Logic programming and co-inductive definitions

Mathieu Jaume

Novembre 1998

N^o 98-140

Logic programming and co-inductive definitions

Mathieu Jaume

Abstract

This paper focuses on the assignment of meaning to infinite derivations in logic programming. Several approaches have been developped by considering infinite elements in the universe of the discourse but none are complete. By considering proofs as objects in a co-inductive set, standard properties of co-inductive definitions are used both to explain this incompleteness and to define a sound and complete semantics, based on the "logic program as co-inductive definition" paradigm, for a subclass of infinite derivations, called infinite derivations over a finite domain (i.e. derivations which do not compute infinite terms).

Programmation logique et définitions co-inductives

Résumé

Les SLD-dérivations infinies sont étudiées en identifiant un programme défini avec un ensemble de définitions co-inductives. Plusieurs approches ont déjà été développées en considérant des termes infinis dans l'univers du discours mais aucune n'a permis de définir une sémantique complète. Les définitions co-inductives fournissent un cadre adéquat pour expliquer ces phénomènes d'incomplètude et permettent de définir une sémantique valide et complète pour la classe des dérivations infinies qui ne calculent pas de termes infinis.

1 Introduction

Standard semantics of definite programs, based on the traditional paradigm "logic program as first order logic", is only concerned with refutations and then is strongly related to termination. Hence, infinite derivations are not taken into account in this classical semantics. For these derivations, there is no satisfactory semantics but several approaches. In all of them, the set proposed for the denotation of a definite program contains possibly infinite atoms. Unfortunately, the semantics obtained, based on the concept of "infinite atoms being computable at infinity", are not complete (i.e. there exist infinite atoms in the denotation of a program which are not computable by an infinite derivation). In these approaches, only infinite derivations "doing useful computations, in some sense" are considered: these derivations must compute an infinite object to be "useful". This corresponds to the informally intended meaning of infinite computations. As a typical example, with the following program:

$$P = \{\mathsf{LN}(x, [x|l]) \leftarrow \mathsf{LN}(S(x), l)\} \tag{1}$$

we can obtain, from the query $\mathsf{LN}(k, l_0)$, an infinite derivation computing at every step a better approximation of the second argument:

$$\cdots \rightarrow \mathsf{LN}(S^{i-1}(k), l_{i-1}) \left[\begin{array}{cc} x_i & l_{i-1} \\ S^{i-1}(k) & [S^{i-1}(k)|l_i] \end{array} \right] \mathsf{LN}(S^i(k), l_i) \rightarrow \cdots \tag{2}$$

The "final result" is the "limit" of the sequence of approximations and corresponds to the infinite sequence of integers starting from k. Incompleteness of these approaches comes from programs often called "bad" programs: a typical example of "bad" program is the following one.

$$P = \{p(x) \leftarrow p(x)\} \tag{3}$$

Of course, the infinite derivation:

$$p(z) \rightarrow p(z) \rightarrow \cdots \rightarrow p(z) \rightarrow \cdots \tag{4}$$

does not compute anything and the denotation of p should be empty. However, we will see that when the denotation of P is defined by a greatest fixpoint of a transformation associated with P, these predicates have a non-empty denotation.

Our approach is the exact opposite: we investigate the class of infinite derivations which do not compute infinite terms. These derivations can be

"useful" to describe the behaviour of nondeterministic programs (flowgraph programs). As stated by K.R. Apt and M.H. Van Emden [4], there is a correspondence between computations of a flowgraph F and derivations obtained from a definite program $P(F)$ associated with F. Since, the only function symbols occurring in the clauses of $P(F)$ are constant symbols, these derivations do not compute anything. Another important and practical class of function-free logic programs corresponds to DATALOG programs. This special class of infinite derivations will be considered in section 5.2. However, this class of derivations is too restricted and infinite derivations computing only finite terms can be envisaged. These derivations, called derivations over a finite domain, are considered in section 5.3.

It is now well-known that standard semantics of definite programs can be expressed by purely proof-theoretic methods. The most immediate way to give such a semantics is to consider clauses as inference rules, rather than logic formulas, and then a definite program as a formal 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, inductive definitions are a natural way to define the denotation of logic programs. Since, proof-theoretically, we can look at a clause $A \leftarrow B_1, \cdots, B_n$ as an introduction rule for A (or similarly as a clause in an inductive definition), by following the Curry-Howard isomorphism, it is possible to represent clauses by constructors of a functional language and each proof can be viewed as a functional expression. Hence, there is a correspondence between proof trees and and proof terms.

In the literature, one can find several approaches dealing with infinite SLD-derivations. Most of them are based on the greatest fixpoint of operators associated to programs and then correspond to the "logic programs as co-inductive definitions" paradigm. In this way, the denotation of a definite 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 a formal system). By following the Curry-Howard isomorphism, proof terms associated with these proofs are *productive*: at each step, a constructor (i.e. a clause) is applied. In this paper, we investigate infinite SLD-derivations by considering the proof terms associated with these derivations. We will see that an approach in which an infinite SLD-derivation must compute an infinite object leads to incompleteness since with this requirement an SLD-derivation is rather viewed as a computation than as a proof. That's why we investigate the class of infinite derivations which do not compute infinite terms and therefore can be directly identified with proofs. Hence, our approach defines an empty denotation for program (1), while derivation (4)

viewed as a proof of $\forall x p(x)$, where the result proved is recursively used, leads to the definition of a non-empty denotation for program (3). We will see that there exists a sound and complete semantics for this class of derivations.

2 Background and notations

2.1 Inductive and co-inductive definitions

Inductive and 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 (for more details, see [3]). A rule is a pair (E, e), usually written $E \to e$, where E is a set, called the set of premises, and e is the conclusion. Let Φ be a rule set and A a set. A is Φ-*closed* if each rule in Φ whose premises are in A also has its conclusion in A and A is Φ-*dense* if for every $a \in A$ there is a set $E \subseteq A$ such that $(E \to a) \in \Phi$. The set inductively (resp. co-inductively) defined by a rule set Φ, written $\mathsf{Ind}(\Phi)$ (resp. $\mathsf{CoInd}(\Phi)$), is defined by $\mathsf{Ind}(\Phi) = \cap\{A,\ A \text{ is } \Phi\text{-closed}\}$ (resp. $\mathsf{CoInd}(\Phi) = \cup\{A,\ A \text{ is } \Phi\text{-dense}\}$). Φ-closed sets and Φ-dense sets exist and the intersection of any collection of Φ-closed sets is Φ-closed (in particular, $\mathsf{Ind}(\Phi)$ is the smallest Φ-closed set). Inductive and co-inductive sets can also be expressed by using *monotone* operators (an operator $\varphi : 2^{\mathcal{B}} \to 2^{\mathcal{B}}$, where $2^{\mathcal{B}}$ denotes the set of all subsets of $\mathcal{B}$, is monotone if $E_1 \subseteq E_2 \subseteq \mathcal{B}$ implies $\varphi(E_1) \subseteq \varphi(E_2)$). Given a monotone operator $\varphi : 2^{\mathcal{B}} \to 2^{\mathcal{B}}$, a set $A \subseteq \mathcal{B}$ is said to be φ-closed (resp. φ-dense) iff $\varphi(A) \subseteq A$ (resp. $A \subseteq \varphi(A)$). The set inductively (resp. co-inductively) defined by φ, written $\mathsf{Ind}(\varphi)$ (resp. $\mathsf{CoInd}(\varphi)$) is defined by $\mathsf{Ind}(\varphi) = \cap_{\varphi(A) \subseteq A \subseteq \mathcal{B}} A$ (resp. $\mathsf{CoInd}(\varphi) = \cup_{A \subseteq \varphi(A) \subseteq \mathcal{B}} A$). If Φ is a rule set, we may define a monotone operator $T_\Phi : 2^{\mathcal{B}} \to 2^{\mathcal{B}}$ as follows:

$$\mathcal{B} = \bigcup_{e \leftarrow E \in \Phi} \{\{e\} \cup E\} \quad T_\Phi(A) = \{e \in \mathcal{B},\ \exists\ e \leftarrow E \in \Phi \quad E \subseteq A\} \qquad (5)$$

and we have $\mathsf{Ind}(\Phi) = \mathsf{Ind}(T_\Phi)$ and $\mathsf{CoInd}(\Phi) = \mathsf{CoInd}(T_\Phi)$. The following result is a special case of one of Tarski's theorems.

Theorem 1 *If φ is a monotone operator, then $\mathsf{Ind}(\varphi)$ is the least fixpoint of φ ($\mathsf{lfp}(\varphi)$) and $\mathsf{CoInd}(\varphi)$ is the greatest fixpoint of φ ($\mathsf{gfp}(\varphi)$).*

It is possible to iterate towards these fixpoints as follows. For every ordinal α, we define ordinal powers of φ by: $\varphi^{\uparrow 0} = \emptyset$ and $\varphi^{\downarrow 0} = \mathcal{B}$, $\varphi^{\uparrow \beta+1} = \varphi(\varphi^{\uparrow \beta})$ and $\varphi^{\downarrow \beta+1} = \varphi(\varphi^{\downarrow \beta})$, if α is a limit ordinal, then $\varphi^{\uparrow \alpha} = \cup_{\beta < \alpha} \varphi^{\uparrow \beta}$ and $\varphi^{\downarrow \alpha} =$

$\cap_{\beta<\alpha}\varphi^{\downarrow\beta}$. The operator φ is said to be $\uparrow$-*continuous* (resp. $\downarrow$-*continuous*) if for every increasing (resp. decreasing) sequence $(E_n)_{n\geq 0}$ of subsets of $\mathcal{B}$, we have $\varphi(\cup_{n\geq 0}E_n) = \cup_{n\geq 0}\varphi(E_n)$ (resp. $\varphi(\cap_{n\geq 0}E_n) = \cap_{n\geq 0}\varphi(E_n)$).

Theorem 2 *If φ is a $\uparrow$-continuous (resp. $\downarrow$-continuous) operator, then* $\mathsf{Ind}(\varphi) = \mathsf{lfp}(\varphi) = \varphi^{\uparrow\omega}$ *(resp.* $\mathsf{CoInd}(\varphi) = \mathsf{gfp}(\varphi) = \varphi^{\downarrow\omega}$*).*

This theorem does not hold without the continuity assumption. However, if φ is *finitary* (i.e. if for every increasing sequence $(E_n)_{n\geq 0}$ of subsets of $\mathcal{B}$, we have $\varphi(\cup_{n\geq 0}E_n) \subseteq \cup_{n\geq 0}\varphi(E_n)$) and monotone, then φ is $\uparrow$-continuous, so $\mathsf{lfp}(\varphi) = \varphi^{\uparrow\overline{\omega}}$. If φ is the operator obtained from a rule set Φ, as described by (5), then φ is finitary if the set of premises of each rule of Φ is finite (in this case, we also say that Φ is finitary).

2.2 Standard concepts of logic programming

2.2.1 Herbrand semantics

In the following sections, we assume familiarity with the standard notions of logic programming as introduced in [24]. Σ, Π and X denote respectively a set of function symbols, a set of predicate symbols, and a set of variable symbols. Elements of $T_\Sigma[X]$ are *terms* over $\Sigma \cup X$. A *substitution* is a mapping from X to $T_\Sigma[X]$ such that $\{x, x \neq \theta x\} = dom(\theta)$ is finite. $range(\theta)$ denotes the set $\{var(\theta x), x \in dom(\theta)\}$. We write $\theta_{\restriction var(E)}$ for the restriction of θ to $var(E)$. 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 $r\colon X \to X$ such that $\forall x, y \in dom(r)$, $x \neq y \Rightarrow r(x) \neq r(y)$. 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 $\theta_2 E_2 = E_1$. $At_{\Sigma,\Pi}[X]$ denotes the set of atoms. Given a clause C, we write C^+ its head and C^- its body. An SLD-*derivation* from R_0 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 $B_1, \cdots, B_q$. The *renaming process* required in an SLD-derivation:

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

is such that for all $i \geq 1$, $var(C_i) \cap (\cup_{j<i} var(C_j) \cup var(R_0)) = \emptyset$. 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. The model-theoretic semantics of logic programs is based on *Herbrand interpretations* (subsets of the Herbrand base). From this point of view, the meaning of a program P is defined as the *least Herbrand model* of P (i.e. ground atoms which are logical consequences of P). This set coincides with the *ground success set* of P (ground atoms A from which there exists an SLD-refutation). This correspondence is proved by using fixpoint results of the operator T_P over Herbrand interpretations, associated with P and defined by:

$$T_P(I) = \{A \in At_{\Sigma,\Pi}[\emptyset], \quad \exists A' \leftarrow A_1, \cdots, A_n \in P \; \exists \theta : X \rightarrow T_\Sigma[\emptyset] \\ \theta A' = A \text{ and } \theta A_i \in I \quad (1 \leq i \leq n)\}$$

In this paper, we use the following notations, coming from [10]:

$$E \subseteq At_{\Sigma,\Pi}[X] \qquad \begin{array}{rcl} \lceil E \rceil & = & \{\theta A \in At_{\Sigma,\Pi}[X], A \in E\} \\ \lfloor E \rfloor & = & \{\theta A \in At_{\Sigma,\Pi}[\emptyset], \; A \in E\} \end{array}$$

