to SLD-proofs with Φ viewed as a program. The SLD-proofs obtained are such that the clauses used are not instanciated (they are just renamed).

4.3 Infinite derivations over the domain of finite terms

SLD-proofs over the domain of finite terms are SLD-derivations which do not compute at infinity infinite terms (i.e. in which the accumulation of mgus is stable from a certain point on). In a more formal way, they can be defined as follows.

DEFINITION 4.6
An **SLD-proof over the domain of finite terms** is either a refutation or a fair infinite derivation:

$$R_0 \overset{C_1,\theta_1}{\rightarrow_P} R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow_P} R_i \rightarrow_P \cdots$$

such that $\forall k \geq 0\ \exists p > k\ \forall q \geq p \quad \theta_q \cdots \theta_p \cdots \theta_{k+1} R_k \approx \theta_p \cdots \theta_{k+1} R_k$.

Hence, for every query R_i occurring in the derivation, there exists a step after which all the used unifiers do not instanciate R_i. It is important to note that it does not suffice that the condition holds for the initial query R_0. For example, even if during the derivation with $P = \{q \leftarrow p(x); p(f(x)) \leftarrow p(x)\}$, starting from q, each mgu θ_i used is such that $\theta_i q = q$, this derivation computes at infinity the infinite term f^ω. A different characterization of SLD-proofs over the domain of finite terms can be obtained from the following lemma, used during the proof of Lemma 4.11.

LEMMA 4.7
A derivation $R_0 \overset{C_1,\theta_1}{\rightarrow_P} R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow_P} R_i \rightarrow_P \cdots$ is an infinite SLD-proof over the domain of finite terms iff $\forall i \geq 0$, $\exists R$, $\forall n \geq i+1$, $\theta_n \theta_{n-1} \cdots \theta_{i+1} R_i \leq R$, where R is a query (i.e. R does not contain infinite atoms).

PROOF. ($\Rightarrow$). By definition, $\forall k \geq 0$, $\exists p > k$ such that $\forall q \geq p$, $\theta_q \cdots \theta_p \cdots \theta_{k+1} R_k \approx \theta_p \cdots \theta_{k+1} R_k$ and it follows $\forall k \geq 0$, $\forall q \geq k+1$, $\theta_q \cdots \theta_{k+1} R_k \leq \theta_p \cdots \theta_{k+1} R_k$.
($\Leftarrow$). Let $k \geq 0$ and suppose there exists a query R of length ℓ such that $\forall n \geq k+1$, $\theta_n \theta_{n-1} \cdots \theta_{k+1} R_k \leq R$. For every atom A_i ($1 \leq i \leq \ell$) occurring in R_k, there exists an atom B_i in R such that $\forall n \geq k+1$, $\theta_n \theta_{n-1} \cdots \theta_{k+1} A_i \leq B_i$. In a classical way, atoms can be represented as partial functions from $\mathbb{N}^\star$ (words on $\mathbb{N}$) to $\Sigma \cup \Pi \cup X$ as follows: $A(u)$ is the symbol occurring in the node of the tree representation of A designated by the word obtained by concatenating the numbers of the arcs of the path from the root to the node (arcs are numbered from left to right, starting with 1). The extensional representation of such a function is $\cup\{(u, A(u))\}$ and $\mathcal{O}(A)$ denotes the set of elements u such that $A(u)$ is defined. Here, we write $|E|$ to denote the cardinality of E. Now, seeing that clearly $A_1 \leq A_2$ implies $|\mathcal{O}(A_1)| \leq |\mathcal{O}(A_2)|$, $(|\mathcal{O}(\theta_n \cdots \theta_{k+1} A_i)|)_{n \geq k+1}$ is an increasing sequence with $|\mathcal{O}(B_i)|$ as upper bound and therefore, there exists $p'_i \geq k+1$ such that $\forall q'_i \geq p'_i$, $|\mathcal{O}(\theta_{q'_i} \cdots \theta_{k+1} A_i)| = |\mathcal{O}(\theta_{p'_i} \cdots \theta_{k+1} A_i)|$. Now, since if $A_1 \leq A_2$ and $|\mathcal{O}(A_1)| = |\mathcal{O}(A_2)|$ then $|var(A_1)| \geq |var(A_2)|$, $(|var(\theta_n \cdots \theta_{k+1} A_i)|)_{n \geq p'_i}$ is a decreasing sequence with 0 as lower bound and therefore, there exists $p_i \geq p'_i$ such that $\forall q_i \geq p_i$, $|var(\theta_{q_i} \cdots \theta_{k+1} A_i)| = |var(\theta_{p_i} \cdots \theta_{k+1} A_i)|$. Now, since every θ_j ($j \geq k+1$) is idempotent, we have $\forall q_i \geq p_i$, $\theta_{q_i} \cdots \theta_{k+1} A_i \approx \theta_{p_i} \cdots \theta_{k+1} A_i$ (if θ is an idempotent substitution such that $\theta A_1 = A_2$ and if $|var(A_1)| = |var(A_2)|$ and $|\mathcal{O}(A_1)| = |\mathcal{O}(A_2)|$, then $A_1 \approx A_2$). Now, since $p =$

$\max_{1 \leq i \leq \ell}(p_i)$ is such that $\forall q \geq p$, $\theta_q \cdots \theta_p \cdots \theta_{k+1} R_k \approx \theta_p \cdots \theta_{k+1} R_k$, this leads to the conclusion that the derivation considered is a derivation over the domain of finite terms. ■

The appropriate rule set allowing to study infinite derivations over the domain of finite terms is the rule set obtained from a program P by considering all the (finite) instances, not necessarily ground, of clauses in P. This corresponds to the $\mathcal{C}$-semantics approach. $\mathcal{C}$-semantics results are concerned with SLD-refutations. Let us now investigate infinite SLD-proofs over the domain of finite terms in the framework of the $\mathcal{C}$-semantics. The soundness theorem can be proved in a direct way by using proof trees.

