
Preuves infinies en Programmation logique

Mathieu Jaume

CERMICS – ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES
6-8 Av. Blaise Pascal, Champs sur Marne
77455 Marne-La-Vallee Cedex 2, France
jaume@cermics.enpc.fr

RÉSUMÉ. Cet article présente une sémantique complète pour une classe de SLD-dérivations infinies. Plusieurs approches ont déjà été développées et sont basées sur la notion d'atome infini calculable à l'infini. L'univers du discours considéré dans de telles approches contient des termes infinis et la dénotation d'un programme est le plus souvent définie en terme de plus grand point fixe. Hélas, aucune de ces approches n'a permis de définir une sémantique complète. Nous prenons ici le parti opposé en nous concentrant sur les dérivations infinies qui ne calculent pas de termes infinis : cet article étudie la contre-partie opérationnelle du plus grand point fixe de l'opérateur de conséquence immédiate défini pour la C-sémantique. Les définitions co-inductives fournissent un cadre adéquat pour expliquer les phénomènes d'incomplétude et permettent de définir une sémantique valide et complète pour la classe des dérivations infinies qui ne calculent pas de termes infinis.

ABSTRACT. This paper focuses on the assignment of meaning to some nonterminating SLD derivations in logic programming. Several approaches have been developed by considering infinite elements in the universe of the discourse but none are complete. We investigate here infinite derivations over the domain of finite terms (i.e., derivations which do not compute infinite terms) and show they correspond to co-induction proofs. A sound and complete semantics for this class of derivations, based on the “logic programs as co-inductive definitions” paradigm, is defined.

MOTS-CLÉS : SLD-résolution, définitions co-inductives, SLD-dérivations infinies.

KEYWORDS: SLD-resolution, co-inductive definitions, infinite SLD-derivations.

1. Introduction – Motivations

Les résultats classiques de la programmation logique [LLO 87] concernent les calculs finis (i.e., définissent une sémantique pour les SLD-réfutations). En effet, il existe une tradition en informatique qui veut que tout « bon » programme soit un programme dont l'exécution termine. La programmation logique n'échappe pas à cette tradition et les propriétés de terminaison des programmes logiques ont fait l'objet de nombreux travaux. Toutefois, certains programmes ont « naturellement » vocation à donner lieu à une exécution infinie et l'étude de la sémantique des « calculs » infinis a désormais été envisagée dans divers paradigmes (λ -calcul, systèmes de réécriture, programmation logique, programmation concurrente par contraintes [BOE 95], ...) et permet d'aborder des notions comme la concurrence ou la réactivité. En effet, certains objets sont, par nature, infinis et les programmes qui permettent de les construire, même s'ils ne terminent pas, effectuent bien un calcul « utile en un certain sens ». Par exemple, dans le domaine de la programmation logique, le programme permettant de construire la liste (infinie) des entiers naturels consécutifs à partir d'un certain rang s'écrit :

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

puisque pour tout $k \in \mathbb{N}$, une dérivation infinie à partir de la requête $\text{LN}(k, l)$ fournit à chaque étape i une approximation $[k, S(k), \dots, S^{i-1}(k)|l_i]$ de la liste des entiers naturels consécutifs à partir de k :

$$\begin{array}{l}
 \text{LN}(k, l) \\
 \downarrow \\
 \text{LN}(S(k), l_1) \\
 \downarrow \\
 \text{LN}(S^2(k), l_2) \\
 \vdots \\
 \text{LN}(S^{i-1}(k), l_{i-1}) \\
 \downarrow \\
 \text{LN}(S^i(k), l_i) \\
 \vdots
 \end{array}
 \quad
 \begin{array}{l}
 \theta_1 = \left[\begin{array}{cc} x_1 & l \\ k & [k|l_1] \end{array} \right] \\
 \theta_2 = \left[\begin{array}{cc} x_2 & l_1 \\ S(k) & [S(k)|l_2] \end{array} \right] \\
 \vdots \\
 \theta_i = \left[\begin{array}{cc} x_i & l_{i-1} \\ S^{i-1}(k) & [S^{i-1}(k)|l_i] \end{array} \right]
 \end{array}
 \quad (2)$$

Cependant, toutes les dérivations infinies ne correspondent pas nécessairement à la construction d'un tel objet : certaines dérivations, appelées dans cet article *dérivations infinies sur le domaine des termes finis*, sont infinies et ne « calculent » que des termes finis. De telles dérivations ne sont pas pour autant dénuées d'une sémantique et ne méritent pas, comme c'est l'usage, d'être reléguées dans une classe de dérivations qui seraient engendrées à partir de « mauvais programmes » : par exemple, il est possible d'écrire un programme logique donnant lieu à des dérivations infinies qui décrivent des séquences infinies d'« états » qui modélisent certains processus (par exemple, le célèbre problème du dîner des philosophes, ou plus généralement certains problèmes

dont le graphe d'états est soit infini soit cyclique). C'est précisément cette classe de dérivations qui est étudiée dans cet article.