2.2.2 $\mathcal{C}$-semantics

The standard semantics of logic programs (*à la* Herbrand), based on the ground success set, is not completely adequate as operational semantics since no variable occurs in this semantics. The $\mathcal{C}$-semantics has been revisited in details by M. Falaschi, G. Levi, C. Palamidessi and M. Martelli [10, 11] and allows variables in the elements of the domain. The Herbrand universe considered, written $T_\Sigma[X]_{/\approx}$, is the quotient set of $T_\Sigma[X]$ with respect to the variance equivalence relation $\approx$. For the sake of simplicity, the elements of $T_\Sigma[X]_{/\approx}$ will have the same representation as the elements of $T_\Sigma[X]$ (the intended meaning of $f(x) \in T_\Sigma[X]_{/\approx}$ is the equivalence class of $f(x)$ belongs to $T_\Sigma[X]_{/\approx}$). It is well-known that the preorder $\leq$ on $T_\Sigma[X]$ induces an order relation, still denoted by $\leq$, on $T_\Sigma[X]_{/\approx}$. The Herbrand base considered is the quotient set $At_{\Sigma,\Pi}[X]_{/\approx}$ which can be ordered by $p(\vec{t}_1) \leq p(\vec{t}_2) \Leftrightarrow \vec{t}_1 \leq \vec{t}_2$. Interpretations are subsets of $At_{\Sigma,\Pi}[X]_{/\approx}$ and the notion of truth coincides with the one of being a member of. In order to avoid the situation where an atom A is true with respect to an interpretation I which does not contain instances of A, we require interpretations to be $\uparrow$-closed (i.e. $(A \in I \wedge A \leq B) \Rightarrow B \in I$): $\uparrow$-closed subsets of $At_{\Sigma,\Pi}[X]_{/\approx}$ are called *$\mathcal{C}$-interpretations*. Note that given any subset I of $At_{\Sigma,\Pi}[X]_{/\approx}$, $\lceil I \rceil$ is a $\mathcal{C}$-interpretation (said another way, I is a $\mathcal{C}$-interpretation iff $I = \lceil I \rceil$). Given a $\mathcal{C}$-interpretation I, *$\mathcal{C}$-truth* is defined as follows:

- an atom A is $\mathcal{C}$-true in I iff $A \in I$ (i.e. the equivalence class of $A \in I$).
- 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 definite program P if every clause in P is $\mathcal{C}$-true in I. $\mathcal{C}$-models and (standard) Herbrand models can be related: if I is a $\mathcal{C}$-model of a definite program P, then $[I]$ is a (standard) Herbrand model of P. Note also that the class of $\mathcal{C}$-interpretations is a complete lattice with respect to set inclusion (if E is a set of $\mathcal{C}$-interpretations, then $\mathsf{glb}(E) = \cap_{I \in E} I$ and $\mathsf{lub}(E) = \cup_{I \in E} I$). 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 a program. Operational semantics, defined by:

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

is related to $\mathcal{M}_P^{\mathcal{C}}$ by considering the least fixpoint of the $\uparrow$-continuous operator defined by:

$$T_P^{\mathcal{C}}(I) = \{A \in At_{\Sigma,\Pi}[X], \quad \exists A' \leftarrow A_1, \cdots, A_n \in P\ \exists \theta\colon X \rightarrow T_\Sigma[X] \\ \theta A' = A \text{ and } \theta A_i \in I \quad (1 \leq i \leq n)\}$$

satisfying standard properties: a $\mathcal{C}$-interpretation I is a $\mathcal{C}$-model of a program P iff $T_P^{\mathcal{C}}(I) \subseteq I$ and $\mathcal{M}_P^{\mathcal{C}} = \mathsf{lfp}(T_P^{\mathcal{C}}) = \mathsf{lnd}(T_P^{\mathcal{C}}) = {T_P^{\mathcal{C}}}^{\uparrow\omega} = S_P^{\mathcal{C}}$.

Theorem 3 (Soundness and completeness [11])

1. *If there exists an SLD-refutation $R = A_1, \cdots, A_q \stackrel{*,\theta}{\rightarrow}_P \Box$, then there exist a substitution θ' and $\{A'_1, \cdots, A'_q\} \subseteq \mathcal{M}_P^{\mathcal{C}}$ such that θ' is a mgu of $(A_1, \cdots, A_q)$ and $(A'_1, \cdots, A'_q)$ and $\theta'_{\restriction var(R)} = \theta_{\restriction var(R)}$.*
2. *Let R be the query $A_1, \cdots, A_q$. If there exist $\{A'_1, \cdots, A'_q\} \subseteq \mathcal{M}_P^{\mathcal{C}}$ and a mgu θ' of $(A_1, \cdots, A_q)$ and $(A'_1, \cdots, A'_q)$, then there exists an SLD-refutation $A_1, \cdots, A_q \stackrel{*,\theta}{\rightarrow}_P \Box$ such that $\theta'_{\restriction var(R)} \geq \theta_{\restriction var(R)}$.*

3 Objects computed at infinity

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 (for a survey, see [8]). However,

infinite behaviour of programs can be useful to model some situations and nonterminating "computations" have been considered for many programming paradigms: λ-calculus [22], rewrite systems [9], constraint logic programming [21], concurrent constraint programming [7] ... In this section, we review the main approaches to assign some meaning to infinite derivations in "pure" logic programming occurring in the literature [1, 2, 15, 19, 23, 24, 25]. All of them concentrate on the aspects related to the semantics of infinite objects and to the models for logic programs which take them into account. The universe of the discourse considered in these approaches contains infinite elements. There are mainly two reasons for this requirement:

1. They all argue that "*a natural requirement for modeling infinite derivations is the presence of infinite elements in the universe of the discourse*". This allows to consider derivations, like (2), which compute infinite objects at infinity. The sense of a "useful" infinite computation 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).

2. Most of them are based on a greatest fixed point characterisation of infinite objects computed by nonterminating derivations and in this case, they all try to obtain the identification $\mathsf{gfp}(T_P) = T_P^{\downarrow\omega}$. Generally, this property does not hold in the Herbrand base. However, programs satisfying this property, called canonical programs, have been studied by J. Jaffar and P.J. Stuckey [20]. For example, with the program:

$$P = \{p(0) \leftarrow q(x) \ ; \ q(S(x)) \leftarrow q(x)\} \tag{6}$$

we have:

$$T_P^{\downarrow\omega} = \bigcap_{n\geq 0} T_P^{\downarrow n} = \bigcap_{n\geq 0} \left(\bigcup_{i\geq n} \{q(S^i(0))\} \cup \{p(0)\} \right) = \{p(0)\}$$

while when a "good" completion of the Herbrand base is defined, we obtain:

$$\begin{aligned} T_P^{\downarrow\omega} &= \bigcap_{n\geq 0} T_P^{\downarrow n} = \bigcap_{n\geq 0} \left(\bigcup_{i\geq n} \{q(S^i(0))\} \cup \{p(0), q(S^\omega)\} \right) \\ &= \{p(0), q(S^\omega)\} = \mathsf{gfp}(T_P) \end{aligned}$$

and then T_P becomes $\downarrow$-continuous.

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 characterised by the presence of infinite elements viewed as the limits of infinite sequences of finite objects.

3.1 The 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 [24], is the metric one: the Herbrand base is made a complete metric space by introducing a distance between terms as follows:

$$d(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 = t_2 \\ 2^{-\inf\{n,\, \tau_n(t_1) \neq \tau_n(t_2)\}} & \text{otherwise} \end{cases}$$

where $\tau_n(t)$ denotes the truncation at height n of the tree t. 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 [6]). 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 operator T_P is both $\uparrow$-continuous and $\downarrow$-continuous and the main results coming from [1] are expressed as follows. Given a derivation:

$$R_0 \overset{C_1,\theta_1}{\rightarrow_P} R_1 \overset{C_2,\theta_2}{\rightarrow_P} \cdots \overset{C_i,\theta_i}{\rightarrow_P} R_i \overset{C_{i+1},\theta_{i+1}}{\rightarrow_P} \cdots$$

we write $d_{|i}$ for the derivation $R_0 \overset{*}{\rightarrow} R_i$ and $[\![d_{|i}(R_0)]\!]$ stands for ground instances, over the completed Herbrand base, of $d_{|i}(R_0) = \theta_i \cdots \theta_1 R_0$. Since $At_{\Sigma,\Pi}^\infty[\emptyset]$ is a complete metric space, $[\![d(R_0)]\!] = \cap_{i\in\mathbb{N}}[\![d_{|i}(R_0)]\!]$ is a non-empty set (while $\cap_{i\in\mathbb{N}}[d_{|i}(R_0)]$ can be empty) and we have the following results:

1. For every atom A, $[\![A]\!] \cap T_P^{\downarrow\omega} = \bigcup\{[\![d(A)]\!],\ d \text{ is fair}\}$.
2. $A \in At_{\Sigma,\Pi}^\infty[\emptyset]$ begins a successful derivation iff $A \in T_P^{\uparrow\omega}$.
3. $A \in At_{\Sigma,\Pi}^\infty[\emptyset]$ is the root of a finite and failed SLD-tree iff $A \notin T_P^{\downarrow\omega}$.
4. $A \in At_{\Sigma,\Pi}^\infty[\emptyset]$ begins a fair derivation iff $A \in T_P^{\downarrow\omega}$.

Note that assertion 4. does not correspond to a completeness result for logic programming since queries cannot contain infinite terms. In [24], 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 [24] is expressed as follows:

$$C_P \subseteq \mathsf{gfp}(T_P)$$

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 a typical example, if we consider the logic program (3), we have $p(f^\omega) \in \mathsf{gfp}(T_P)$ but $p(f^\omega)$ is not computable by an infinite derivation (i.e $p(f^\omega) \notin C_P$): the construction of the greatest fixpoint does not reflect how the infinite terms are constructed during a computation. However, the metric approach can be very useful when lattice-theoretic arguments cannot be used (for example with programs, containing negations, which are not stratified). In this case, M. Fitting suggests in [12] to find a metric with respect to which T_P is a contraction and then has a unique fixpoint[1].

3.2 Completion by ideals

Another approach, due to W.G. Golson [15], 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 leat 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 [15], ideals of $At_{\Sigma,\Pi}[X]$, called objects, are defined as the sets $\mathcal{A}\Theta = \{\mathcal{A}',\ \exists\sigma \in \Theta\ \mathcal{A}' \leq \sigma\mathcal{A}\}$ 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 $\{\alpha \in I,\ \forall\beta \in I\ \beta \subseteq \alpha \Rightarrow \alpha = \beta\}$. The operator T_P is defined by:

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

[1] Given a metric space (E, d), a mapping $f : E \to E$ is a contraction if for a k $(0 \leq k < 1)$ we have for all $x, y \in E$, $d(f(x), f(y)) \leq k.d(x, y)$. A contraction on a complete metric space has a unique fixpoint

and is shown $\downarrow$-continuous by considering only programs whose clauses are such that any variable in the body also appears in the head (for example, program (6) cannot be considered). The main result expressed in [15] is stated as follows: $\alpha \in \min(\mathsf{gfp}(T_P))$ iff there exists a fair derivation from $\mathcal{A}$ with mgu's $(\sigma_i)_i$ such that $\mathcal{A}\{\cup_i\{\sigma_i\}\} = \alpha$ where $\mathcal{A}$ is a collection of distinct rule head variants of P. For example, with program (3), we have $\min(\mathsf{gfp}(T_P)) = \{p(z)\}\{[]\}$ which is not an infinite object (derivation (4) does not compute an infinite term), while with the program:

$$P = \{p(f(x)) \leftarrow p(x)\} \tag{7}$$

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 infnite derivation:

$$p(z)\begin{bmatrix} z \\ f(x_1) \end{bmatrix}_{\longrightarrow} p(x_1)\begin{bmatrix} x_1 \\ f(x_2) \end{bmatrix}_{\longrightarrow} \cdots \begin{bmatrix} x_{i-1} \\ f(x_i) \end{bmatrix}_{\longrightarrow} p(x_i)\begin{bmatrix} x_i \\ f(x_{i+1}) \end{bmatrix}_{\longrightarrow} \cdots \tag{8}$$

However, infinite SLD-derivations are not completely characterised: T_P is shown $\downarrow$-continuous by considering a subclass of definite programs and furthermore, only a subclass of the finite and infinite elements constructible by nonterminating computations of a logic program (called "minimal" objects) are characterised 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:

$$\{p(x,y)\}\left\{\left[\begin{array}{c} x \\ f^i(x_i) \end{array}\right]\right\}_{i\geq 0}$$

while from $p(x, g(y))$, no minimal object is computed.

4 Induction, co-induction and logic programming

4.1 Logic programs as inductive definitions

The "orthodox" view of logic programming is based on the identification of a logic program with a first order theory: every clause in a definite program stands for a first order formula. Definite clauses enjoy a remarkable property: the *model intersection property* (if P is a definite program and $\{M_i\}_{i\in I}$ is

a non-empty set of Herbrand models of P, then $\cap_{i\in I} M_i$ is an Herbrand model of P). The usual model-theoretic semantics is given by the least Herbrand model, written $\mathcal{M}_P$, which is the intersection of all Herbrand models. From the operational point of view, this set coincides with the ground success set of P. This correspondence is proved by considering the least fixpoint of the operator T_P, associated with the program P, also called the fixed point semantics. This *fixed point semantics* is an alternative to the traditional paradigm and can be obtained by considering logic programs as inductive definitions of sets and relations: a definite program defines a new "logic" (i.e. a formal system) and denotes a set of theorems in this logic. 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 $A_1, \cdots, A_n$. From this point of view, this clause c is a function mapping a n-uple of proofs π_{A_i} of A_i to a proof π_A of A. We write $\pi_A{:}\, A$ to express that π_A is a proof of A and we say the type of π_A is A. This correspondence between, proofs and functions, and, propositions and types, is now well-known and is based on the Curry-Howard isomorphism [14]. As we said, the traditional semantics of logic programs is defined in terms of least Herbrand model and ground success set. Within the "logic program as inductive definition" paradigm, the same semantics can be expressed by considering a program P as a schematic rule which abbreviates an infinite set of rules $[P]$: all ground instances over the Herbrand universe. By following this approach, clauses should not be viewed as assertions in first order logic, but as rules generating a set. The fixed point semantics has long been used as a technical device. It corresponds to the "logic program as inductive definition" paradigm and can be considered as the logic program's intrinsic declarative content. Indeed, many properties of logic programs are similar to these enjoyed by inductive definitions. Recall that an Herbrand interpretation I is a model of P iff $T_P(I) \subseteq I$ and the model intersection property, allows to consider the least Herbrand model of P as the intersection of all Herbrand models of P. Since T_P is exactly the operator $T_{[P]}$ obtained from the rule set $[P]$, as described by (5), each Herbrand model of P is a $T_{[P]}$-closed set (i.e. a $[P]$-closed set) and, since $\mathsf{Ind}(T_{[P]})$ is defined as the intersection of all $T_{[P]}$-closed sets, we have $\mathcal{M}_P = \mathsf{Ind}([P])$. By theorem 1, it follows $\mathcal{M}_P = \mathsf{Ind}([P]) = \mathsf{lfp}(T_{[P]})$. Now, since the body of each definite clause contains a finite number of atoms, the rule set $[P]$ is finitary and therefore $T_{[P]}$ is $\uparrow$-continuous. Hence by theorem 2, we obtain the well-known result $\mathcal{M}_P = \cap_{T_{[P]}(I)\subseteq I} I = \mathsf{lfp}(T_{[P]}) = T_{[P]}^{\uparrow\omega}$ which only follows from properties of inductive definitions: the least Herbrand model can be directly expressed by an inductive definition. In a similar way, the least $\mathcal{C}$-model can

be also directly expressed by an inductive definition based on the rule set:

$$\lceil P \rceil = \{\theta C,\ C \in P,\ \theta : X \to T_\Sigma[X]\}$$

Therefore, $T_P^{\mathcal{C}}$ is the operator $T_{\lceil P \rceil}$ associated, as described by (5), with $\lceil P \rceil$.