PROPOSITION 4.8 (Soundness)
If there exists an SLD-proof over the domain of finite terms

$$A_1, \cdots, A_n \overset{C_1,\theta_1}{\rightarrow_P} R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow_P} R_i \rightarrow_P \cdots$$

then there exists $k \geq 0$, such that for all i $(1 \leq i \leq n)$ $\theta_k \cdots \theta_1 A_i \in \mathsf{CoInd}(\lceil P \rceil)$.

PROOF. We first prove the proposition for $n = 1$ (see Figure 3). By Lemma 4.3, it suffices to prove that for a natural k, there exists a proof tree of $\theta_k \cdots \theta_1 A_1$ for $\lceil P \rceil$. For this, let us define the sequence $T_1, \cdots, T_i, \cdots$ of partial proof trees, such that every atom occurring in R_i is a leaf in T_i, which is a partial proof tree of $\theta_i \cdots \theta_1 A_1$ for $\lceil P \rceil$. T_1 is obtained from the first transition: its root is $\theta_1 A_1$ whose children (which are leaves) are all the atoms occurring in $\theta_1 C_1^-$. Since $\theta_1 C_1 \in \lceil P \rceil$ and $\theta_1 A_1 = \theta_1 C_1^+$, T_1 is a partial proof tree of $\theta_1 A_1$ for $\lceil P \rceil$. Suppose now that T_{n-1} is a partial proof tree of $\theta_{n-1} \cdots \theta_1 A_1$ for $\lceil P \rceil$ (corresponding to the $n-1$ first transitions) such that atoms in R_{n-1} are leaves of T_{n-1}. By applying the substitution θ_n to each node of T_{n-1}, we get a partial proof tree of $\theta_n \cdots \theta_1 A_1$ for $\lceil P \rceil$ such that atoms in $\theta_n R_{n-1}$ are leaves. If A is the selected atom in R_{n-1}, then A is a leaf of T_{n-1} and $\theta_n A$ is a leaf in the new partial proof tree. Now, it suffices to add all the atoms in $\theta_n C_n^-$ as children of $\theta_n A$ (these children are leaves). In this way, we obtain a partial proof tree T_n satisfying the desired properties since $R_n = \theta_n R_{n-1}[k \leftarrow C_n^-]$. Because the derivation does not compute infinite terms and therefore there exists a natural $k \geq 0$ such that for all $q \geq k$, $\theta_q \cdots \theta_k \cdots \theta_1 A_1 \approx \theta_k \cdots \theta_1 A_1$, by iterating this process, we obtain a proof tree of $\theta_k \cdots \theta_1 A_1$ for $\lceil P \rceil$. Furthermore each leaf corresponds to a unit clause of $\lceil P \rceil$ since the derivation is fair.

For $n > 1$, the proof is similar: instead of building a sequence of partial proof trees, we build a sequence $((T_1^1, \cdots, T_1^n), \cdots, (T_i^1, \cdots, T_i^n), \cdots)$ of tuples of n partial proof trees for $\lceil P \rceil$ such that $\theta_i \cdots \theta_1 A_j$ is the root of T_i^j $(1 \leq j \leq n)$ and each atom occurring in R_i is a leaf of T_i^j for a j. ■

Since, by Proposition 4.5, there exists an SLD-proof with the program $\lceil P \rceil$ from each atom occurring in $\mathsf{CoInd}(\lceil P \rceil)$, we have to show how to 'translate' an SLD-derivation with $\lceil P \rceil$ into an SLD-derivation with P (for example, how to obtain derivation (1.1) from derivation (4.1)). Lemma 4.11 describes such a translation and can be viewed as a 'program lifting lemma' playing the same role as the (classical) lifting lemma in the proof of the (classical) completeness theorem for refutations. Its proof requires the two following technical lemmas.

LEMMA 4.9 ([23])
Let A, A_0 and B be three atoms, and θ be a substitution such that $\theta A = \theta B$ and $\theta A_0 = A_0$. There exists a mgu ρ of A and B such that $\rho A_0 = A_0$.

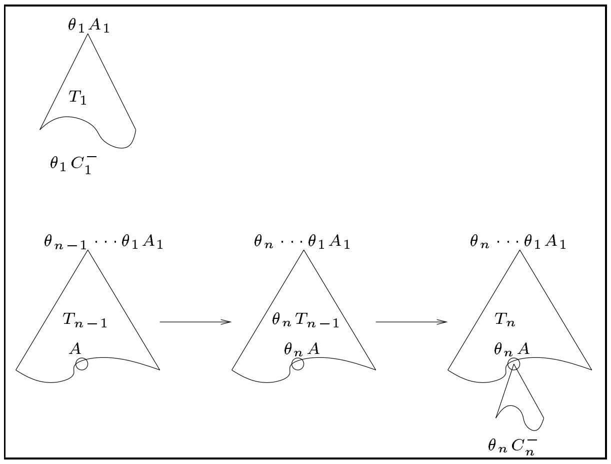


FIGURE 3. Soundness

LEMMA 4.10 ([23])
Let A_1 and A_2 be two atoms such that $var(A_1)\cap var(A_2)=\emptyset$. If for a substitution θ, such that $dom(\theta)\subseteq var(A_2)$, $A_1=\theta A_2$, then θ is a mgu of A_1 and A_2.