En programmation logique, il est courant de comprendre les dérivations comme des « preuves qui calculent » et c'est bien sûr l'aspect calculatoire de telles preuves qui permet l'utilisation de programmes logiques dans la résolution de problèmes. C'est aussi cet aspect calculatoire qui est considéré dans la littérature consacrée aux dérivations infinies [ABD 84, ABD 85, GOL 88, HEI 92, JAF 86b, LLO 87] : une dérivation infinie est alors vue comme un processus de calcul à l'infini d'un objet infini. En effet, les sémantiques proposées, qui reposent, pour la plupart, sur la notion de plus grand point fixe, cherchent toutes à capturer la notion d'« atome (infini) calculable à l'infini ». Dans les deux principales approches existantes, l'univers du discours est complété de manière à disposer de termes infinis : dans [ABD 84, ABD 85, JAF 86b, LLO 87], une notion de distance entre termes permet de définir une complétion métrique de l'univers de Herbrand – dans [GOL 88], c'est une complétion par idéaux qui est envisagée. Hélas, ces approches sont toutes incomplètes (i.e., il existe des atomes infinis dans la dénotation de certains programmes qui ne sont pas « calculables à l'infini ») : la complétude de l'approche métrique ne s'obtient qu'en rendant possible la présence de termes infinis dans les requêtes¹ puisqu'elle est alors exprimée par :

$$\begin{aligned} & A \text{ (atome éventuellement infini)} \in \text{gfp}(TP) \\ \Leftrightarrow & \text{ il existe une dérivation équitable à partir de } A \end{aligned}$$

tandis que l'approche par idéaux, outre le fait qu'elle restreint la classe des programmes envisagés, donne seulement une sémantique pour une classe de dérivations infinies caractérisée en termes d'objets minimaux. Il semble qu'un des obstacles à la complétude de ces approches provienne de la difficulté à prendre en compte les dérivations infinies sur le domaine des termes finis : la construction d'un plus grand point fixe ne reflète pas la manière avec laquelle les termes infinis sont construits lors d'une dérivation infinie. Par exemple, si l'on considère l'approche métrique, la dénotation du programme :

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

contient $p(f^\omega)$ même si le terme f^ω ne peut être construit dans une dérivation infinie (d'où la nécessité d'autoriser la présence de termes infinis dans les requêtes pour obtenir la complétude).

La définition d'une sémantique par plus grand point fixe revient à identifier programmes logiques et définitions co-inductives et, puisque les dérivations sont avant

1. Cette condition, explicitement mentionnée dans [POD 99], ne correspond pas à la sémantique opérationnelle « standard ». D'autre part, la présence de termes infinis dans les requêtes nécessite d'utiliser une version modifiée de l'algorithme d'unification. Enfin, le problème de la représentation des termes infinis dans le langage (fini) des requêtes se pose, même s'il peut dans certains cas être résolu en adoptant une représentation implicite (par exemple avec des contraintes) des termes infinis.

tout des preuves, il semble pertinent d'envisager cette correspondance plus en profondeur en comparant les dérivations infinies avec les preuves par co-induction. Pour ce faire, les clauses vont être considérées comme des signatures fonctionnelles de constructeurs de preuves. Cette comparaison, effectuée dans la section 2, va permettre de mettre en évidence les phénomènes d'incomplétude observés dans les approches existantes et montrera que pour la classe des dérivations infinies sur le domaine des termes finis, il existe une correspondance directe entre dérivations (SLD) et preuves (par co-induction). Aussi, une sémantique valide et complète sera définie dans la section 3 (les preuves complètes et détaillées des résultats présentés figurent dans un rapport de recherche [JAU 98]). Dans cet article, nous nous intéressons donc à la cohérence interne entre deux théories (programmation logique et définitions co-inductives) : les sections 2 et 3 sont indépendantes (la première présente une discussion et la seconde expose les résultats « techniques » obtenus). Nous supposons le lecteur familier avec les concepts classiques de la programmation logique [LLO 87], de la \mathcal{C} -sémantique [FAL 93, FAL 89], des définitions (co-)inductives [ACZ 77] et de l'isomorphisme de Curry-Howard [LAL 93].