Since grammars are inductive definitions, this approach can explain why logic programming works so well at natural language processing. Inductive definitions are also useful to give a semantics to negation in logic programming: when the program can be partitioned into several inductive definitions, so that each negation refers to a set that has already been defined (i.e. when the dependency graph is acyclic), it can be interpreted as an iterated inductive definition. This "logic program as inductive definition" paradigm has also be used to extend logic programming languages in order to increase the power of "pure" declarative programming [16, 17, 18, 26].

4.2 Logic programs as co-inductive definitions

4.2.1 Atoms computed at infinity and greatest fixpoints

As we said in section 3, the main approaches to assign some meaning to infinite SLD-derivations are based on a greatest fixed point characterisation of infinite objects computed by nonterminating derivations. For programs, like program (7), these approaches are sound and complete since we have $p(f^\omega) \in \mathsf{gfp}(T_P)$ and $p(f^\omega)$ is computable by the infinite derivation (8). This may be explained by considering the "logic program as co-inductive definition" paradigm: the greatest fixpoint of the operator T_P over the completed Herbrand base corresponds to the co-inductive set $\mathsf{CoInd}(T_{[\![P]\!]})$, where $[\![P]\!]$ denotes all the ground instances of clauses occurring in P over the completed Herbrand base, and then we have $p(f^\omega) \in \mathsf{CoInd}([\![P]\!])$ since the clause $p(f^\omega) \leftarrow p(f^\omega)$ is in $[\![P]\!]$ and therefore $\{p(f^\omega)\}$ is $[\![P]\!]$-dense (i.e. $T_{[\![P]\!]}$-dense) because $f(f^\omega) = f^\omega$. However, these approaches do not lead to a complete semantics: there exist atoms in $\mathsf{gfp}(T_P)$ which are not computable by an infinite derivation. As a typical example, if we consider the logic program (3), $p(f^\omega)$ is not computable by an infinite derivation but we have $p(f^\omega) \in \mathsf{gfp}(T_P)$ by the same density argument. The incompleteness comes from the fact that clauses of $[\![P]\!]$ are expressed over a language richer than the language of clauses of P and the language of queries. However, by allowing infinite elements in queries and programs, the metric approach becomes complete: for example, with program (3) we have the following infinite derivation from the query $p(f^\omega)$:

$$p(f^\omega) \to p(f^\omega) \to \cdots p(f^\omega) \to \cdots \tag{9}$$

which will be viewed as a proof of $p(f^\omega)$.

4.2.2 Infinite SLD-derivations as productive terms

Logic programs express properties on terms which can be proved through SLD-derivations. Within the "logic programs as co-inductive definitions" paradigm, it is also possible to establish these properties by co-induction. In this section, we compare proofs by co-induction according to the use of infinite terms or not. *Co-induction* is a proof principle based on the following remark:

> (T. Coquand [5]) *In order to establish that a proposition ϕ follows from other propositions $\phi_1, \cdots, \phi_q$, it is enough to build a proof term e for it, using not only natural deduction, case analysis, and already proven lemmas, but also using the proposition we want to prove recursively, provided such a recursive call is guarded by introduction rules.*

Let us first introduce some examples. Sequences of positive integers can be defined with the two following introduction rules:

$$(\mathsf{nil}) : \frac{}{\mathsf{nil}\colon L_{\mathbb{N}}} \qquad (\mathsf{cons}) : \frac{n\colon \mathbb{N} \quad l\colon L_{\mathbb{N}}}{\mathsf{cons}(n,l)\colon L_{\mathbb{N}}}$$

In the case of an inductive definition, these two rules are associated with the following elimination scheme:

$$(L_{\mathbb{N}}\text{-}\mathsf{Ind}) : \frac{\begin{array}{rcl} P & : & L_{\mathbb{N}} \to \mathsf{Prop} \\ l & : & L_{\mathbb{N}} \\ \pi_1 & : & P(\mathsf{nil}) \\ \pi_2 & : & (n_0\colon \mathbb{N})(l_0\colon L_{\mathbb{N}})P(l_0) \to P(\mathsf{cons}(n_0,l_0)) \end{array}}{\left(\begin{array}{l} \mathsf{Fix}\ F(l') : P(l') := \\ \mathsf{match}\ l'\ \mathsf{with} \\ \qquad\qquad \mathsf{nil} \ \to \pi_1 \quad | \\ \mathsf{cons}(n_1,l_1) \ \to \pi_2(n_1,l_1,F(l_1)). \end{array}\right)(l) : P(l)}$$

and elements of $L_{\mathbb{N}}$ are the result of a finite number of applications of these two rules. A co-inductive definition is obtained by relaxing this condition and admitting that an element can also be introduced by a non-ending process of construction defined by these rules. Given such a definition, we have the

following elimination scheme:

$$(L_{\mathbb{N}}\text{-Coind}) : \frac{\begin{array}{rcl} P & : & L_{\mathbb{N}} \to \mathsf{Prop} \\ l & : & L_{\mathbb{N}} \\ \pi_1 & : & P(\mathsf{nil}) \\ \pi_2 & : & (n_0{:}\,\mathbb{N})(l_0{:}\,L_{\mathbb{N}})P(\mathsf{cons}(n_0, l_0)) \end{array}}{\mathsf{Case}\ l\ \mathsf{of}\ \ \pi_1\ \pi_2\ \mathsf{end} : P(l)}$$

associated with the two reduction rules:

$$\begin{array}{llll} \mathsf{Case} & \mathsf{nil} & \mathsf{of}\ \pi_1\ \pi_2\ \mathsf{end} \to & \pi_1 \\ \mathsf{Case} & \mathsf{cons}(n, l) & \mathsf{of}\ \pi_1\ \pi_2\ \mathsf{end} \to & \pi_2(n, l) \end{array}$$

Now, it is possible to define several infinite sequences of integers in a recursive way. For example, the (infinite) sequence of consecutive integers starting from n can be defined by $\mathsf{from}(n)$ where:

$$\mathsf{from} := \lambda n{:}\,\mathbb{N}.\mathsf{cons}(n, \mathsf{from}(S(n))) : \mathbb{N} \to L_{\mathbb{N}}$$

However, every recursive definition is not valid: any finite segment of the sequence must be explicitly constructed using the two rules (for more details, see [13]). For example, we have :

$$\mathsf{from}(n) \to \mathsf{cons}(n,\mathsf{from}(S(n))) \to \mathsf{cons}(n,\mathsf{cons}(S(n),\mathsf{from}(S^2(n)))) \to \cdots$$

The definition of from is correct because it starts building the object providing an explicit finite initial segment. Now, if we consider the following definition:

$$\mathsf{zeros} := \mathsf{cons}(0,\mathsf{tail}(\mathsf{zeros})) : L_{\mathbb{N}}$$

where tail is defined by:

$$\mathsf{tail}(l) := \mathsf{match}\ l\ \mathsf{with}\ \mathsf{nil} \to \mathsf{nil} \mid \mathsf{cons}(n_0, l_0) \to l_0.$$

then we have the following sequence of reductions:

$$\mathsf{zeros} \to \mathsf{cons}(0,\mathsf{tail}(\mathsf{zeros})) \to \mathsf{cons}(0,\mathsf{tail}(\mathsf{zeros})) \to \cdots$$

Hence, any finite segment of length greater than 1 cannot be obtained by reductions. from is said to be a productive term, while zeros is not a productive term.

In a more formal way, productive terms are defined as follows. Recall that a term t in T is in *canonical form* iff it starts with a constructor of T. A (direct) *component* of a term t in T is a term t' in T if t can be

reduced to a canonical term $c(\cdots, t', \cdots)$. A term t is *productive* iff it can be reduced to a canonical term and if all its components are productive. For example, a term in L_{IN} is productive iff it can be reduced either to nil or $\mathsf{cons}(a, b)$ where b is productive. Like for terminaison of recursive functions on inductive sets, there exists a simple syntactical criteria for productive terms over co-inductive sets: *guardedness*. This class of definitions, called guarded (by constructors) definitions, has been noticed, in the context of type theory, by T. Coquand [5]. A *guarded by constructors* definition is a definition such that all the recursive calls of the definition are done after having explicitly gived which is (at least) the first rule to start building the element and such that no other functions apart from constructors are applied to recursive calls. However, note that this condition is too restrictive: there exist productive terms which do not satisfy this condition.

In the following, we consider equality as an inductive relation. That is, given a set A and an x in A, the set $\{z,\ z = x\}$ is the smallest which contains x. This definition, due to C. Paulin-Mohring, is equivalent to define $=$ as the smallest reflexive relation:

$$(=) : \frac{}{x = x}$$

Leibniz'equality is obtained as the elimination scheme associated with this definition:

$$(\mathsf{eq\_ind}) : \frac{\begin{array}{rcl} x & : & A \\ P & : & A \to \mathsf{Prop} \\ \pi_1 & : & P(x) \\ y & : & A \\ \pi_2 & : & x = y \end{array}}{P(y)}$$

A fair SLD-derivation, like (2), which computes at least one infinite term t is both a computation (of the infinite term t) and a proof that this infinite term is such that $p(\cdots, t, \cdots)$ for a predicate p. In this case, it can be noticed that the proof term corresponding to the proof of $p(\cdots, t, \cdots)$ is defined by using the elimination scheme $\mathsf{eq\_ind}$. For example, with program (7), the proof term of $p(f^\omega)$ is defined by:

$$\pi := \mathsf{eq\_ind}(f(f^\omega), \lambda x.p(x), C(f^\omega, \pi), f^\omega, \ell_\omega) : p(f^\omega) \tag{10}$$

where the recursive call is guarded by the constructor C where C is the clause of P and where ℓ_ω is a proof of $f^\omega = f(f^\omega)$[2], and does not correspond

[2]Note that f^ω is defined by a guarded by constructors definition ($f^\omega := f(f^\omega)$) since possibly infinite terms are co-inductively defined with function symbols as constructors.

to the infinite derivation (8) computing f^ω. However, proof term (10) can be viewed as the derivation (9) we could obtain from the query $p(f^\omega)$. That's why by allowing infinite terms in queries, a complete semantics for all infinite derivations could be obtained. Another example can be shown by considering the logic program (1), where the clause C corresponds to a co-inductive definition. We can prove that $\forall n\ \mathsf{LN}(n, \mathsf{from}(n))$. For this we need the following property:

$$\ell_\omega : \forall n\ \mathsf{from}(n) = \mathsf{cons}(n, \mathsf{from}(S(n)))$$

The proof term π of $\forall n\ \mathsf{LN}(n, \mathsf{from}(n))$ can be defined by the following guarded by constructors definition:

$$\begin{array}{lll} \pi := \lambda n. & \mathsf{eq\_ind}(& \mathsf{cons}(n, \mathsf{from}(S(n))), \\ & & \lambda u.\mathsf{LN}(n, u), \\ & & C(n, \mathsf{from}(S(n)), \pi(S(n))), \\ & & \mathsf{from}(n), \\ & & \ell_\omega(n)) \end{array}$$

Clearly, this proof does not correspond to the infinite derivation (2).

Let us consider now an infinite proof over a finite object: with program (3), it is possible to prove that $\forall x\ p(x)$. The corresponding proof π is defined by the guarded by constructors term $\pi{:=}\lambda x.C(x, \pi(x))$ and can be directly related to the infinite derivation (4). The application of the constructor C corresponds to the first transition of this derivation, while the recursive call corresponds to the next ones. In a similar way, with the program $P = \{p(x) \leftarrow p(f(x))\}$, we can prove $\forall x p(x)$ as follows:

$$\pi{:=}\lambda z.C\left(z, \pi\left(\left[\begin{array}{c} z \\ f(z) \end{array}\right] p(z)\right)\right)$$

and clearly π corresponds to the derivation:

$$p(z) \to \left[\begin{array}{c} z \\ f(z) \end{array}\right] p(z) \to \left[\begin{array}{c} z \\ f(z) \end{array}\right] \left[\begin{array}{c} z \\ f(z) \end{array}\right] p(z) \to \cdots$$

Here again, the application of the constructor C corresponds to the first transition of this derivation, while the recursive call corresponds to the next ones (i.e. the derivation starting from the query $p(f(z))$). A more significant example is the program testing connectivity in a directed graph. Since the directed graph considered by the program (see figure 1) is cyclic, there exists an infinite derivation from the query $\mathsf{path}(a, x)$:

$$\begin{array}{ll} \mathsf{path}(a, x) & \to \mathsf{edge}(a, b), \mathsf{path}(b, x) \to \mathsf{path}(b, x) \to \mathsf{edge}(b, c), \mathsf{path}(c, x) \\ & \to \mathsf{path}(c, x) \to \mathsf{edge}(c, a), \mathsf{path}(a, x) \to \mathsf{path}(a, x) \to \cdots \end{array}$$

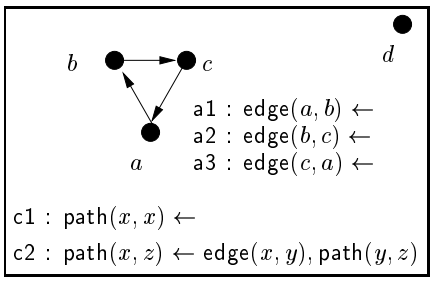

Figure 1: Connectivity in a directed graph

This derivation can be viewed as a proof of $\forall x\, \mathsf{path}(a, x)$ which can be defined recursively:

$$\pi := \lambda x.\mathsf{c2}(a, b, x, \mathsf{a1}, \mathsf{c2}(b, c, x, \mathsf{a2}, \mathsf{c2}(c, a, x, \mathsf{a3}, \pi(x))))$$

Of course, with the following version of the predicate path:

$$\mathsf{path}(0, x, x) \leftarrow$$
$$\mathsf{path}(S(n), x, z) \leftarrow \mathsf{edge}(x, y), \mathsf{path}(n, y, z)$$

we have $\mathsf{path}(S^\omega, x, d)$ for $x \in \{a, b, c\}$ since the length of this "path" is infinite (as the length of the proof) and $\mathsf{Ind}[\mathsf{path}]$ characterises finite paths in the graph.