LEMMA 4.11 (Program lifting lemma)
If there exists an SLD-proof :

$$A_0 \overset{C_1,\theta_1}{\rightarrow}_{\lceil P\rceil} R_1 \rightarrow_{\lceil P\rceil} \cdots \rightarrow_{\lceil P\rceil} R_{i-1} \overset{C_i,\theta_i}{\rightarrow}_{\lceil P\rceil} R_i \rightarrow_{\lceil P\rceil} \cdots$$

such that for all $i\geq 1$, θ_i is an idempotent renaming substitution such that $dom(\theta_i)=var(C_i^+)$, then there exists an SLD-proof over the domain of finite terms:

$$A_0 \overset{C_{P,1},\sigma_1}{\rightarrow}_P R'_1 \rightarrow_P \cdots \rightarrow_P R'_{i-1} \overset{C_{P,i},\sigma_i}{\rightarrow}_P R'_i \rightarrow_P \cdots$$

such that for all $i\geq 1$, $\sigma_i A_0 = A_0$, $C_i=\mu_i C_{P,i}$ and $R_i=\rho_i R'_i$ where ρ_i is the restriction of $\theta_i\mu_i\theta_{i-1}\mu_{i-1}\cdots\theta_1\mu_1$ to the variables occurring in R'_i.

PROOF. It can be proved (see [23]) that there exists a set $\{C_{P,1},\cdots,C_{P,i},\cdots\}$ of variants of clauses of P such that:

$$\forall i>0\quad var(C_{P,i})\cap(var(A_0)\cup\cup_{1\leq j<i}var(C_{P,j})\cup\cup_{j\geq 1}var(C_j))=\emptyset$$

and such that each $C_{P,i}$ satisfies $C_i=\mu_i C_{P,i}$ where μ_i is an idempotent substitution such that $dom(\mu_i)=var(C_{P,i})$.
• *For the first transition.* By definition $\theta_1 A_0=\theta_1 C_1^+=\theta_1\mu_1 C_{P,1}^+$. Furthermore, since $dom(\mu_1)=var(C_{P,1})$ and $var(C_{P,1})\cap var(A_0)=\emptyset$, we have $\mu_1 A_0=A_0$ and it follows that $\theta_1\mu_1 A_0=\theta_1\mu_1 C_{P,1}^+$. Similarly, since $dom(\theta_1)=var(C_1^+)$ and $var(C_1)\cap var(A_0)=\emptyset$, we have $\theta_1 A_0=A_0$. Therefore, we have $\theta_1\mu_1 A_0=A_0=\theta_1\mu_1 C_{P,1}^+$. By Lemma 4.10, the restriction σ_1 of $\theta_1\mu_1$ to $var(C_{P,1}^+)$ is a mgu of A_0 and $C_{P,1}^+$ and we get the transition

$$A_0 \overset{C_{P,1},\sigma_1}{\rightarrow}_P R'_1.$$

Clearly, we have $\sigma_1 A_0 = A_0$ since $\theta_1\mu_1 A_0 = A_0$. Now, let us prove that the restriction ρ_1 of $\theta_1\mu_1$ to $var(R'_1)$ satisfies $\rho_1 R'_1 = R_1$. For this, we have to prove that $\rho_1 R'_1 = \rho_1\sigma_1 C^-_{P,1} = \theta_1\mu_1 C^-_{P,1} = \theta_1 C^-_1 = R_1$. Let $v \in var(C^-_{P,1})$, two cases are possible. If $v \in var(C^+_{P,1})$, then $\sigma_1 v = \theta_1\mu_1 v$ and we can conclude since $\theta_1\mu_1\theta_1\mu_1 v = \theta_1\mu_1 v$. Else, if $v \notin var(C^+_{P,1})$, then we have $\rho_1\sigma_1 v = \rho_1 v = \theta_1\mu_1 v$ which settles the claim.

• *For the* i*th transition.* Let us show how from the transition $R_{i-1} \overset{C_i,\theta_i}{\rightarrow}_{\lceil P\rceil} R_i$ we can obtain a transition

$$R'_{i-1} \overset{C_{P,i},\sigma_i}{\rightarrow}_P R'_i$$

from R'_i satisfying $\rho_{i-1}R'_{i-1} = R_{i-1}$ where ρ_{i-1} is the restriction of $\theta_{i-1}\mu_{i-1}\cdots\theta_1\mu_1$ to $var(R'_{i-1})$, such that $\sigma_i A_0 = A_0$ and such that the restriction ρ_i to $var(R'_i)$ of $\theta_i\mu_i\theta_{i-1}\mu_{i-1}\cdots\theta_1\mu_1$ satisfies $\rho_i R'_i = R_i$. If A is the selected atom in R_{i-1} at position k, then there exists an atom A' occurring at position k in R'_{i-1} such that $A = \rho_{i-1}A'$ and we get $\theta_i\rho_{i-1}A' = \theta_i\mu_i C^+_{P,i}$. From:

$$\begin{array}{rl} & dom(\rho_{i-1}) \subseteq var(R'_{i-1}) \subseteq (\cup_{1\le j<i} var(C_{P,j}) \cup var(A_0)) \\ \text{and} & var(C_{P,i}) \cap (\cup_{1\le j<i} var(C_{P,j}) \cup var(A_0)) = \emptyset \end{array}$$

it follows that $\rho_{i-1}C_{P,i} = C_{P,i}$ and therefore $\theta_i\rho_{i-1}A' = \theta_i\mu_i\rho_{i-1}C^+_{P,i}$. Furthermore, $dom(\mu_i) = var(C_{P,i})$ and we have $\mu_i A = A$. Hence $\theta_i\mu_i\rho_{i-1}A' = \theta_i\mu_i\rho_{i-1}C^+_{P,i}$, and since $\theta_i\mu_i\cdots\theta_1\mu_1 A_0 = A_0$, by Lemma 4.9, there exists a mgu σ_i of A' and $C^+_{P,i}$ such that $\sigma_i A_0 = A_0$ and for a substitution η_i, we have $\eta_i\sigma_i = \theta_i\mu_i\rho_{i-1}$. Hence, we get the desired transition. In order to relate R_i to R'_i, let us prove that $\theta_i\mu_i\rho_{i-1}\sigma_i = \theta_i\mu_i\rho_{i-1}$. Since σ_i is idempotent, we have $\theta_i\mu_i\rho_{i-1}\sigma_i = \eta_i\sigma_i\sigma_i = \eta_i\sigma_i = \theta_i\mu_i\rho_{i-1}$. We are now in position to prove $\theta_i\mu_i\rho_{i-1}R'_i = R_i$:

$$\begin{array}{rll} \theta_i\mu_i\rho_{i-1}R'_i = & \theta_i\mu_i\rho_{i-1}\sigma_i R'_{i-1}[k \leftarrow C^-_{P,i}] & \\ = & \theta_i\mu_i\rho_{i-1}R'_{i-1}[k \leftarrow C^-_{P,i}] & (\theta_i\mu_i\rho_{i-1}\sigma_i = \theta_i\mu_i\rho_{i-1}) \\ = & \theta_i\rho_{i-1}R'_{i-1}[k \leftarrow \mu_i C^-_{P,i}] & (\mu_i R_{i-1} = R_{i-1}) \\ = & \theta_i(\rho_{i-1}R'_{i-1})[k \leftarrow \mu_i C^-_{P,i}] & (\rho_{i-1}C_i = C_i)) \\ = & \theta_i R_{i-1}[k \leftarrow C_i] & \\ = & R_i. & \end{array}$$

Therefore, the restriction ρ_i of $\theta_i\mu_i\rho_{i-1}$ to the variables occurring in R'_i satisfies $\rho_i R'_i = R_i$. To terminate, we have to prove that the derivation obtained is a derivation over the domain of finite terms. For this, let us prove that:

$$\forall n \ge i+1 \quad \rho_n\sigma_n\sigma_{n-1}\cdots\sigma_{i+1}R'_i = R_i.$$

We proceed by induction over n. For $n = i+1$, we have:

$$\rho_{i+1}\sigma_{i+1}R'_i = \theta_{i+1}\mu_{i+1}\rho_i\sigma_{i+1}R'_i = \theta_{i+1}\mu_{i+1}\rho_i R'_i = \theta_{i+1}\mu_{i+1}R_i = R_i.$$

For $n > i+1$, by induction hypothesis, we have $\rho_{n-1}\sigma_{n-1}\cdots\sigma_{i+1}R'_i = R_i$. Therefore, in order to prove $\rho_n\sigma_n\sigma_{n-1}\cdots\sigma_{i+1}R'_i = R_i$, we have to prove that for every variable v occurring in $\sigma_{n-1}\cdots\sigma_{i+1}R'_i$, $\rho_n\sigma_n v = \rho_{n-1}v$. First, note that $\rho_n\sigma_n = \theta_n\mu_n\rho_{n-1}\sigma_n =$

$\theta_n\mu_n\rho_{n-1}$, and it suffices to prove that $\theta_n\mu_n\rho_{n-1}v = \rho_{n-1}v$. Furthermore, if $v \in \sigma_{n-1}\cdots\sigma_{i+1}R'_i$, then we have:

$$v \in (var(R'_i) \cup \cup_{i+1\leq j\leq n-1} var(C_{P,j})) \subseteq (var(A_0) \cup \cup_{1\leq j\leq n-1} var(C_{P,j})).$$

Therefore, if $v \in dom(\rho_{n-1})$, then $var(\rho_{n-1}v) \subseteq R_{n-1}$ and $\theta_n\mu_n\rho_{n-1}v = \rho_{n-1}v$ since $\theta_n\mu_n R_{n-1} = R_{n-1}$, else we can also conclude since $\theta_n\mu_n v = v$. Hence, since R_i contains only finite atoms and for all $n \geq i+1$, $\sigma_n\sigma_{n-1}\cdots\sigma_{i+1}R'_i \leq R_i$, by Lemma 4.7, the derivation obtained is a derivation over the domain of finite terms. ∎

We are now in position to prove the completeness result.

PROPOSITION 4.12 (Completeness)
If $A \in \mathsf{CoInd}(\lceil P\rceil)$, then there exists an SLD-proof over the domain of finite terms

$$A \overset{C_{P,1},\sigma_1}{\rightarrow_P} R'_1 \rightarrow_P \cdots \rightarrow_P R'_{i-1} \overset{C_{P,i},\sigma_i}{\rightarrow_P} R'_i \rightarrow_P \cdots$$

such that for all $i \geq 1$, $\sigma_i A = A$

PROOF. Immediate by Lemma 4.3, Proposition 4.5 and Lemma 4.11. ∎

4.4 Infinite derivations which do not compute anything

The derivation obtained by Proposition 4.5, like derivation (4.1), is a special case of derivations which do not compute infinite terms: such a derivation does not compute anything since the mgus used are just renaming substitutions. In a more general way, we will say that a derivation does not compute anything if each mgu used is just an instantiation of the clause used and does not instantiate variables occurring in the derivation. Hence, such derivations can be viewed as rewriting sequences and we will call them restricted SLD-proofs. As claimed in [29], many results for logic programs have corresponding results for constraint logic programs and, in [20], results on fixpoint semantics of constraint logic programs are closely related to the results proved in this subsection. This particular class of derivations can be considered within the 'logic programs as (co-)inductive definitions' paradigm obtained with the rule set $\lceil P\rceil$. However, there is a more restricted rule set which captures the semantics of this class of derivations and as we will see, we can also give a semantics to non-necessarily fair derivations in this framework. In this subsection, we present the main results obtained for these derivations.

DEFINITION 4.13
A **restricted SLD-proof** is an SLD-derivation:

$$R_0 \overset{C_1,\theta_1}{\rightarrow_P} R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow_P} R_i \rightarrow_P \cdots$$

such that for every $i \geq 1$, $dom(\theta_i) \subseteq var(C_i^+)$. In particular, a restricted SLD-refutation is a restricted SLD-proof ending with the empty query.