2. (Co-)Induction et SLD-résolution

Rappelons qu'un ensemble peut être défini (co-)inductivement à l'aide d'un ensemble Φ de *constructeurs* de la forme $e \leftarrow E$ où e est l'élément construit à partir d'un ensemble d'objets E . Dans le cas d'une *définition inductive*, l'ensemble ainsi défini, noté $\text{Ind}(\Phi)$, correspond à l'intersection de tous les ensembles Φ -clos (i.e., les ensembles A vérifiant pour chaque constructeur $e \leftarrow E$ de Φ , $E \subseteq A \Rightarrow e \in A$) et contient tous les éléments obtenus en appliquant un nombre de fois fini les constructeurs, tandis que dans le cas d'une *définition co-inductive*, l'ensemble défini, noté $\text{CoInd}(\Phi)$, correspond à l'union de tous les ensembles Φ -denses (i.e., les ensembles A tels que pour tout $a \in A$ il existe un ensemble $E \subseteq A$ tel que $a \leftarrow E$ est un constructeur de Φ) et contient tous les éléments obtenus en appliquant un nombre de fois éventuellement infini les constructeurs. De manière équivalente, ces ensembles peuvent être définis à partir d'un opérateur monotone T_Φ défini à partir de Φ par :

$$\mathcal{B} = \bigcup_{e \leftarrow E \in \Phi} \{\{e\} \cup E\} \quad T_\Phi(A) = \{e \in \mathcal{B} \mid \exists e \leftarrow E \in \Phi \quad E \subseteq A\} \quad (4)$$

On a alors $\text{Ind}(\Phi) = \bigcap_{T_\Phi(A) \subseteq A} A$ et $\text{CoInd}(\Phi) = \bigcup_{A \subseteq T_\Phi(A)} A$. Ces deux ensembles correspondent respectivement au plus petit point fixe ($\text{lfp}(T_\Phi)$) et au plus grand point fixe ($\text{gfp}(T_\Phi)$) de T_Φ . Enfin, ces ensembles peuvent aussi être définis à l'aide de la notion d'arbre de preuve : un *arbre de preuve* de x pour Φ est un arbre fini ou infini de racine x tel que pour tout noeud z de l'arbre admettant z_1, \dots, z_q pour fils, $z \leftarrow z_1, \dots, z_q \in \Phi$ (en particulier, si z est une feuille, alors $z \leftarrow \in \Phi$). On a alors :

$$\begin{aligned} \text{Ind}(\Phi) &= \{x \mid x \text{ est racine d'un arbre de preuve fini pour } \Phi \\ &\quad \text{(lorsque pour chaque constructeur } e \leftarrow E \in \Phi, E \text{ est fini)}\} \\ \text{CoInd}(\Phi) &= \{x \mid x \text{ est racine d'un arbre de preuve fini ou infini pour } \Phi\} \end{aligned}$$

Définitions inductives et programmes logiques présentent de nombreuses similitudes. Un programme défini P peut être vu comme un ensemble de constructeurs $[P]$ correspondant aux instances fermées (i.e., sans variables) des clauses de P . L'opérateur $T_{[P]}$ associé à cet ensemble de constructeurs, comme indiqué en (4), correspond exactement à l'opérateur de « conséquence immédiate » (traditionnellement noté T_P). D'autre part, une interprétation de Herbrand I est un modèle de P ssi $T_P(I) \subseteq I$, ou en d'autres termes, ssi I est $T_{[P]}$ -clos. Enfin, le plus petit modèle de Herbrand \mathcal{M}_P est défini comme l'intersection de tous les modèles de Herbrand ce qui, en termes de définitions inductives, revient à définir la dénotation de P par l'ensemble $\text{Ind}([P]) = \text{lfp}(T_{[P]})^2$. Aussi, plutôt qu'un simple outil technique, les définitions inductives constituent une alternative « naturelle » au paradigme « *logic programs as first-order theories* ». C'est cette idée qui est défendue dans [PAU 89, HUE 90] et utilisée avec profit dans [DER 93]. Cette lecture sémantique de la programmation logique permet de considérer un programme défini, non plus comme une théorie du premier ordre, mais comme la définition d'une « nouvelle logique ». En adoptant ce point de vue, la dénotation d'un programme correspond à l'ensemble des théorèmes qu'il est possible de « dériver » dans cette logique. Cette approche conduit donc à identifier un programme à une définition inductive ou co-inductive, selon que l'on considère ou non les dérivations infinies. Malheureusement, cette identification n'est plus aussi fructueuse lors de la définition d'une sémantique pour les dérivations infinies. En effet, envisageons à présent plus en profondeur le paradigme « *logic programs as co-inductive definitions* ». Pour cela, considérons les clauses C de la forme $A \leftarrow A_1, \dots, A_n$, en suivant l'isomorphisme de Curry-Howard, comme des signatures fonctionnelles de constructeurs de preuve (qui associent à partir des preuves π_{A_i} des A_i une preuve π_A de A , A correspond alors au type du λ -terme π_A) et comparons les preuves représentées par les dérivations SLD avec celles représentées par des λ -termes. Avant de mener cette comparaison, rappelons qu'un λ -terme de preuve établissant une proposition ϕ peut être défini de manière récursive, et donc utiliser récursivement la preuve de ϕ , sous certaines conditions. Il s'agit de la notion de « preuve gardée » introduite par T. Coquand dans le contexte de la théorie des types :