Proof terms over finite objects do not use eq_ind and can be viewed as definitions of the sequences of clauses used in the corresponding derivations. Furthermore, since a clause is applied at each resolution step of a derivation, the associated proof terms should be productive. However, it is not always possible to associate a productive proof term with an infinite SLD-derivation. Consider, for example, the following program:

$$P = \{\underbrace{p(x) \leftarrow p(f(x))}_{C_1}\ ;\ \underbrace{p(x) \leftarrow p(g(x))}_{C_2}\} \tag{11}$$

At each resolution step in a derivation from the query $p(z)$, we can apply both C_1 and C_2 and if the following function:

$$\mathcal{F}_C : \mathbb{N} \to \{C_1, C_2\}$$

defining the clause used at the n-th transition, is not "computable", then the proof term associated with the derivation cannot be written in a finite form,

since this term is defined by:

$$\begin{array}{ll} \pi{:=}\lambda x.\pi_d(0,x) : \Pi_x p(x) & \text{with} \\ \pi_d{:=}\lambda n.\lambda x.(\mathcal{F}_C(n))(x,\pi_d(S(n),(\mathcal{F}_F(n))(x))) & \text{and} \\ \mathcal{F}_F : \mathbb{N} \to \{f,g\}{:=}\lambda n.(\mathsf{if}\ \mathcal{F}_C(n) = C_1\ \mathsf{then}\ f\ \mathsf{else}\ g) & \end{array}$$

This means that the proof term associated with a derivation can be viewed as the definition of the infinite sequence of the clauses applied during the derivation: given a productive proof term, it is possible to define the sequence of the clauses used during the associated derivation. For example, clauses in program (11) stands for the following introduction rules:

$$(C_1) : \frac{\begin{array}{rcl} x & : & T_\Sigma[X] \\ \pi & : & p(f(x)) \end{array}}{C_1(x,\pi(x)) : p(x)} \qquad (C_2) : \frac{\begin{array}{rcl} x & : & T_\Sigma[X] \\ \pi & : & p(g(x)) \end{array}}{C_2(x,\pi(x)) : p(x)}$$

associated with the following co-inductive elimination scheme:

$$(\mathsf{Elimination}) : \frac{\begin{array}{rcl} x & : & T_\Sigma[X] \\ \mathcal{P} & : & p(x) \to s \\ \pi & : & p(x) \\ \pi_1 & : & (x : T_\Sigma[X])(\pi_f : p(f(x)))\mathcal{P}(C_1(x,\pi_f)) \\ \pi_2 & : & (x : T_\Sigma[X])(\pi_g : p(g(x)))\mathcal{P}(C_2(x,\pi_g)) \end{array}}{\mathsf{Case}\ \pi\ \mathsf{of}\ \pi_1\ \pi_2\ \mathsf{end} : \mathcal{P}(\pi)}$$

Reduction rules are:

$$\begin{array}{rcl} \mathsf{Case}\ C_1(x,\pi')\ \mathsf{of}\ \pi_1\ \pi_2\ \mathsf{end} & \to & \pi_1(x,\pi') \\ \mathsf{Case}\ C_2(x,\pi')\ \mathsf{of}\ \pi_1\ \pi_2\ \mathsf{end} & \to & \pi_2(x,\pi') \end{array}$$

and allows to define $\mathcal{F}_P : \forall x p(x) \to \mathbb{N} \to \{C_1, C_2\}$ as follows:

$$\begin{array}{rl} \mathcal{F}_P(\pi,n){:=} & \mathsf{match}\ \pi\ \mathsf{with} \\ & C_1(x,\pi') \to (\mathsf{match}\ n\ \mathsf{with}\ 0 \to C_1 \,|\, S(k) \to \mathcal{F}_P(\pi',k))| \\ & C_2(x,\pi') \to (\mathsf{match}\ n\ \mathsf{with}\ 0 \to C_2 \,|\, S(k) \to \mathcal{F}_P(\pi',k)). \end{array}$$

Hence, we have $\mathcal{F}_C = \mathcal{F}_P(\pi)$.

Since the presence of infinite elements in the Herbrand base leads to incompleteness of the approaches based on the greatest fixpoint, we focus in the following on the infinite derivations which do not compute infinite terms.

5 Infinite SLD-proofs

Examples presented in section 4.2.2 show that infinite derivations which do not compute infinite terms can be related to proof terms over co-inductive sets. Therefore, we investigate in this section this class of 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.

Definition 1 (SLD-proofs) *An SLD-proof is either an SLD-refutation or a fair infinite SLD-derivation.*

5.1 Proof trees and fair derivations

In order to prove the completeness of our approach, we prove in this section that, given a rule set Φ, there exists an SLD-proof with Φ as program, which do not compute anything, from each element in $\mathsf{Coind}(\Phi)$. For this, we introduce the classical notion of proof trees and we relate this notion to SLD-derivations.

Definition 2 (Proof trees) *Given a rule set Φ, a proof tree of x 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 sons, 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 1 *$x \in \mathsf{Coind}(\Phi)$ iff x is the root of a proof tree for Φ.*

Furthermore, the proof tree is finite iff $x \in \mathsf{Ind}(\Phi)$. In order to be able to "translate" any proof tree into an SLD-derivation, we need two lemmas expressing properties about variables renaming. Their proofs are quite technical and are presented in appendix. However, 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 avoid some confusions usually due to the fact that the meaning of "renaming" is often assumed to be simpler than its formal definition implies. Furthermore, we prove the following lemma.

Lemma 2 *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.*

We are now in position to prove the main result of this section relating proof trees with SLD-proofs introduced in definition 1.

Theorem 4 *Given a definite program P and an atom A, if $A \in \mathsf{Colnd}(P)$, then there exists an SLD-proof from A with P such that, for all $i \geq 1$, the mgu θ_i, used during the i-th resolution step of the SLD-proof, is a renaming substitution whose domain coincides with the variables occurring in the head of the clause used.*

Proof. If $A \in \mathsf{Colnd}(P)$, then, by lemma 1, A is root of a proof tree T for P. 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 list a $\mathcal{L}$. Since T is a proof tree for P, for each node $A_{\vec{\imath}}$ in T, there exists a clause $C_{T,\vec{\imath}} \in P$ 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:

$$\begin{array}{rcl} \vec{\imath} \prec \vec{\jmath} & \Leftrightarrow & A_{\vec{\imath}} \text{ occurs before } A_{\vec{\jmath}} \text{ in } \mathcal{L} \\ & \Leftrightarrow & ((|\vec{\imath}| < |\vec{\jmath}|) \vee (|\vec{\imath}| = |\vec{\jmath}| \wedge \vec{\imath} \prec_\ell \vec{\jmath})) \end{array}$$

$Z_T = \cup var(C_{T,\vec{\imath}})$ is the set, possibly infinite, of variables occurring in T. By lemma 8, given a clause $C_{T,\vec{\imath}} \in P$, 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 $r_0^{\vec{\imath}} C^+_{T,\vec{\imath}}$. In the following, we write $\mathcal{T}(C_{T,\vec{\imath}}, r_0^{\vec{\imath}}, Z_{\vec{\imath}})$ for the transition:

$$r_0^{\vec{\imath}} C^+_{T,\vec{\imath}} \overset{C_{\vec{\imath}}, \theta_{\vec{\imath}}, Z_{\vec{\imath}}}{\rightarrow}_P \theta_{\vec{\imath}} C^-_{\vec{\imath}}$$

From $\mathcal{L}$, we can define the following sequence of resolution steps:

$$\begin{cases} t_\varepsilon = \mathcal{T}(C_{T,\varepsilon}, s_{id}, Z_T) & \\ & 1 \leq k \leq n_{\vec{\imath}} \\ t_{\vec{\imath}k} = \mathcal{T}\left(C_{T,\vec{\imath}k}, r_0^{\vec{\imath}k}, Z_{\vec{\imath}k}\right) & r_0^{\vec{\imath}k} = r_1^{\vec{\imath}} \\ & Z_{\vec{\imath}k} = Z_T \cup \bigcup_{\vec{\jmath} \prec \vec{\imath}k} var(C_{\vec{\jmath}}) \end{cases}$$

where s_{id} is the empty substitution. In order to verify the soundness of this definition, we have to prove that:

$$\forall \vec{\imath} \quad range(r_0^{\vec{\imath}}) \cap var(C_{T,\vec{\imath}}) = \emptyset$$

For this, let us prove that $\forall \vec{\imath} \quad range(r_0^{\vec{\imath}}) \cap Z_T = \emptyset$. We proceed by induction over $\vec{\imath}$: suppose the property holds for every $\vec{\jmath} \prec \vec{\imath}$. If $|\vec{\imath}| = 0$, then the property holds since $range(s_{id}) \cap var(C_{T,\varepsilon}) = \emptyset$. Else, $\vec{\imath}$ can be written $\vec{\jmath}k$ and, by definition, we have $r_0^{\vec{\jmath}k} = r_1^{\vec{\jmath}} = r^{\vec{\jmath}} r_0^{\vec{\jmath}}$ where $r^{\vec{\jmath}}$ is a substitution such that $range(r^{\vec{\jmath}}) \cap Z_T = \emptyset$ since:

$$range(r^{\vec{\jmath}}) \subseteq var(C_{\vec{\jmath}}^-) \backslash var(C_{\vec{\jmath}}^+) \qquad Z_T \subseteq Z_{\vec{\jmath}} \qquad var(C_{\vec{\jmath}}) \cap Z_{\vec{\jmath}} = \emptyset$$

By induction hypothesis, we can conclude since $\vec{\jmath} \prec \vec{\jmath}k$ and:

$$range(r_0^{\vec{\jmath}k}) = range(r_1^{\vec{\jmath}}) \subseteq (range(r_0^{\vec{\jmath}}) \cup range(r^{\vec{\jmath}})) \qquad var(C_{T,\vec{\jmath}k}) \subseteq Z_T$$

Hence, we can obtain the following derivation:

$$A \overset{C_\varepsilon,\theta_\varepsilon,Z_T}{\rightarrow_P} R \overset{C_1,\theta_1,Z_1}{\rightarrow_P} R_1 \rightarrow \cdots \rightarrow R_{n-1} \overset{C_n,\theta_n,Z_n}{\rightarrow_P} R_n \overset{C_{11},\theta_{11},Z_{11}}{\rightarrow_P} R_{11} \rightarrow \cdots$$

Clearly, since the derivation is obtained from a breadth-first traversal of T, it is either a refutation or an infinite fair derivation. We prove in appendix that this SLD-proof is correct and satisfies the desired properties. ◀

In this section, proof trees for a rule set Φ have been related to SLD-proofs with Φ viewed as a program. We will see that the appropriate rule set allowing to study infinite derivations, which do not compute infinite 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.

5.2 Direct SLD-proofs

The derivation obtained by theorem 4 is a special case of a derivation which do not compute infinite terms: such a derivation does not compute anything since the mgu's used are just renaming substitutions. This particular class of derivation correspond to the (co-)inductive definition obtained by considering a subset of the rule set $\lceil P \rceil$. In this paragraph, we present the main results obtained for these derivations.

Definition 3 (Direct SLD-proofs) *A direct SLD-proof is an SLD-proof:*

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

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

In order to give a "model-theoretic" semantics to direct SLD-proofs, we extend the notion of $\mathcal{C}$-truth 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:

$$P = \{p(x) \leftarrow p(y) \ ; \ p(f(w)) \leftarrow\} \tag{12}$$

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 enjoy the model intersection property and are useful to define a declarative semantics for direct SLD-proofs.

The "logic program as inductive definition" paradigm is obtained by considering the rule set associated with P defined by:

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

which is associated, as described by (5), with the following operator:

$$T_{\lceil P \rceil^+}(I) = \{A \in At_{\Sigma,\Pi}[X], \quad \exists A \leftarrow A_1, \cdots, A_n \in \lceil P \rceil^+ \quad A_i \in I \quad (1 \leq i \leq n) \quad \}$$

This operator satisfies the following properties which are proved in appendix:

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

$\mathcal{C}^+$-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 (12), we have $\mathcal{M}_P^{\mathcal{C}^+} = \lceil p(f(x)) \rceil$ and $\mathcal{M}_P^{\mathcal{C}} = \lceil p(x) \rceil$. For direct SLD-refutations, we have standard results.