Of course, every fair restricted SLD-proof is an SLD-proof over the domain of finite terms.

Finite restricted SLD-proofs
It is possible to give a 'model-theoretic' semantics to finite restricted SLD-proofs by extending the notion of $\mathcal{C}$-truth [12] for clauses as follows. Given a $\mathcal{C}$-interpretation I, a clause

$A \leftarrow B_1, \cdots, B_q$ is $\mathcal{C}^+$*-true* in I iff for every substitution θ such that $dom(\theta) \subseteq var(A)$, if atoms θB_i ($1 \leq i \leq q$) are $\mathcal{C}$-true in I, then θA is $\mathcal{C}$-true in I. This notion extends the $\mathcal{C}$-truth since if a clause is $\mathcal{C}$-true in I then it is $\mathcal{C}^+$-true in I. A $\mathcal{C}$-interpretation I is a $\mathcal{C}^+$*-model* of a program P if every clause in P is $\mathcal{C}^+$-true in I. Hence, every $\mathcal{C}$-model of P is a $\mathcal{C}^+$-model of P. However, every $\mathcal{C}^+$-model of P is not necessarily a $\mathcal{C}$-model of P. For example, if we consider the program:

$$P = \{p(x) \leftarrow p(y) \ ; \ p(f(w)) \leftarrow\} \tag{4.2}$$

the $\mathcal{C}$-interpretation $I = \lceil\{p(f(z))\}\rceil$ is clearly a $\mathcal{C}^+$-model of P, since for every substitution θ such that $dom(\theta) \subseteq var(p(x))$, $\theta p(y) = p(y) \notin I$. But, I is not a $\mathcal{C}$-model of P, since with the substitution $\theta = \{y/f(y)\}$, we have $\theta p(y) = p(f(y)) \in I$ but $\theta p(x) = p(x) \notin I$. However, $\mathcal{C}^+$-models satisfy the model intersection property and are useful to define a declarative semantics for restricted SLD-refutations in terms of *least* $\mathcal{C}^+$*-model* of a program P, written $\mathcal{M}_P^{\mathcal{C}^+}$.

The 'logic program as (co-)inductive definition' paradigm is obtained by considering the rule set associated with P defined by:

$$\lceil P\rceil^+ = \{\theta C \mid C \in P \text{ and } dom(\theta) \subseteq var(C^+)\}$$

which is associated, as described by (2.1), with the monotone operator $\mathcal{T}_{\lceil P\rceil^+}$ satisfying the following standard properties (proved in [23]):

– $\mathcal{T}_{\lceil P\rceil^+}$ is monotone and upward continuous.
– A $\mathcal{C}$-interpretation I is a $\mathcal{C}^+$-model of a program P iff $\mathcal{T}_{\lceil P\rceil^+}(I) \subseteq I$.
– $\mathcal{M}_P^{\mathcal{C}^+} = \mathsf{lfp}(\mathcal{T}_{\lceil P\rceil^+}) = \mathsf{Ind}(\lceil P\rceil^+) = \mathcal{T}_{\lceil P\rceil^+}^{\uparrow\omega}$.

$\mathcal{C}^+$-semantics is a special case of $\mathcal{C}$-semantics, and not surprisingly, we have $\mathcal{M}_P^{\mathcal{C}^+} \subseteq \mathcal{M}_P^{\mathcal{C}}$. However, the converse inclusion does not hold: for example, with program (4.2), we have $\mathcal{M}_P^{\mathcal{C}^+} = \lceil\{p(f(x))\}\rceil$ and $\mathcal{M}_P^{\mathcal{C}} = \lceil\{p(x)\}\rceil$. For restricted SLD-refutations, we have standard results (proved in [23]).

PROPOSITION 4.14 (Soundness and Completeness [23])
There exists a restricted SLD-refutation from the query $A_1, \cdots, A_q$ iff $\{A_1, \cdots, A_q\} \subseteq \mathcal{M}_P^{\mathcal{C}^+}$.

Therefore, atoms in $\mathcal{M}_P^{\mathcal{C}} \backslash \mathcal{M}_P^{\mathcal{C}^+}$ are atoms from which SLD-refutations, but no restricted ones, exist. For example, with program (4.2), whereas $p(z) \in \mathcal{M}_P^{\mathcal{C}}$, $p(z) \notin \mathcal{M}_P^{\mathcal{C}^+}$. Even if there exists an SLD-refutation:

$$p(z) \begin{bmatrix} x_1 \\ z \\ \rightarrow_P \end{bmatrix} p(y_1) \begin{bmatrix} y_1 \\ f(w_1) \\ \rightarrow_P \end{bmatrix} \square \tag{4.3}$$

there is no restricted SLD-refutation from $p(z)$. However, for the class of programs P whose clauses do not contain local variables (i.e. such that $var(C^-) \subseteq var(C^+)$), we have $\lceil P\rceil^+ = \lceil P\rceil$ (and then $\mathcal{M}_P^{\mathcal{C}} = \mathcal{M}_P^{\mathcal{C}^+}$). As proved in [28], another important property satisfied by these programs is $\mathcal{T}_{\lceil P\rceil}^{\downarrow\omega} = \mathsf{gfp}(\mathcal{T}_{\lceil P\rceil})$.

Infinite restricted SLD-proofs
For fair infinite restricted SLD-proofs, we can prove results similar to those of SLD-proofs

over the domain of finite terms. Soundness and completeness properties are easily obtained. But, as we will see at the end of this subsection, we can also prove a supplementary result concerning unfair infinite derivations which do not compute anything. Let us first prove the soundness result which is proved in a different way than proof of Proposition 4.8.