[COQ 94] « *In order to establish that a proposition ϕ follows from other propositions ϕ_1, \dots, ϕ_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.* »

Cette condition syntaxique garantit le caractère « productif » d'une classe de λ -termes. Puisqu'à chaque transition d'une dérivation, une clause (i.e., un constructeur) est appliquée, les dérivations peuvent être vues comme des preuves « gardées ». Voyons cela sur l'exemple typique du programme de la figure 1 permettant de caractériser les chemins d'un graphe orienté. L'existence d'un cycle dans ce graphe rend

2. De plus, on a $\text{lfp}(T_{[P]}) = T_{[P]}^{\uparrow\omega}$ car le corps de chaque clause est fini, et donc, en termes de définitions inductives, $[P]$ est finitaire.

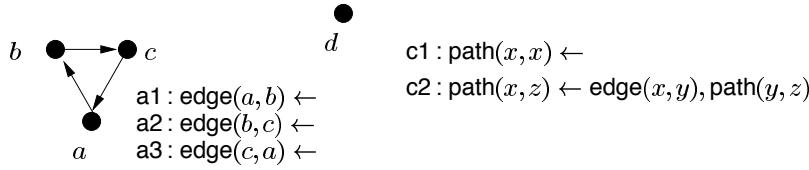


Figure 1. *Chemin entre deux sommets d'un graphe orienté*

possible la construction d'une dérivation infinie à partir de la requête $\text{path}(s_1, s_2)$ où s_1 est un sommet apparaissant sur un cycle et s_2 est un sommet quelconque. Par exemple, on a :

$$\begin{aligned}
 \text{path}(a, x) &\xrightarrow{c2} \text{edge}(a, y_1), \text{path}(y_1, x) \xrightarrow{a1} \text{path}(b, x) \\
 &\xrightarrow{c2} \text{edge}(b, y_2), \text{path}(y_2, x) \xrightarrow{a2} \text{path}(c, x) \\
 &\xrightarrow{c2} \text{edge}(c, y_3), \text{path}(y_3, x) \xrightarrow{a3} \text{path}(a, x) \rightarrow \dots
 \end{aligned} \tag{5}$$

Il est possible de construire le λ -terme π_{ax} de type $\forall x \text{path}(a, x)$ (qui vérifie bien la condition de garde puisque l'appel récursif à π_{ax} est bien « gardé » par c2) correspondant à la dérivation (5) :

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

Les six premières transitions de la dérivation correspondent aux six applications des constructeurs, tandis que l'appel récursif correspond à la suite de la dérivation³. Un tel λ -terme peut être vu comme la définition récursive de la suite des clauses appliquées dans la dérivation et *vice versa*. Néanmoins, il existe des dérivation infinies pour lesquelles cette suite de clauses n'est pas toujours calculable : considérons par exemple le programme $P = \{p(x) \leftarrow p(f(x)); p(x) \leftarrow p(g(x))\}$. Si l'on se restreint à n'utiliser que la première clause, notée C_1 , on obtient la dérivation⁴ :

$$p(z) \rightarrow \underbrace{\left[\begin{array}{c} x_1 \\ f(x_1) \end{array} \right] p(x_1) \rightarrow \dots \rightarrow \left[\begin{array}{c} x_i \\ f^i(x_i) \end{array} \right] p(x_i) \rightarrow \dots}_{\text{preuve de } \forall x_1 p(f(x_1))}$$

Ici encore, le λ -terme de preuve de type $\forall z p(z)$ défini par :

$$\pi_p := \lambda z. C_1 \left(z, \pi_p \left(\left[\begin{array}{c} z \\ f(z) \end{array} \right] z \right) \right)$$

3. Un phénomène « rassurant » peut cependant être observé si l'on rajoute un argument au prédicat **path** correspondant à la longueur du chemin qui sépare les deux sommets considérés : dans ce cas, la longueur du chemin séparant le sommet a du sommet d est S^w (la longueur du chemin considéré correspond à la longueur de la preuve de l'existence d'un chemin).