Theorem 5 ($\mathcal{C}^+$-soundness) *If there exists a direct SLD-refutation from a query $A_1, \cdots, A_q$, then $\{A_1, \cdots, A_q\} \subseteq \mathcal{M}_P^{\mathcal{C}^+}$.*

Proof. Induction over the refutation:

$$A_1, \cdots, A_q \overset{C_1,\theta_1}{\rightarrow}_P R_1 \rightarrow_P \cdots \rightarrow_P R_{i-1} \overset{C_i,\theta_i}{\rightarrow}_P R_i \rightarrow_P \cdots \rightarrow_P \square$$

- If the refutation is a transition, then $q = 1$ and C_1 is a unit clause $A \leftarrow$. By hypothesis, we have $A_1 = \theta_1 A$ and since $dom(\theta_1) \subseteq var(C_1^+)$ we can conclude seeing that $\mathcal{M}_P^{\mathcal{C}^+}$ is a $\mathcal{C}^+$-model of P.
- Consider the derivation $A_1, \cdots, A_q \overset{C_1,\theta_1}{\rightarrow}_P R_1 \overset{*,\sigma}{\rightarrow}_P \square$. If k ($1 \leq k \leq q$) is the position of the selected atom in the first transition, then R_1 is the query $\theta_1(A_1, \cdots, A_{k-1}, C_1^-, A_{k+1}, \cdots, A_q)$. Since $dom(\theta_1) \subseteq var(C_1^+)$, we have $R_1 = A_1, \cdots, A_{k-1}, \theta_1 C_1^-, A_{k+1}, \cdots, A_q$. By induction hypothesis, $R_1 \subseteq \mathcal{M}_P^{\mathcal{C}^+}$ and now it suffices to prove $A_k \in \mathcal{M}_P^{\mathcal{C}^+}$. Since $\mathcal{M}_P^{\mathcal{C}^+}$ is a $\mathcal{C}^+$-model of P and $\theta_1 C_1^- \subseteq \mathcal{M}_P^{\mathcal{C}^+}$, we have $\theta_1 C_1^+ = A_k \in \mathcal{M}_P^{\mathcal{C}^+}$. ◀

Theorem 6 ($\mathcal{C}^+$-completeness) *If $\{A_1, \cdots, A_q\} \subseteq \mathcal{M}_P^{\mathcal{C}^+}$, then there exists a direct SLD-refutation from the query $A_1, \cdots, A_q$.*

Proof. We first prove the theorem for $q = 1$. By lemma 12, $A_1 \in T_{\lceil P \rceil^+}^{\uparrow\omega}$ and there exists a natural k such that $A_1 \in T_{\lceil P \rceil^+}^{\uparrow k}$. We prove by induction that for all k, if $A \in T_{\lceil P \rceil^+}^{\uparrow k}$, then there exists a direct SLD-refutation from A.

- If $k = 0$, then $A \in T_{\lceil P \rceil^+}^{\uparrow 0} = \emptyset$ induces a contradiction.
- If $k = m + 1$, then $A \in T_{\lceil P \rceil^+}^{\uparrow k} = T_{\lceil P \rceil^+}(T_{\lceil P \rceil^+}^{\uparrow m})$ and there exist a clause C', written $A' \leftarrow B'_1, \cdots, B'_r$, and a substitution θ, whose domain is included in $var(A')$, such that $\theta A' = A$ and $\{\theta B'_1, \cdots, \theta B'_r\} \subseteq T_{\lceil P \rceil^+}^{\uparrow m}$. Furthermore,

by lemma 7, we can suppose that $var(C') \cap var(A) = \emptyset$. Therefore, by lemma 2, θ is a mgu of A and A' and we get the transition:

$$A \overset{C',\theta}{\rightarrow}_P \theta B'_1, \cdots, \theta B'_r$$

Now, since $\{\theta B'_1, \cdots, \theta B'_r\} \subseteq T^{\uparrow m}_{\lceil P \rceil^+}$, by induction hypothesis, there exist r direct SLD-refutations:

$$\theta B'_1 \rightarrow^*_P \square \quad \cdots \quad \theta B'_r \rightarrow^*_P \square$$

and, there exist r direct SLD-refutations:

$$d'_1 : \theta B'_1 \rightarrow^*_P \square \quad \cdots \quad d'_r : \theta B'_r \rightarrow^*_P \square$$

such that $\forall i \quad (1 \leq i \leq r) \quad \vartheta(d'_i) \cap \left(var(A) \cup var(C') \cup \bigcup_{1 \leq j < i} \vartheta(d'_j) \right) = \emptyset$

This allows to get the direct SLD-refutation:

$$A \overset{C',\theta}{\rightarrow}_P \theta B'_1, \cdots, \theta B'_r \rightarrow^*_P \square$$

For $q > 1$, the theorem is proved in the same way (combinaison of direct SLD-refutations). ◀

Therefore, atoms in $\mathcal{M}^{\mathcal{C}}_P \backslash \mathcal{M}^{\mathcal{C}^+}_P$ are atoms from which SLD-refutations, but no direct ones, exist. For example, with program (12), whereas $p(z) \in \mathcal{M}^{\mathcal{C}}_P$, $p(z) \notin \mathcal{M}^{\mathcal{C}^+}_P$. Even if there exists an SLD-refutation:

$$p(z) \begin{bmatrix} x_1 \\ z \end{bmatrix}_{\rightarrow_P} p(y_1) \begin{bmatrix} y_1 \\ f(w_1) \end{bmatrix}_{\rightarrow_P} \square \tag{13}$$

there is no direct SLD-refutation from $p(z)$. However, for the class of programs P such that $var(C^-) \subseteq var(C^+)$ for every clause of P, we have $\lceil P \rceil^+ = \lceil P \rceil$ (and then $\mathcal{M}^{\mathcal{C}}_P = \mathcal{M}^{\mathcal{C}^+}_P$). Another important property satisfied by these programs is $T^{\downarrow\omega}_{\lceil P \rceil} = \mathsf{gfp}(T_{\lceil P \rceil})$. For infinite direct SLD-proofs, we have the following results.

Theorem 7 ($\mathcal{C}^+$-soundness) *Let P be a definite program and A_0 be an atom. If there exists a direct SLD-proof:*

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

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

Proof. By definition, it suffices to prove that there exists a $T_{\lceil P \rceil^+}$-dense set containing A_0. Let us prove that $\cup_{i\geq 1}\theta_i C_i^+$ satisfies these two properties.

- By definition, $A_0 = \theta_1 A_0 = \theta_1 C_1^+ \subseteq \cup_{i\geq 1}\theta_i C_i^+$.
- We have to prove $\cup_{i\geq 1}\theta_i C_i^+ \subseteq T_{\lceil P \rceil^+}\left(\cup_{i\geq 1}\theta_i C_i^+\right)$. 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 $\theta_k C_k \in \lceil P \rceil^+$. 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 residu (i.e. the further instantiated version) of A_k is the selected atom:

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

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. ◀

Completeness theorem for infinite direct SLD-proofs is proved by using the following lemma.

Lemma 3 *If there exists a transition* $R_0 \overset{\mu C,\theta}{\rightarrow} R_1$ *such that:*

$$dom(\theta) = var(\mu C^+) \qquad var(C) \cap var(R_0) = \emptyset \qquad dom(\mu) = var(C^+)$$

then there exists a transition $R_0 \overset{C,\sigma}{\rightarrow} R_1$ *such that* $dom(\sigma) = var(C^+)$.

Proof. Let A be the selected atom in R_0 at position k. Since $dom(\theta) = var(\mu C^+)$, we have $A = \theta A = \theta\mu C^+$. Moreover, since $var(C)\cap var(R_0) = \emptyset$, and by lemma 2, the restriction σ of $\theta\mu$ to the variables occurring in C^+ is a mgu of A and C^+. Therefore, we get the transition:

$$R_0 \overset{C,\sigma}{\rightarrow} R_1'$$

Let us prove that $R_1' = R_1$. Since:

$$\begin{array}{l} R_1 = \theta R_0[k \leftarrow \mu C^-] = R_0[k \leftarrow \theta\mu C^-] \quad \text{and} \\ R_1' = \sigma R_0[k \leftarrow C^-] = R_0[k \leftarrow \sigma C^-] \end{array}$$

it suffices to prove $\theta\mu C^- = \sigma C^-$. If $v \in var(C^-)$, then two cases are possible:

1. If $v \in var(C^+)$, then, by definition of σ, we have $\theta\mu v = \sigma v$.
2. Else, $v \in var(C^-) \backslash var(C^+)$, and by hypothesis $v \notin dom(\sigma)$ and $v \notin dom(\mu)$. Therefore, $v \in var(\mu C^-) \backslash var(\mu C^+)$ and $v \notin dom(\theta)$. This terminates the proof since $\theta\mu v = \sigma v = v$. ◀

Theorem 8 ($\mathcal{C}^+$-completeness) *Let P be a definite program and A be an atom. If $A \in \mathsf{gfp}(T_{\lceil P \rceil^+})$, then there exists a direct SLD-proof from A with P.*

Proof. If $A \in \mathsf{gfp}(T_{\lceil P \rceil^+})$, then, by theorem 1, $A \in \mathsf{Colnd}(\lceil P \rceil^+)$ and by theorem 4, there exists a direct SLD-proof from A with $\lceil P \rceil^+$ such that for all $i \geq 1$, the mgu θ_i, used during the i-th resolution step of the SLD-proof, is a renaming substitution whose domain coincides with the variables occurring in the head of the clause used:

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

By lemma 14, there exists a set $\{C_{P,1}, \cdots, C_{P,i}, \cdots\}$ of variants of clauses of P such that:

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

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 3, there exists a direct SLD-proof from A with P. ◀

Unfair infinite "direct" SLD-derivations 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 $P \models R_i \Rightarrow P \models \theta_i \cdots \theta_1 R_0$. This result can be generalised for "direct derivations" by considering:

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

Theorem 9 *Let P be a program and A_0 be an atom. If there exists an infinite derivation:*

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

such that for all $i > 0$, $dom(\theta_i) \subseteq var(C_i^+)$, then:

$$R_\infty \subseteq \mathsf{gfp}(T_{\lceil P \rceil^+}) \Rightarrow A_0 \in \mathsf{gfp}(T_{\lceil P \rceil^+})$$

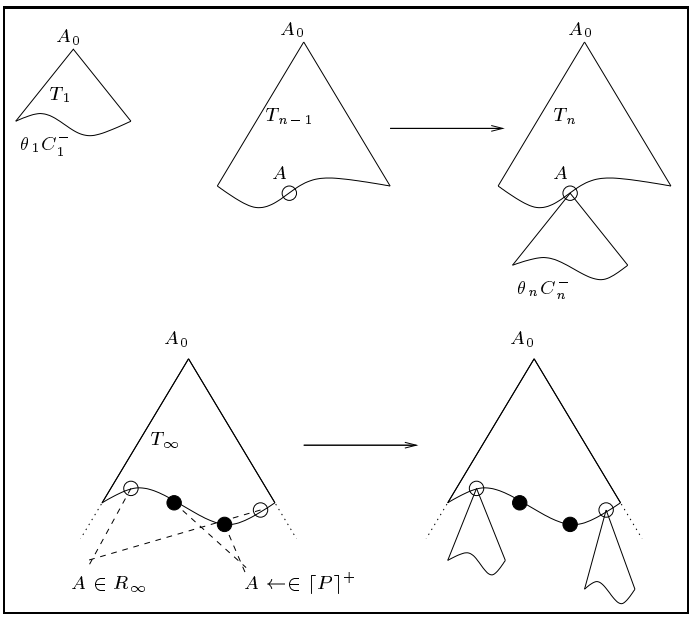

Figure 2: Proof of theorem 9

Proof. Suppose that $\cup_{p \geq 0} \cap_{p \leq n} R_n \subseteq \mathsf{gfp}(T_{\lceil P \rceil^+})$ and let us prove that $A_0 \in \mathsf{gfp}(T_{\lceil P \rceil^+})$. For this, by theorem 1 and by lemma 1, 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 (see fig. 2).

- (T_1) 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 sons.
- (T_n) We show 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 sons 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 $\mathsf{gfp}(T_{\lceil P \rceil^+})$ and correspond, by theorem 1 and by lemma 1, 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 $\lceil P \rceil^+$. ◀

This theorem is not a special case of theorem 8, it just gives another way to interpret infinite "direct derivations". For example, if we consider derivation (4), then by theorem 8 we have $p(x) \in \mathsf{gfp}(T_{\lceil P \rceil^+})$ while by theorem 9, we just have $p(x) \in \mathsf{gfp}(T_{\lceil P \rceil^+}) \Rightarrow p(x) \in \mathsf{gfp}(T_{\lceil P \rceil^+})$ 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 existential variables (i.e. $var(C^-) \subseteq var(C^+)$). However, derivations, like derivation (13), are not considered in this approach. The next section take into account these derivations by using the $\mathcal{C}$-semantics approach (but no result about unfair derivations, like theorem 9, will be obtained).

5.3 SLD-proofs over a finite domain

SLD-proofs over a finite domain are SLD-derivations which do not compute infinite terms. In a more formal way, they can be defined as follows.

Definition 4 (SLD-proofs over a finite domain) *An SLD-proof over a finite domain 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 $\forall k \geq 0 \ \exists p > k \ \forall q \geq p \quad \theta_q \cdots \theta_p \cdots \theta_{k+1} R_k \approx \theta_p \cdots \theta_{k+1} R_k$.

It is important to note that it does not suffice that the condition holds for the initial query. Consider for example the program:

$$P = \{q(x) \leftarrow p(x) \ ; \ p(f(x)) \leftarrow p(x)\}$$

Even if during the derivation :

$$q(z) \overset{\theta_1=\begin{bmatrix} x \\ z \end{bmatrix}}{\rightarrow_P} p(z) \overset{\theta_2=\begin{bmatrix} z \\ f(x_1) \end{bmatrix}}{\rightarrow_P} p(x_1) \rightarrow_P \cdots \overset{\theta_{i-1}=\begin{bmatrix} x_{i-1} \\ f(x_i) \end{bmatrix}}{\rightarrow_P} p(x_i) \rightarrow_P \cdots$$

each θ_i is such that $\theta_i q(z) = q(z)$, this derivation computes the infinite term f^ω. We will need an equivalent definition for SLD-proofs over a finite domain. This definition follows from the next lemma.