PROPOSITION 4.15 ($\mathcal{C}^+$-soundness)
If there exists a restricted SLD-refutation from A_0 or a fair infinite restricted SLD-proof:

$$A_0 \overset{C_1,\theta_1}{\to_P} R_1 \to_P \cdots \to_P R_{i-1} \overset{C_i,\theta_i}{\to_P} R_i \to_P \cdots$$

then $A_0 \in \mathsf{CoInd}(\lceil P \rceil^+)$.

PROOF. It suffices to prove that there exists a $\mathcal{T}_{\lceil P \rceil^+}$-dense set containing A_0. Let us prove that $\cup_{i\geq 1}\theta_i C_i^+$ satisfies these two properties. First, note that by definition, $A_0 = \theta_1 A_0 = \theta_1 C_1^+ \subseteq \cup_{i\geq 1}\theta_i C_i^+$. Now, we have to prove $\cup_{i\geq 1}\theta_i C_i^+ \subseteq \mathcal{T}_{\lceil P \rceil^+}\left(\cup_{i\geq 1}\theta_i C_i^+\right)$. If $A \in \cup_{i\geq 1}\theta_i C_i^+$, then there exists a natural $k \geq 1$ such that $A = \theta_k C_k^+$ and, since $dom(\theta_k) \subseteq var(C_k^+)$ it follows that $\theta_k C_k \in \lceil P \rceil^+$. It suffices to prove that each atom occurring in $\theta_k C_k^-$ occurs in $\cup_{i\geq 1}\theta_i C_i^+$. If $A_k \in \theta_k C_k^-$, then $A_k \in R_k$ and since the derivation is either a refutation or a fair infinite derivation, there exists a resolution step in which the residue (i.e. the further instantiated version) of A_k is the selected atom:

$$\cdots \overset{C_k,\theta_k}{\to_P} R_k \to_P \cdots \to_P R_m \overset{C_{m+1},\theta_{m+1}}{\to_P} R_{m+1} \to_P \cdots .$$

Hence, for $m \geq k$, we have $\theta_{m+1}\cdots\theta_{k+1}A_k = \theta_{m+1}C_{m+1}^+$. Since variables occurring in C_i do not occur in clauses C_j $(j < i)$ and since, each mgu θ_j is such that $dom(\theta_j) \subseteq var(C_j^+)$, it follows $\theta_{m+1}\cdots\theta_{k+1}A_k = A_k = \theta_{m+1}C_{m+1}^+ \subseteq \cup_{i\geq 1}\theta_i C_i^+$ and we can conclude. ∎

The completeness theorem for infinite restricted SLD-proofs is proved by using the following technical lemma.

LEMMA 4.16 ([23])
If there exists a transition $R_0 \overset{\mu C,\theta}{\to} R_1$ such that , $var(C) \cap var(R_0) = \emptyset$, $dom(\theta) = (\mu C^+)$ and $dom(\mu) = var(C^+)$, then there exists a transition

$$R_0 \overset{C,\sigma}{\to} R_1$$

such that $dom(\sigma) = var(C^+)$.

PROPOSITION 4.17 ($\mathcal{C}^+$-completeness)
If $A \in \mathsf{CoInd}(\lceil P \rceil^+)$, then there exists a restricted SLD-refutation or a fair infinite restricted SLD-proof from A with P.

PROOF. If $A \in \mathsf{CoInd}(\lceil P \rceil^+)$, then by Proposition 4.5, there exists a restricted refutation or a fair infinite SLD-proof from A with $\lceil P \rceil^+$ such that for all $i \geq 1$, the mgu θ_i, used during the ith resolution step of the SLD-proof, is a renaming substitution whose domain coincides with the variables occurring in the head of the clause used:

$$A \overset{C_1,\theta_1}{\to_{\lceil P \rceil^+}} R_1 \to_{\lceil P \rceil^+} \cdots \to_{\lceil P \rceil^+} R_{i-1} \overset{C_i,\theta_i}{\to_{\lceil P \rceil^+}} R_i \to_{\lceil P \rceil^+} \cdots .$$

It can be proved [23] that there exists a set $\{C_{P,1}, \cdots, C_{P,i}, \cdots\}$ of variants of clauses of P such that:

$$\forall i > 0 \quad var(C_{P,i}) \cap (var(A_0) \cup \cup_{1\leq j<i} var(C_{P,j})) = \emptyset$$

and such that each clause $C_{P,i}$ satisfies $C_i = \mu_i C_{P,i}$ where μ_i is an idempotent substitution such that $dom(\mu_i) = var(C^+_{P,i})$. Then, by Lemma 4.16, there exists a restricted SLD-proof from A with P. ∎

Non-necessarily fair infinite SLD-derivations which do not compute anything can be viewed as partial proofs. Recall that given a derivation

$$R_0 \overset{C_1,\theta_1}{\rightarrow_P} R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow_P} R_i \rightarrow_P \cdots$$

for all $i \geq 1$ we have the well-known result $P \models R_i \Rightarrow P \models \theta_i \cdots \theta_1 R_0$. This result can be 'generalized' for restricted SLD-proofs by considering:

$$R_\infty = \bigcup_{p \geq 0} \bigcap_{p \leq n} R_n$$

containing all the persisting atoms in residual queries.

PROPOSITION 4.18
If there exists an infinite restricted SLD-proof

$$R_0 = A_0 \overset{C_1,\theta_1}{\rightarrow_P} R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow_P} R_i \rightarrow_P \cdots$$

then $R_\infty \subseteq \mathsf{CoInd}(\lceil P \rceil^+)$ implies $A_0 \in \mathsf{CoInd}(\lceil P \rceil^+)$.