4. Cette dérivation ne « calcule » pas de terme infini (la suite d'applications des unificateurs utilisés sur la requête initiale ne conduit pas à la construction d'un terme infini) même si la suite des requêtes résiduelles « tend » vers $p(f^w)$.

correspond bien à la dérivation ci-dessus (la première transition correspond à l'application de C_1 tandis que l'appel récursif correspond aux suivantes, c'est à dire à une preuve de $\forall z p(f(z))$). Par contre, si l'on utilise les deux clauses du programme, à chaque étape de la dérivation ces deux clauses peuvent être utilisées. La fonction qui associe à chaque transition la clause utilisée n'est alors pas forcément calculable et le λ -terme n'admet pas dans ce cas de représentation finie. Quoi qu'il en soit, il existe bien une correspondance directe entre dérivations et λ -termes lorsqu'aucun terme infini apparaît : les dérivations infinies qui ne construisent aucun terme infini correspondent à des preuves par co-induction. En présence de termes infinis, cette correspondance n'est plus immédiate : reprenons l'exemple du programme (1), vu comme une définition co-inductive. Il est alors possible de prouver, par co-induction, que la liste, notée $\text{from}(k)$, des entiers consécutifs à partir de k vérifie bien $\text{LN}(k, \text{from}(k))$. Or, pour construire le λ -terme correspondant, il est nécessaire de disposer de cette liste définie⁵ par :

$$\text{from} := \lambda n. \text{cons}(n, \text{from}(S(n)))$$

Le λ -terme s'écrit alors :

$$\begin{aligned} \pi_{LN} := \lambda n. \text{eq_ind}(\quad & \text{cons}(n, \text{from}(S(n))), \\ & \lambda u. \text{LN}(n, u), \\ & \text{C}(n, \text{from}(S(n)), \pi_{LN}(S(n))), \\ & \text{from}(n), \\ & \text{from_unfold}(n)) \end{aligned}$$

où from_unfold est la preuve de la proposition :

$$\forall n \text{ from}(n) = \text{cons}(n, \text{from}(S(n)))$$

et où eq_ind correspond au schéma d'élimination de l'égalité (i.e., égalité à la Leibniz) et associe une preuve de $P(y)$ à partir d'un élément x , d'un prédicat P , d'une preuve de $P(x)$, d'un élément y et d'une preuve de $x = y$. La définition de ce terme est bien gardée puisque l'appel récursif à π_{LN} est protégé par le constructeur C . On voit donc bien sur cet exemple que le λ -terme ne reflète pas le processus de construction de la liste infinie (dérivation (2)) : prouver par co-induction $\text{LN}(k, \text{from}(k))$ revient à appliquer une infinité de fois le constructeur de preuve C en utilisant à chaque appel récursif la propriété from_unfold .

3. Dérivations infinies sur le domaine des termes finis

3.1. \mathcal{C} -sémantique appliquée aux dérivations infinies

Nous nous plaçons à présent dans le cadre de la \mathcal{C} -sémantique [FAL 93, FAL 89] pour étudier les dérivations infinies. Toutefois, dans ce qui suit, nous ne basons pas

5. La définition de from est gardée puisque from est protégé par le constructeur cons . L'utilisation de termes infinis dans l'univers du discours revient à considérer une définition co-inductive pour les termes (les symboles de fonction sont les constructeurs).