Lemma 4 *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 SLD-proof over a finite domain iff:*

$$\forall i \geq 0 \quad \exists R \quad \forall n \geq i+1 \quad \theta_n \theta_{n-1} \cdots \theta_{i+1} R_i \leq R$$

where R is a query (i.e. R does not contain infinite atoms).

Proof. ($\Rightarrow$). By definition:

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

and it follows $\forall k \geq 0 \quad \forall q \geq k+1 \quad \theta_q \cdots \theta_{k+1} R_k \leq \theta_p \cdots \theta_{k+1} R_k$.
($\Leftarrow$). Let $k \geq 0$ and suppose there exists a query R of length ℓ such that $\forall n \geq k+1 \quad \theta_n \theta_{n-1} \cdots \theta_{k+1} R_k \leq R$. 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. 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 $A_1 \approx A_2$). This leads to the conclusion that the derivation considered is a derivation over a finite domain since $p = \max_{1 \leq i \leq \ell}(p_i)$ is such that $\forall q \geq p$, $\theta_q \cdots \theta_p \cdots \theta_{k+1} R_k \approx \theta_p \cdots \theta_{k+1} R_k$. ◀

$\mathcal{C}$-semantics results correspond SLD-refutations. Let us investigate infinite SLD-proofs over a finite domain. The soundness theorem can be proved directly by using proof trees.

Lemma 5 *If there exists an SLD-proof over a finite domain :*

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

then, for a $k \geq 0$, we have $\theta_k \cdots \theta_1 A_0 \in \mathsf{gfp}(T_{\lceil P \rceil})$.

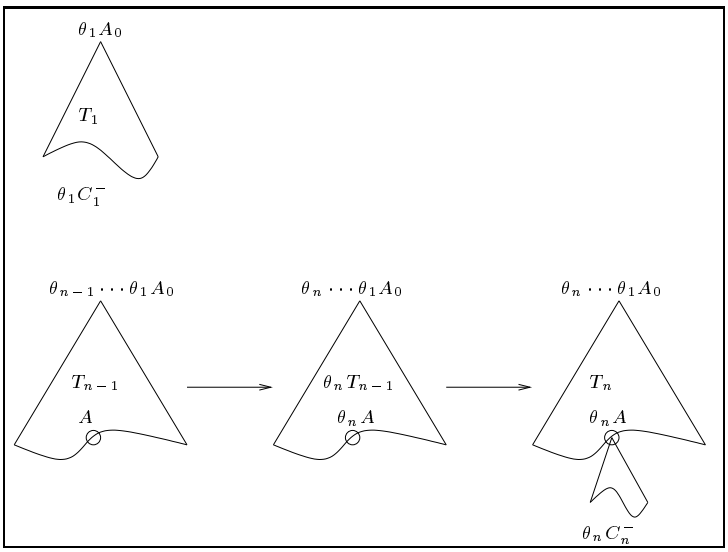

Figure 3: Proof of lemma 5

Proof. By theorem 1 and by lemma 1, it suffices to prove that for a natural k, there exists a proof tree of $\theta_k \cdots \theta_1 A_0$ 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 $\theta_i \cdots \theta_1 A_0$ for $\lceil P \rceil$ (see fig. 3).

- (T_1) T_1 is obtained from the first transition: its root is $\theta_1 A_0$ whose sons (which are leaves) are all the atoms occurring in $\theta_1 C_1^-$. Since $\theta_1 C_1 \in \lceil P \rceil$ and $\theta_1 A_0 = \theta_1 C_1^+$, T_1 is a partial proof tree of $\theta_1 A_0$ for $\lceil P \rceil$. Furthermore, atoms occurring in $R_1 = \theta_1 C_1^-$ are leaves of T_1.

- (T_n) Suppose T_{n-1} is a partial proof tree of $\theta_{n-1} \cdots \theta_1 A_0$ 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 $\theta_n \cdots \theta_1 A_0$ 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 sons of $\theta_n A$ (these sons 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 $q \geq k$, $\theta_q \cdots \theta_k \cdots \theta_1 A_0 \approx \theta_k \cdots \theta_1 A_0$, by iterating this process, we obtain a proof tree of $\theta_k \cdots \theta_1 A_0$ for $\lceil P \rceil$. Furthermore each leaf corresponds to a unit clause of $\lceil P \rceil$ since the derivation is fair. ◀

Theorem 10 (Soundness) *If there exists an SLD-proof over a finite domain:*

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

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

Proof. The proof is similar to the proof of lemma 5 : instead of building a sequence of partial proof trees, we build a sequence of tuples of n partial proof trees for $\lceil P \rceil$: $((T_1^1, \cdots, T_1^n), \cdots, (T_i^1, \cdots, T_i^n), \cdots)$ such that $\theta_i \cdots \theta_1 A_j$ is the root of T_i^j $(1 \leq j \leq n)$ and such that each atom occurring in R_i is a leaf of T_i^j for a j. ◀

Since, by theorem 4, there exists an SLD-proof with the program $\lceil P \rceil$ from each atom occurring in $\mathsf{CoInd}(T_{\lceil P \rceil})$, lemma 6 describes how to "translate" an SLD-derivation with $\lceil P \rceil$ into an SLD-derivation with P. It 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.

Lemma 6 (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 a idempotent renaming substitution such that $dom(\theta_i) = var(C_i^+)$, then there exists an SLD-proof over a finite domain:

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

such that for all $i \geq 1$, $\sigma_i A_0 = A_0$ 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. By lemma 15, there exists a set $\{C_{P,1}, \cdots, C_{P,i}, \cdots\}$ of variants of clauses of P such that:

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

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 $\theta_1\mu_1 A_0 = \theta_1\mu_1 C_{P,1}^+$. 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 2, the restriction σ_1 of $\theta_1\mu_1$ to the variables occurring in $C_{P,1}^+$ 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 the variables occurring in 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 from a query 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 the variables occurring in R_{i-1}':

$$R_{i-1}' \overset{C_{P,i},\sigma_i}{\rightarrow_P} R_i'$$

such that $\sigma_i A_0 = A_0$ and such that the restriction ρ_i of $\theta_i\mu_i\theta_{i-1}\mu_{i-1}\cdots\theta_1\mu_1$ to the variables occurring in R_i' 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:

$$dom(\rho_{i-1}) \subseteq var(R_{i-1}') \subseteq \left(\bigcup_{1\leq j<i} var(C_{P,j}) \cup var(A_0)\right)$$
$$\text{and} \quad var(C_{P,i}) \cap \left(\bigcup_{1\leq j<i} var(C_{P,j}) \cup var(A_0)\right) = \emptyset$$

it follows $\rho_{i-1}C_{P,i} = C_{P,i}$ 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 we have $\theta_i\mu_i\rho_{i-1}A' = \theta_i\mu_i\rho_{i-1}C_{P,i}^+$, and since $\theta_i\mu_i\cdots\theta_1\mu_1 A_0 = A_0$, by lemma 16,

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 transition:

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

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 $\theta_i\mu_i\rho_{i-1}R'_i = R_i$:

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

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 a finite domain. For this, let us prove that:

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

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:

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

Hence, 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 $\theta_n\mu_n v = v$. Therefore, 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, the derivation obtained is a derivation over a finite domain. ◀

We are now in position to prove the completeness theorem.

Theorem 11 (Completeness) *Given a definite program P and an atom A, if $A \in \mathsf{gfp}(T_{\lceil P \rceil})$, then there exists an SLD-proof over a finite domain from A with P :*

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

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

Proof. Completeness theorem follows from theorem 1, lemma 1, theorem 4 and lemma 6. ◀

6 Conclusion

In this paper, semantics of nonterminating derivations has been investigated within a proof-theoretic framework: definite clauses have been considered as rules of a formal system. Following this approach, 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: an atom is the starting point of an infinite derivation over a finite domain if and only if it is in the greatest fixpoint of the transformation $T_{\lceil P \rceil}$.

The restriction to the class of derivations over a finite domain is justified by incompleteness results of others approaches, allowing infinite terms, in which the greatest fixpoint construction, corresponding to the "logic program as co-inductive definition" paradigm, is not equivalent to the operational semantics: co-induction is too rich to give a semantics to nonterminating SLD-derivations. This observation, illustrated in section 4.2.2, explains why most attempts to give a complete semantics to derivations computing infinite terms have not been successful. Therefore, while all the approaches existing in this area are based on the concept of "atoms computable at infinity", we have presented a semantics based on the concept of "atoms provable at infinity".

It seems that the operational notion of "computability at infinity" (associated with infinite derivations computing infinite terms) is better captured by a least fixpoint characterisation. This idea has been developped by G. Levi and C. Palamidessi in [25] and revisited in [23]. 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, characterised 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 characterised 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, the semantics obtained is sound but not complete.

Of course, a satisfactory semantics for all infinite derivations from a definite program has not yet been found, but this paper allows us to gain a better understanding of the problem and suggests an area for future investigations.

References

[1] M.A. Nait Abdallah. On the interpretation of infinite computations in logic programming. In J. Paredaens, editor, *11th International Colloquium on Automata, Languages and Programming, ICALP'84*, volume 172 of *Lecture Notes in Computer Science*, pages 358–370. 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(1):67–76, 1985.

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

[4] K.R. Apt and M.H. Van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, 1982.

[5] T. Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *Selected Papers of the 1st International Workshop on Types for Proofs and Programs, TYPES'93*, volume 806 of *Lecture Notes in Computer Science*, pages 62–78. Springer-Verlag, 1994.

[6] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25(2):95–169, 1983.

[7] F.S. de Boer, A. Di Pierro, and C. Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theoretical Computer Science*, 151(1):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] N. Dershowitz, S. Kaplan, and D.A. Plaisted. Rewrite, rewrite, rewrite, rewrite, rewrite. *Theoretical Computer Science*, 83(1):71–96, 1991.

[10] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A model-theoretic reconstruction of the operational semantics of logic programs. *Information and Computation*, 103(1):86–113, 1993.

[11] M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[12] M. Fitting. Metric methods three examples and a theorem. *Journal of Logic Programming*, 21(3):113–127, 1994.

[13] C.E. Giménez. *Un calcul des constructions infinies et son application à la vérification de systèmes communicants.* Thèse de doctorat, Ecole Normale Supérieure de Lyon, 1996.

[14] J.Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[15] W.G. Golson. Toward a declarative semantics for infinite objects in logic programming. *Journal of Logic Programming*, 5(2):151–164, 1988.

[16] M. Hagiya and T. Sakurai. Foundation of logic programming based on inductive definition. *New Generation Computing*, 2(1):59–77, 1984.

[17] L. Hallnäs and P. Schroeder-Heister. A proof-theoretic approach to logic programming. I. Clauses as rules. *Journal of Logic and Computation*, 1(2):261–283, 1990.

[18] L. Hallnäs and P. Schroeder-Heister. A proof-theoretic approach to logic programming. II. Programs as definitions. *Journal of Logic and Computation*, 1(5):635–660, 1990.

[19] J. Hein. Completions of perpetual logic programs. *Theoretical Computer Science*, 99(1):65–78, 1992.

[20] J. Jaffar and P.J. Stuckey. Canonical logic programs. *Journal of Logic Programming*, 3(2):143–155, 1986.

[21] J. Jaffar and P.J. Stuckey. Semantics of infinite tree logic programming. *Theoretical Computer Science*, 46(2-3):141–158, 1986.

[22] J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. de Vries. Infinitary lambda calculus. *Theoretical Computer Science*, 175(1):93–125, 1997.

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

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

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

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

A Proofs

Proof of lemma 1 ($\subseteq$). Let $x \in \mathsf{Coind}(\Phi)$, by theorem 1, $x \in \mathsf{gfp}(T_\Phi)$ and it follows $x \in T_\Phi(\mathsf{gfp}(T_\Phi))$. Hence, there exists a rule $x \leftarrow x_1, \cdots, x_q \in \Phi$ such that $\{x_1, \cdots, x_q\} \subseteq \mathsf{gfp}(T_\Phi)$. We can obtain a partial proof tree with x as root where $\{x_1, \cdots, x_q\}$ are sons of x. Now, it suffices to iterate on the sons of x, which are in $\mathsf{Coind}(\Phi)$, to get a proof tree of x for Φ. ($\supseteq$). Let x be the root of a proof tree for Φ and Z be the set of nodes occurring in the proof tree. Let us prove that $Z \subseteq T_\Phi(Z)$. If $z \in Z$, then, there exists $z \leftarrow z_1, \cdots, z_q \in \Phi$ where $\{z_1, \cdots, z_q\}$ are sons of z. Hence, $\{z_1, \cdots, z_q\} \subseteq Z$ and it follows $z \in T_\Phi(Z)$. Since $Z \subseteq T_\Phi(Z)$, Z is Φ-dense and, by definition, we have $x \in \mathsf{Coind}(\Phi)$.

Lemma 7 *If Z is a set of variables and C is the clause $A \leftarrow B_1, \cdots, B_q$, then there exists a clause C' and an idempotent renaming substitution ρ such that:*

$$\begin{array}{ll} var(C') \cap (var(C) \cup Z) = \emptyset & \rho C' = C \\ dom(\rho) = var(C') & range(\rho) = var(C) \end{array}$$

Proof of lemma 7 If $var(C) = \{x_1, \cdots, x_n\}$ and $\{y_1, \cdots, y_n\}$ is a collection of distinct variables such that $\{y_1, \cdots, y_n\} \cap (var(C) \cup Z) = \emptyset$, then C' and ρ can be defined by:

$$C' = \left[\begin{array}{ccc} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{array} \right] C \qquad \rho = \left[\begin{array}{ccc} y_1 & \cdots & y_n \\ x_1 & \cdots & x_n \end{array} \right]$$

and satisfy the desired properties.

Lemma 8 *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 transition:*

$$r_0 C_T^+ \stackrel{C,\theta}{\rightarrow} \theta C^-$$

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 $r r_0 C_T^- = \theta C^-$.