PROOF. Suppose that $\cup_{p\geq 0} \cap_{p \leq n} R_n \subseteq \mathsf{CoInd}(\lceil P \rceil^+)$ and let us prove that $A_0 \in \mathsf{CoInd}(\lceil P \rceil^+)$. For this, by Lemma 4.3, it suffices to prove that there exists a proof tree T of A_0 for $\lceil P \rceil^+$. Let us define the sequence $T_1, \cdots, T_i, \cdots$ of partial proof trees such that every atom occurring in R_i is a leaf in T_i. T_1 is obtained by considering the first transition: its root $A_0 = \theta_1 A_0$ has atoms in $R_1 = \theta_1 C_1^-$ as children. We show now how we can obtain T_n from T_{n-1}. We know that atoms occurring in R_{n-1} are leaves in T_{n-1}. Let A be the selected atom in R_{n-1}, since $dom(\theta_n) \subseteq var(C_n^+)$, T_n is obtained by adding atoms occurring in $\theta_n C_n^-$ as children of A. Since, $R_n = \theta_n R_{n-1}[k \leftarrow C_n^-] = R_{n-1}[k \leftarrow \theta_n C_n^-]$, T_n is a partial proof tree of A_0 for $\lceil P \rceil^+$ such that every atom occurring in R_n is a leaf in T_n.

By iterating this process, we obtain a partial proof tree T_∞ whose leaves are either the head of a unit clause in $\lceil P \rceil^+$ or an atom in R_∞, which is, by hypothesis, in $\mathsf{CoInd}(\lceil P \rceil^+)$ and correspond, by Lemma 4.3, to the root of a proof tree for $\lceil P \rceil^+$. Therefore, by adding in T_∞ these proof trees at the corresponding leaf, we obtain a proof tree of A_0 for $\lceil P \rceil^+$ (see Figure 4). ∎

This proposition is not a special case of Proposition 4.17, it just gives another way to interpret a subclass of infinite derivations. For example, if we consider the derivation we can obtain with program (1.2) from the query $p(x)$, then by Proposition 4.17 we have $p(x) \in \mathsf{CoInd}(\lceil P \rceil^+)$ while by Proposition 4.18, we just have $p(x) \in \mathsf{CoInd}(\lceil P \rceil^+) \Rightarrow p(x) \in \mathsf{CoInd}(\lceil P \rceil^+)$ since for this derivation we have $R_\infty = p(x)$. $\mathcal{C}^+$-semantics works well to give a semantics to programs whose clauses do not contain local variables (i.e. $var(C^-) \subseteq var(C^+)$). However, derivations, like derivation (4.3), are not considered in this approach.

5 Conclusion — future work

Our work focuses on programs that do not compute infinite objects in order to avoid previous incompleteness results due to problems in 'computing an infinite object in the limit'. Indeed,

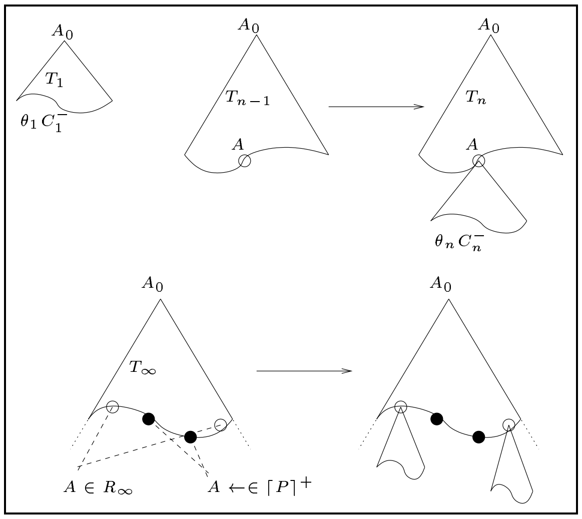


FIGURE 4. Proof of Proposition 4.18

the greatest fixpoint approach cannot always be constructed as a simple limit in the way that the least fixpoint can and we have seen that the 'logic programs as co-inductive definitions' paradigm is not equivalent to the operational notion of atoms computed at infinity. We believe the root of the difficulty lies with this notion of computation at infinity which is not captured by a greatest fixpoint construction (there are atoms which are not computable at infinity and yet belong to the greatest fixpoint). We think that the operational notion of 'computability at infinity' (associated with infinite derivations computing infinite terms) could be better captured by a least fixpoint characterisation as done in [26, 31]. Since the relationship between infinite derivations and the greatest fixpoint construction is not quite satisfactory, and sometimes unclear, we have tried to explain in a proof-theoretic framework why most attempts to give a complete semantics to derivations computing infinite terms have not been successful by representing clauses as rules of an inference system. Therefore, instead of trying to capture the notion of atoms that can be computed at infinity, we have considered atoms that can be 'proved at infinity'. However, since (programs and) queries are written in finite terms, we claim that a greatest fixpoint semantics is not suitable for capturing the notion of computation at infinity.

For these reasons, a semantics for the class of infinite derivations which do not compute infinite terms has been defined and proved sound and complete by using purely proof-theoretic methods. Infinite derivations over the domain of finite terms correspond to the operational counterpart of the greatest fixpoint of the one-step-inference operator for $\mathcal{C}$-semantics: an atom is the starting point of an infinite derivation over the domain of finite terms if and only if it is in the greatest fixpoint of the transformation $\mathcal{T}_{\lceil P \rceil}$. Of course, one may question the relevance of this work. After all, an infinite process which does not compute an infinite object can always be transformed in a process computing an infinite object: the trace of its execution. However, in the context of logic programming such a trace is nothing else than the derivation and our approach has tried to convey that the derivation (viewed as a proof) is what is important here.

Of course, a satisfactory semantics for all infinite derivations from a logic program has not yet been found. However, even if the results proved in this paper may seem unsurprising[3], they allow us to gain a better understanding of the problem. In fact, current approaches in this area only give meaning to infinite derivations that compute at least an infinite term and ignore derivations over the domain of finite terms. Now, we would like to focus on infinite derivations which do not compute anything in order to look at an infinite derivation as an infinite rewriting sequence and in order to give an interpretation in the context of logic programming of the results existing in (infinite) rewriting theory.

Acknowledgements

Many thanks to René Lalement, Catherine Dubois and Baudouin Le Charlier for enlightening discussions about this work, as well as the anonymous referees for some very useful comments.

References

[1] M. A. Nait Abdallah. On the interpretation of infinite computations in logic programming. In *11th International Colloquium on Automata, Languages and Programming, ICALP'84*, J. Paredaens, ed. pp. 358–370. Volume 172 of *Lecture Notes in Computer Science*, Springer-Verlag, 1984.

[2] M.A. Nait Abdallah and M.H. van Emden. Top-down semantics of fair computations of logic programs. *Journal of Logic Programming*, **2**, 67–76, 1985.

[3] P. Aczel. An introduction to inductive definitions. In *Handbook of Mathematical Logic*, K. J. Barwise, ed. Studies in Logic and Foundations of Mathematics. North Holland, 1977.

[4] K. R. Apt. Logic programming. In *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, J. van Leewen, ed. pp. 493–574. The MIT Press, New York, 1990.

[5] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, **25**, 95–169, 1983.

[6] P. Cousot and R. Cousot. Inductive definbitions, semantics and abstract interpretations. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 83–94. ACM, January 1992.

[7] F. S. de Boer, A. Di Pierro, and C. Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theoretical Computer Science*, **151**, 37–78, 1995.

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

[9] P. Deransart and J. Maluszynski. *A Grammatical View of Logic Programming*. The MIT Press, 1993.

[10] N. Dershowitz, S. Kaplan, and D. A. Plaisted. Rewrite, rewrite, rewrite, rewrite, rewrite. *Theoretical Computer Science*, **83**, 71–96, 1991.

[11] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A model-theoretic reconstruction of the operational semantics of logic programs. *Information and Computation*, **103**, 86–113, 1993.

[12] M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science*, **69**, 289–318, 1989.

[13] W.G. Golson. Toward a declarative semantics for infinite objects in logic programming. *Journal of Logic Programming*, **5**, 151–164, 1988.

[14] M. Hagiya and T. Sakurai. Foundation of logic programming based on inductive definition. *New Generation Computing*, **2**, 59–77, 1984.

[15] L. Hallnäs and P. Schroeder-Heister. A proof-theoretic approach to logic programming. I. Clauses as rules. *Journal of Logic and Computation*, **1**, 261–283, 1990.

[16] L. Hallnäs and P. Schroeder-Heister. A proof-theoretic approach to logic programming. II. Programs as definitions. *Journal of Logic and Computation*, **1**, 635–660, 1990.

[3]In a similar way, the result asserting that atoms computed at infinity belongs to the greatest fixpoint of the extended one-step-inference operator may seem unsurprising.

[17] J. Hein. Completions of perpetual logic programs. *Theoretical Computer Science*, **99**, 65–78, 1992.

[18] G. Huet. A uniform approach to type theory. In *Logical Foundations of Functional Programming*, pp. 337–398. Addison-Wesley, 1990.

[19] J. Jaffar and J.L. Lassez. Constraint logic programming. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pp. 111–119, 1987.

[20] J. Jaffar and M.J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, **19 & 20**, 503–582, 1994.

[21] J. Jaffar and P. J. Stuckey. Canonical logic programs. *Journal of Logic Programming*, **3**, 143–155, 1986.

[22] J. Jaffar and P. J. Stuckey. Semantics of infinite tree logic programming. *Theoretical Computer Science*, **46**, 141–158, 1986.

[23] M. Jaume. Logic programming and co-inductive definitions. Research Report 98-140, ENPC-CERMICS, 1998.

[24] M. Jaume. A full formalisation of SLD-resolution in the calculus of inductive constructions. *Journal of Automated Reasoning, Special Issue on Formal Proof*, **23**, 347–371, 1999.

[25] J.R . Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. Infinitary lambda calculus. *Theoretical Computer Science*, **175**, 93–125, 1997.

[26] G. Levi and C. Palamidessi. Contributions to the semantics of logic perpetual processes. *Acta Informatica*, 25(6):691–711, 1988.

[27] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second, extended edition, 1987.

[28] M. J. Maher. Equivalence of logic programs. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, ed. pp. 627–658. Morgan Kaufmann, 1988.

[29] M.J. Maher. A CLP view of logic programming. In *Algebraic and Logic Programming, Third International Conference*, volume 632 of *Lecture Notes in Computer Science*, H. Kirchner and G. Levi, eds. pp. 364–383. Springer-Verlag, 1992.

[30] S. Nystrom and B. Jonsson. Indeterminate concurrent constraint programming: a fixpoint semantics for non-terminating computations. In *Logic Programming - Proceedings of the 1993 International Symposium*, D. Miller, ed. pp. 335–352. The MIT Press, 1993.

[31] C. Palamidessi, G. Levi, and M. Falaschi. The formal semantics of processes and streams in logic programming. In *Colloquia Mathematica Societatis Janos Bolyai*, **42**, 363–377, 1985.

[32] L.C. Paulson and A.W. Smith. Logic programming, functional programming, and inductive definitions. In *Proceedings of the International Workshop on Extensions of Logic Programming*, volume 475 of *Lecture Notes in Computer Science*, P. Schroeder-Heister, ed. pp. 283–310. Springer-Verlag, 1989.

[33] A. Di Pierro. *Negation and Infinite computations in Logic programming*. PhD thesis, Università di Pisa-Genova-Udine, 1994.

[34] A. Podelski, W. Charatonik, and M. Müller. Set-based failure analysis for logic programs and concurrent constraint programs. In *8th European Symposium on Programming, ESOP'99*, Lecture Notes in Computer Science, S. Doaitse Swierstra, ed. pp. 177–192. Springer-Verlag, 1999.

[35] V.A. Saraswat, M.C. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pp. 333–352, 1991.

Received September 2000