Proof of lemma 8 By lemma 7, there exists a clause C and an idempotent renaming substitution ρ such that:

$$\begin{array}{ll} var(C) \cap (var(r_0 C_T) \cup Z) = \emptyset & \rho C = r_0 C_T \\ dom(\rho) = var(C) & range(\rho) = var(r_0 C_T) \end{array}$$

By lemma 2, the restriction θ of ρ to the variables occurring in C^+ is a mgu of $r_0 C_T^+$ and C^+. Hence the following resolution step is correct:

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

If ρ is the substitution defined by:

$$\rho = \begin{bmatrix} x_1^+ & ... & x_{n_1}^+ & x_1^\pm & ... & x_{n_2}^\pm & x_1^- & ... & x_{n_3}^- \\ y_1^+ & ... & y_{n_1}^+ & y_1^\pm & ... & y_{n_2}^\pm & y_1^- & ... & y_{n_3}^- \end{bmatrix}$$

$$\begin{array}{rl} & \{x_1^+, ..., x_{n_1}^+\} = var(C^+) \backslash var(C^-) \\ \text{where} & \{x_1^\pm, ..., x_{n_2}^\pm\} = var(C^+) \cap var(C^-) \\ & \{x_1^-, ..., x_{n_3}^-\} = var(C^-) \backslash var(C^+) \end{array}$$

then we can define the renaming substitution $r_1 = r r_0$ by:

$$r_1 = \begin{bmatrix} y_1^- & ... & y_{n_3}^- \\ x_1^- & ... & x_{n_3}^- \end{bmatrix} r_0$$

Let us prove that $r_1 C_T^- = \theta C^-$:

$$\begin{array}{rcl} r_1 C_T^- & = & \begin{bmatrix} y_1^- & ... & y_{n_3}^- \\ x_1^- & ... & x_{n_3}^- \end{bmatrix} r_0 C_T^- \\ & = & \begin{bmatrix} y_1^- & ... & y_{n_3}^- \\ x_1^- & ... & x_{n_3}^- \end{bmatrix} \rho C^- \\ & = & \begin{bmatrix} y_1^- & ... & y_{n_3}^- \\ x_1^- & ... & x_{n_3}^- \end{bmatrix} \begin{bmatrix} x_1^\pm & ... & x_{n_2}^\pm & x_1^- & ... & x_{n_3}^- \\ y_1^\pm & ... & y_{n_2}^\pm & y_1^- & ... & y_{n_3}^- \end{bmatrix} C^- \\ & = & \begin{bmatrix} x_1^\pm & ... & x_{n_2}^\pm \\ y_1^\pm & ... & y_{n_2}^\pm \end{bmatrix} C^- \\ & = & \theta C^- \end{array}$$

Proof of lemma 2. First, note that since $var(A_1) \cap var(A_2) = \emptyset$ and $dom(\theta) \subseteq var(A_2)$, we have $\theta A_1 = A_1 = \theta A_2$. Therefore, θ is a unifier of A_1 and A_2. Furthermore, for every variable v, if $v \in range(\theta)$, then there exists a variable $y \in dom(\theta) \subseteq var(A_2)$ such that $v \in \theta y$. It follows $v \in \theta A_2 = A_1$ and we get $v \notin dom(\theta)$ since $\theta A_1 = A_1$. Hence, θ is an idempotent substitution. Now, let us prove that θ is a mgu. For this, let μ be a unifier of A_1 and A_2. Since $A_1 = \theta A_2$, we have $\mu\theta A_2 = \mu A_2$ and it suffices to prove that for every variable v, $\mu\theta v = \mu v$. If $v \notin dom(\theta)$ then $\mu\theta v = \mu v$ is immediate, else, since $dom(\theta) \subseteq var(A_2)$ we can conclude since $\mu\theta A_2 = \mu A_2$. This leads to $\theta \leq \mu$.

End of the proof of theorem 4 We prove here that the derivation obtained in the proof of the theorem 4 is valid and satisfies the desired properties. Given an index $\vec{\imath} \in \mathbb{N}^\star$ of T, we write $\overleftarrow{\vec{\imath}}$ the index of the node just before $A_{\vec{\imath}}$ in $\mathcal{L}$. In a more formal way:

$$\overleftarrow{\vec{\imath}} = \max^{\prec}\{\vec{\jmath},\ \vec{\jmath} \prec \vec{\imath}\}$$

$\triangleright k$ and $k \triangleleft$ are indexes of T defined as follows:

$$\forall k \in \mathbb{N} \qquad \begin{array}{l} \triangleright k = \max^{\prec_\ell}\{\vec{\imath},\ |\vec{\imath}| = k\} \\ k \triangleleft = \min^{\prec_\ell}\{\vec{\imath},\ |\vec{\imath}| = k\} \end{array}$$

In order to prove the desired properties of the derivation, it suffices to prove the following assertions.

(1). $\forall \vec{\imath} \quad var(C_{\vec{\imath}}) \cap \left(var(A) \cup \bigcup_{\vec{\jmath} \prec \vec{\imath}} var(C_{\vec{\jmath}}) \right) = \emptyset$
Immediate by definition of $Z_{\vec{\imath}}$.

(2). The resolution step $A \overset{C_\varepsilon, \theta_\varepsilon, Z_T}{\rightarrow_P} R$ is correct.
Immediate since it corresponds to t_ε.

(3). $\forall \vec{\imath} \quad range(r_1^{\vec{\imath}}) \subseteq \bigcup_{\vec{\jmath} \preceq \vec{\imath}} \Big(var(C_{\vec{\jmath}}^-) \backslash var(C_{\vec{\jmath}}^+) \Big)$
Induction over $\vec{\imath}$.
• If $|\vec{\imath}| = 0$, then we have:

$$range(r_1^\varepsilon) = range(r_0^\varepsilon r^\varepsilon) = range(r^\varepsilon) = var(C_\varepsilon^-) \backslash var(C_\varepsilon^+)$$

• If $\vec{\imath} = \vec{\jmath}k$, then, by induction hypothesis, we can conclude since:

$$r_1^{\vec{\jmath}k} = r^{\vec{\jmath}k} r_0^{\vec{\jmath}k} = r^{\vec{\jmath}k} r_1^{\vec{\jmath}} \qquad range(r^{\vec{\jmath}k}) = var(C_{\vec{\jmath}k}^-) \backslash var(C_{\vec{\jmath}k}^+)$$

(4). $\forall p \in \mathbb{N} \left\{ \begin{array}{ll} \forall \vec{\imath}\ ((p+1) \triangleleft \preceq \vec{\imath} \preceq \triangleright(p+1)) & r_0^{\vec{\imath}} A_{\vec{\imath}} \in R_{\triangleright p} \\ \forall \vec{\imath}\ ((p+1) \triangleleft \prec \vec{\imath} \preceq \triangleright(p+1)) & r_0^{\vec{\imath}} A_{\vec{\imath}} \in R_{\overleftarrow{\vec{\imath}}} \end{array} \right.$
(in particular $r_0^{(p+1)\triangleleft} A_{(p+1)\triangleleft} \in R_{\triangleright p}$)
Induction over p.
• If $p = 0$, then for every k such that $1 \leq k \leq n$, we have:

$$r_0^k A_k = r_1^\varepsilon A_k \in \theta_\varepsilon C_\varepsilon^- = R_\varepsilon = R_{\triangleright 0}$$

Let us prove that for every k $(1 < k \leq n)$, we have $r_0^k A_k = r_1^\varepsilon A_k \in R_{k-1}$. Since $R_\varepsilon = \theta_\varepsilon C_\varepsilon^- = r_1^\varepsilon C_{T,\varepsilon}^-$, we know that $r_1^\varepsilon A_k \in R_\varepsilon$. Furthermore, for every m $(1 \leq m \leq k-1)$, we have:

$$dom(\theta_m) = var(C_m^+) \quad \text{and} \quad var(C_m) \cap \left(Z_T \cup \bigcup_{j \prec m} C_j \right) = \emptyset$$

Since $r_1^\varepsilon = r^\varepsilon r_0^\varepsilon = r^\varepsilon$ where $range(r^\varepsilon) = var(C_\varepsilon^-) \backslash var(C_\varepsilon^+)$, we have:

$$var(r_1^\varepsilon A_k) \subseteq (Z_T \cup var(C_\varepsilon))$$

Hence, it follows, $var(r_1^\varepsilon A_k) \cap var(C_m) = \emptyset$ and we have $\theta_m r_1^\varepsilon A_k = r_1^\varepsilon A_k$. We can now conclude since at each resolution step m, $r_1^\varepsilon A_m$ is the selected atom and the mgu used θ_m does not affect the variables occurring in $r_1^\varepsilon A_k$. Therefore $r_1^\varepsilon A_k$ occurs in R_m.

• If $p > 0$, then, by induction hypothesis, we have:

$$\begin{cases} \forall \vec{\imath} \ (p\triangleleft \preceq \vec{\imath} \preceq \triangleright p) & r_0^{\vec{\imath}} A_{\vec{\imath}} \in R_{\triangleright(p-1)} \\ \forall \vec{\imath} \ (p\triangleleft \prec \vec{\imath} \preceq \triangleright p) & r_0^{\vec{\imath}} A_{\vec{\imath}} \in R_{\overleftarrow{\imath}} \end{cases}$$

and it suffices to prove:

$$\forall \vec{\jmath} \ (p\triangleleft \prec \vec{\jmath} \preceq \triangleright p) \quad dom(\theta_{\vec{\jmath}}) \cap var\left(\bigcup_{\vec{u} \prec \vec{\jmath}} \bigcup_{k=1}^{n_{\vec{u}}} r_0^{\vec{u}k} A_{\vec{u}k} \right) = \emptyset$$

If $\vec{\jmath}$ is such that $p\triangleleft \prec \vec{\jmath} \preceq \triangleright p$, then we know that:

$$dom(\theta_{\vec{\jmath}}) = var(C_{\vec{\jmath}}^+) \quad \text{and} \quad var(C_{\vec{\jmath}}) \cap \left(Z_T \cup \bigcup_{\vec{u} \prec \vec{\jmath}} C_{\vec{u}} \right) = \emptyset$$

Furthermore, we have:

$$var\left(\bigcup_{\vec{u} \prec \vec{\jmath}} \bigcup_{k=1}^{n_{\vec{u}}} r_0^{\vec{u}k} A_{\vec{u}k} \right) = var\left(\bigcup_{\vec{u} \prec \vec{\jmath}} \bigcup_{k=1}^{n_{\vec{u}}} r_1^{\vec{u}} A_{\vec{u}k} \right) = var\left(\bigcup_{\vec{u} \prec \vec{\jmath}} r_1^{\vec{u}} C_{T,\vec{u}}^- \right)$$

and since:

$$var\left(\bigcup_{\vec{u} \prec \vec{\jmath}} r_1^{\vec{u}} C_{T,\vec{u}}^- \right) \subseteq \bigcup_{\vec{u} \prec \vec{\jmath}} \left(range(r_1^{\vec{u}}) \cup var(C_{T,\vec{u}}^-) \right)$$

(1) and (3) allow to conclude.

Lemma 9 *$T_{\lceil P \rceil^+}$ is monotone and $\uparrow$-continuous.*

Proof of lemma 9 Since $T_{\lceil P \rceil^+}$ is defined from the rule set $\lceil P \rceil^+$, it is clearly monotone. Every clause in P has a finite body and $\lceil P \rceil^+$ is finitary and we can conclude.

Lemma 10 *If I is a $\uparrow$-closed set, then $T_{\lceil P \rceil^+}(I)$ is also a $\uparrow$-closed set.*

Proof of lemma 10 Let $A \in T_{\lceil P \rceil^+}(I)$. By definition, there exists a clause $A' \leftarrow B_1, \cdots, B_q$ in P such that for a substitution θ satisfying $dom(\theta) \subseteq var(A')$, we have $\theta A' = A$ and $\{\theta B_1, \cdots, \theta B_q\} \subseteq I$. Let A_0 be an atom such that $A \leq A_0$. There exists a substitution μ such that $dom(\mu) \subseteq var(A)$ and $\mu A = A_0$. By considering the restriction of $\mu\theta$ to the variables occurring in A', it follows $\mu\theta A' = A_0$ and we can conclude since $\{\mu\theta B_1, \cdots, \mu\theta B_q\} \subseteq I$ because I is $\uparrow$-closed.

Lemma 11 $\mathsf{Ind}(T_{\lceil P \rceil^+}) = \mathsf{lfp}(T_{\lceil P \rceil^+}) = T^{\uparrow\omega}_{\lceil P \rceil^+}$

Proof of lemma 11 By lemma 9, $T_{\lceil P \rceil^+}$ is monotone and $\uparrow$-continuous and by theorem 1, we can conclude.

Lemma 12 *A $\mathcal{C}$-interpretation I is a $\mathcal{C}^+$-model of a definite program P iff $T_{\lceil P \rceil^+}(I) \subseteq I$.*

Proof of lemma 12 ($\Rightarrow$). Let I be a $\mathcal{C}^+$-model of P and $A \in T_{\lceil P \rceil^+}(I)$. By definition, there exists a clause $A \leftarrow B_1, \cdots, B_q \in \lceil P \rceil^+$ such that $\{B_1, \cdots, B_q\} \subseteq I$. Since I is a $\mathcal{C}^+$-model of P, it follows $A \in I$ and we can conclude. ($\Leftarrow$). Let I be a $\mathcal{C}$-interpretation such that $T_{\lceil P \rceil^+}(I) \subseteq I$. If $A \leftarrow B_1, \cdots, B_q$ is a clause in P and θ is a substitution such that $dom(\theta) \subseteq var(A)$, then if $\{\theta B_1, \cdots, \theta B_q\} \subseteq I$, we have $\theta A \in T_{\lceil P \rceil^+}(I)$ and since $T_{\lceil P \rceil^+}(I) \subseteq I$ it follows $\theta A \in I$. Hence I is a $\mathcal{C}^+$-model of P.

Theorem 12 $\mathcal{M}_P^{\mathcal{C}^+} = \mathsf{lfp}(T_{\lceil P \rceil^+}) = \mathsf{Ind}(T_{\lceil P \rceil^+}) = T^{\uparrow\omega}_{\lceil P \rceil^+}$

Proof of theorem 12 By definition, $\mathcal{M}_P^{\mathcal{C}^+}$ is the intersection of all $\mathcal{C}^+$-models of P, which are, by lemma 12, $T_{\lceil P \rceil^+}$-closed sets, which corresponds to $\mathsf{Ind}(T_{\lceil P \rceil^+})$, and by lemma 11 we can conclude.

Lemma 13 $\mathcal{M}_P^{\mathcal{C}^+} \subseteq \mathcal{M}_P^{\mathcal{C}}$

Proof of lemma 13. Since $\lceil P \rceil^+ \subseteq \lceil P \rceil$, we have:

$$\begin{array}{ll} & \forall I \quad T_{\lceil P \rceil^+}(I) \subseteq T_{\lceil P \rceil}(I) \\ \Rightarrow & \forall n \quad T^{\uparrow n}_{\lceil P \rceil^+} \subseteq T^{\uparrow n}_{\lceil P \rceil} \\ \Rightarrow & T^{\uparrow \omega}_{\lceil P \rceil^+} \subseteq T^{\uparrow \omega}_{\lceil P \rceil} \\ \Rightarrow & \mathcal{M}^{\mathcal{C}^+}_P \subseteq \mathcal{M}^{\mathcal{C}}_P \quad \text{(by theorem 12)} \end{array}$$

Lemma 14 *Let P be a definite program and A_0 be an atom. Given a possibly infinite set of clauses $\{C_1, \cdots, C_i, \cdots\} \subseteq \lceil P \rceil^+$ such that:*

$$\forall i > 0 \quad var(C_i) \cap \left(var(A_0) \cup \bigcup_{1 \leq j < i} var(C_j) \right) = \emptyset$$

there exists a set $\{C_{P,1}, \cdots, C_{P,i}, \cdots\}$ of variants of clauses of P such that:

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

and such that every clause $C_{P,i}$ satisfies $C_i = \mu_i C_{P,i}$ where μ_i is an idempotent substitution satisfying $dom(\mu_i) = var(C^+_{P,i})$.

Proof of lemma 14 Since $C_i \in \lceil P \rceil^+$, for all i, there exist a substitution σ_i and a clause $C^P_i \in P$ such that $C_i = \sigma_i C^P_i$ and $dom(\sigma_i) \subseteq var(C^{P\,+}_i)$:

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

Let $\{y_1, \cdots, y_q\} = var(C^{P\,+}_i) \backslash dom(\sigma_i)$. In order to define a clause $C_{P,i} = r_i C^P_i$, as a variant of a clause in P, we introduce the following idempotent renaming substitution:

$$r_i = \begin{bmatrix} x_1 & \cdots & x_k & y_1 & \cdots & y_q \\ w_1 & \cdots & w_k & z_1 & \cdots & z_q \end{bmatrix}$$

where:

$$range(r_i) \cap \left(var(A_0) \cup \bigcup_{1 \leq j < i} var(C_{P,j}) \cup \bigcup_{j \geq 1} var(C_j) \cup var(C^P_i) \right) = \emptyset$$

Let us prove that:

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

First, we prove that:

$$\forall i > 0 \quad var(C_{P,i}) \subseteq (range(r_i) \cup var(C_i))$$

For this, let $v \in var(C_{P,i})$. Since $C_{P,i} = r_i C_i^P$, two cases are possible:

1. if $v \in range(r_i)$, then we can conclude
2. else, $v \in var(C_i^P)$, and:
   (a) either $v \in var(C_i^{P+})$ and since $dom(r_i) = var(C_i^{P+})$, r_i is idempotent induces a contradiction
   (b) or $v \in var(C_i^{P-}) \backslash var(C_i^{P+})$ and since $C_i = \sigma_i C_i^P$ and $dom(\sigma_i) \subseteq var(C_i^{P+})$, it follows $v \in C_i$ and we can conclude

By definition of r_i, variables occurring both in $C_{P,i}$ and $range(r_i)$ satisfy the property and since $var(C_i) \cap var(A_0) = \emptyset$, it suffices to prove that:

$$var(C_i) \cap \bigcup_{1 \leq j < i} var(C_{P,j}) = \emptyset$$

This property holds if:

$$var(C_i) \cap \bigcup_{1 \leq j < i} (range(r_j) \cup var(C_j)) = \emptyset$$

which follows from:

$$\forall j > 0 \quad range(r_j) \cap \bigcup_{k \geq 1} var(C_k) = \emptyset \quad \text{and} \quad var(C_i) \cap \bigcup_{1 \leq j < i} var(C_j) = \emptyset$$

Now, we have to prove that there exists an idempotent substitution μ_i such that $dom(\mu_i) = var(C_{P,i}^+)$ and $C_i = \mu_i C_{P,i}$. This substitution is defined by:

$$\mu_i = \begin{bmatrix} w_1 & \cdots & w_k & z_1 & \cdots & z_q \\ t_1 & \cdots & t_k & y_1 & \cdots & y_q \end{bmatrix}$$

Let us prove $C_i = \sigma_i C_i^P = \mu_i r_i C_i^P = \mu_i C_{P,i}$. If $v \in var(C_i^P)$, then two cases are possible:

1. if $v \in var(C_i^{P+})$ then:
   (a) either $v = y_j$ $(1 \leq j \leq q)$ and we can conclude since $\mu_i r_i y_j = \mu_i z_j = y_j = \sigma_i y_j$.
   (b) or $v = x_j$ $(1 \leq j \leq k)$ and we can also conclude since $\mu_i r_i x_j = \mu_i w_j = t_j = \sigma_i x_j$.
2. else $v \in var(C_i^{P-}) \backslash var(C_i^{P+})$ and it follows $v \notin dom(\sigma_i)$ and $v \notin dom(r_i)$, hence we can conclude since, by definition of r_i, we have $v \notin dom(\mu_i) = range(r_i)$.

To terminate, we prove that μ_i is idempotent, or equivalently $dom(\mu_i) \cap range(\mu_i) = \emptyset$. Since r_i is idempotent and $dom(\mu_i) = range(r_i)$, we have $dom(\mu_i) \cap \{y_1, \cdots, y_q\} = \emptyset$. Furthermore, since $C_i = \sigma_i C_i^P$, we know that:

$$\bigcup_{1 \leq j \leq k} var(t_j) \subseteq var(C_i)$$

and we can conclude since $range(r_i) \cap var(C_i) = \emptyset$.

Lemma 15 *Let P be a definite program and A_0 be an atom. Given a possibly infinite set of clauses $\{C_1, \cdots, C_i, \cdots\} \subseteq \lceil P \rceil$ such that :*

$$\forall i > 0 \quad var(C_i) \cap \left(var(A_0) \cup \bigcup_{1 \leq j < i} var(C_j) \right) = \emptyset$$

there exists a set $\{C_{P,1}, \cdots, C_{P,i}, \cdots\}$ of variants of clauses of P such that :

$$\forall i > 0 \quad var(C_{P,i}) \cap \left(var(A_0) \cup \bigcup_{1 \leq j < i} var(C_{P,j}) \cup \bigcup_{k \geq 0} var(C_k) \right) = \emptyset$$

and such that every clause $C_{P,i}$ satisfies $C_i = \mu_i C_{P,i}$ where μ_i is an idempotent substitution satisfying $dom(\mu_i) = var(C_{P,i})$.

Proof of lemma 15 Since $C_i \in \lceil P \rceil$, for all i, there exist a substitution σ_i and a clause $C_i^P \in P$, such that $C_i = \sigma_i C_i^P$ and $dom(\sigma_i) \subseteq var(C_i^P)$:

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

Let $\{y_1, \cdots, y_q\} = var(C_i^P) \backslash dom(\sigma_i)$. In order to define a clause $C_{P,i} = r_i C_i^P$, as a variant of a clause in P, we introduce the following idempotent renaming substitution:

$$r_i = \left[\begin{array}{cccccc} x_1 & \cdots & x_k & y_1 & \cdots & y_q \\ w_1 & \cdots & w_k & z_1 & \cdots & z_q \end{array} \right]$$

where :

$$range(r_i) \cap \left(var(A_0) \cup \bigcup_{1 \leq j < i} var(C_{P,j}) \cup \bigcup_{j \geq 1} var(C_j) \right) = \emptyset$$

Now, let us prove that there exists an idempotent substitution μ_i such that $dom(\mu_1) = var(C_{P,i})$ and $C_i = \mu_i C_{P,i}$. This substitution is defined by:

$$\mu_i = \left[\begin{array}{cccccc} w_1 & \cdots & w_k & z_1 & \cdots & z_q \\ t_1 & \cdots & t_k & y_1 & \cdots & y_q \end{array} \right]$$

In order to prove $C_i = \sigma_i C_i^P = \mu_i r_i C_i^P = \mu_i C_{P,i}$, let $v \in var(C_i^P)$. Two cases are possible : either $v \notin dom(\sigma_i)$ and it follows $v = y_j$ $(1 \leq j \leq q)$ which settles the claim since $\sigma_i y_j = y_j = \mu_i z_j = \mu_i r_i y_j$, or $v \in dom(\sigma_i)$ and we have $v = x_j$ $(1 \leq j \leq k)$ which allows to conclude since $\sigma_i x_j = t_j = \mu_i w_j = \mu_i r_i x_j$. Let us prove that μ_i is idempotent, or equivalently $dom(\mu_i) \cap range(\mu_i) = \emptyset$. Since r_i is idempotent and $dom(\mu_i) = range(r_i)$, we have $dom(\mu_i) \cap \{y_1, \cdots, y_q\} = \emptyset$. Furthermore, since $C_i = \sigma_i C_i^P$, we know that:

$$\bigcup_{1 \leq j \leq k} var(t_j) \subseteq var(C_i)$$

and we can conclude since $range(r_i) \cap var(C_i) = \emptyset$.

Lemma 16 *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 $\rho A_0 = A_0$.*

Proof of lemma 16 Since $\theta A = \theta B$, there exists a mgu σ of A and B such that $\sigma \leq \theta$. Hence for a substitution η, we have $\eta\sigma = \theta$. Furthermore, from $\theta A_0 = A_0$, it follows $\eta\sigma A_0 = A_0$ and therefore, σ can be viewed as a renaming substitution for the variables occurring in A_0:

$$\sigma = \left[\begin{array}{cccccc} v_1 & \cdots & v_k & x_1 & \cdots & x_n \\ t_1 & \cdots & t_k & y_1 & \cdots & y_n \end{array} \right] \begin{array}{l} \{v_1, \cdots, v_k\} = dom(\sigma) \backslash var(A_0) \\ \{x_1, \cdots, x_k\} = dom(\sigma) \cap var(A_0) \end{array}$$

Let us define the following idempotent renaming substitution (corresponding to a restriction of η):

$$r = \begin{bmatrix} y_1 & \cdots & y_n \\ x_1 & \cdots & x_n \end{bmatrix}$$

and let us prove that $\rho = r\sigma$. First note that clearly we have $\rho A_0 = A_0$. Therefore, it suffices to prove that ρ is a mgu of A and B. $\rho A = \rho B$ is immediate (because $\sigma A = \sigma B$). In order to prove that ρ is idempotent, let $v \in dom(\rho)$. Two cases are possible.

1. If $v \in dom(\sigma)$, then $v = v_j$ $(1 \leq j \leq k)$ since $\rho x_i = x_i$ for every $i \in \{1, \cdots, n\}$. Therefore, $v \notin \{x_1, \cdots, x_n\}$ and $v \notin \cup_{1 \leq i \leq k} var(rt_i)$ since $\cup_{1 \leq i \leq k} var(rt_i) \subseteq \cup_{1 \leq i \leq k} var(t_i) \cup range(r)$ and σ is idempotent. Hence, $v \notin range(\rho)$.

2. If $v \in dom(r)$, then $v = y_j$ $(1 \leq j \leq n)$ and since clearly $v \notin \{x_1, \cdots, x_n\}$ and $v \notin \cup_{1 \leq i \leq k} var(rt_i)$, we have $v \notin range(\rho)$.

To terminate, we have to prove that ρ is minimal. For this, let μ be a substitution such that $\mu A = \mu B$. Since σ is a mgu of A and B we have $\sigma \leq \mu$ and there exists a substitution ν such that $\nu\sigma = \mu$. Therefore, $\rho \leq \mu$ since we prove that $\nu r^{-1}\rho = \nu r^{-1} r\sigma = \nu\sigma = \mu$ where r^{-1} is the inverse of r (i.e. $rr^{-1} = r^{-1}r = s_{id}$). For this, let w be a variable. Two cases are possible.

1. If $w \in dom(\sigma)$, then:

   (a) if $w = v_j$ $(1 \leq j \leq k)$, then $\nu r^{-1} r\sigma v_j = \nu r^{-1} r t_j$ and we can conclude since $r^{-1} r t_j = t_j$ because for every variable $y \in var(t_j)$:

      i. either $y \in dom(r)$ and $r^{-1} r y = y$ is immediate

      ii. or $y \notin dom(r)$ and we have $r^{-1} y = y$ since σ is idempotent and $var(t_j) \subseteq range(\sigma)$ and $dom(r^{-1}) \subseteq dom(\sigma)$.

   (b) if $w = x_j$ $(1 \leq j \leq n)$, then we can conclude since $\nu r^{-1} r\sigma x_j = \nu r^{-1} r y_j = \nu r^{-1} x_j = \nu y_j = \nu\sigma x_j$.

2. If $w \notin dom(\sigma)$, then let us prove that $\nu r^{-1} r w = \nu w$. Indeed, either $w \in dom(r)$ and $r^{-1} r w = w$ which settles the claim, or $w \notin dom(r)$ and we have $r^{-1} w = w$ since $dom(r^{-1}) \subseteq dom(\sigma)$ and $w \notin dom(\sigma)$.