la sémantique proposée sur la notion de \mathcal{C} -vérité : la dénotation d'un programme est définie en termes de définitions co-inductives. Pour ce faire, un programme P est vu comme l'ensemble des constructeurs défini par :

$$\llbracket P \rrbracket = \{\theta C \mid C \in P, \theta : X \rightarrow T_\Sigma[X]\}$$

où X , Σ et $T_\Sigma[X]$ dénotent respectivement un ensemble de variables, un ensemble de fonctions muni d'une fonction d'arité et l'ensemble des termes finis construits sur $X \cup \Sigma$. D'autre part, dans ce qui suit, étant donnée une substitution θ , $\text{dom}(\theta)$ (resp. $\text{range}(\theta)$) dénote le domaine (resp. l'image) de θ et on écrira $E_1 \leq E_2$ lorsque pour une substitution θ , $\theta E_1 = E_2$. A cet ensemble de constructeurs peut être associé l'opérateur $T_{\llbracket P \rrbracket}$, comme indiqué en (4). Les résultats prouvés dans [FAL 93, FAL 89] donnent une sémantique pour les réfutations en terme de plus petit point fixe de $T_{\llbracket P \rrbracket}$:

$$S_P^{\mathcal{C}} = \{A \text{ (atome fini)} \mid A \xrightarrow{*,\theta} \square \text{ et } \theta A = A\} = \text{lfp}(T_{\llbracket P \rrbracket}) = \text{Ind}(\llbracket P \rrbracket)$$

Nous nous proposons ici d'étudier la contre-partie opérationnelle du plus grand point fixe de $T_{\llbracket P \rrbracket}$. Il s'agit bien sûr des dérivations infinies sur le domaine des termes finis. La définition suivante donne une caractérisation formelle des dérivations que nous considérons ici.

Definition 1 Une SLD-preuve sur le domaine des termes finis est soit une réfutation, soit une dérivation infinie équitable⁶ : $R_0 \xrightarrow{C_1, \theta_1} R_1 \rightarrow \dots \rightarrow R_{i-1} \xrightarrow{C_i, \theta_i} R_i \rightarrow \dots$ telle que $\forall k \geq 0 \quad \exists p > k \quad \forall q \geq p \quad \theta_q \dots \theta_p \dots \theta_{k+1} R_k \approx \theta_p \dots \theta_{k+1} R_k$.

\approx dénote la relation d'équivalence au renommage près ($E_1 \approx E_2 \Leftrightarrow (E_1 \leq E_2 \wedge E_2 \leq E_1)$). L'information calculée doit donc l'être en un nombre fini d'étapes (i.e., pour chaque requête R apparaissant dans la dérivation, il existe un certain rang à partir duquel la suite des instanciations de R par les unificateurs est stationnaire). De manière équivalente, on peut montrer qu'une dérivation infinie :

$$R_0 \xrightarrow{C_1, \theta_1} R_1 \rightarrow \dots \rightarrow R_{i-1} \xrightarrow{C_i, \theta_i} R_i \rightarrow \dots$$

est une SLD-preuve infinie sur le domaine des termes finis si et seulement si :

$$\forall i \geq 0 \quad \exists R \quad \forall n \geq i + 1 \quad \theta_n \theta_{n-1} \dots \theta_{i+1} R_i \leq R \quad (6)$$

où R est une requête (dans laquelle aucun terme infini ne peut apparaître).

La propriété de validité s'obtient assez facilement en associant un n -uplet d'arbres de preuve à toute SLD-preuve sur le domaine des termes finis. Elle permet de caractériser un certain rang à partir duquel la requête initiale instanciée contient des atomes appartenant à la dénotation du programme.

6. Rappelons qu'une dérivation est *équitable* si elle échoue ou si pour tout atome A y apparaissant, A ou l'un de ses résidus est sélectionné en un nombre fini d'étapes. D'autre part, rappelons que les *conditions de renommage* des dérivations envisagées dans cet article garantissent que : $\forall i \geq 1, \text{var}(C_i) \cap (\cup_{j < i} \text{var}(C_j) \cup \text{var}(R_0)) = \emptyset$

Proposition 1 (Validité) *Si il existe une SLD-preuve sur le domaine des termes finis avec le programme $P : A_1, \dots, A_n \xrightarrow{C_1, \theta_1} R_1 \rightarrow \dots \rightarrow R_{i-1} \xrightarrow{C_i, \theta_i} R_i \rightarrow \dots$ alors il existe $k \geq 0$ tel que pour tout i ($1 \leq i \leq n$) $\theta_k \dots \theta_1 A_i \in \text{Colnd}(\lceil P \rceil)$.*

PREUVE. On construit une suite de n -uplets d'arbres de preuve partiels (i.e., dont les feuilles ne correspondent pas nécessairement à une clause unité) :

$$((T_1^1, \dots, T_1^n), \dots, (T_i^1, \dots, T_i^n), \dots)$$

pour $\lceil P \rceil$ telle que $\theta_i \dots \theta_1 A_j$ est racine de T_i^j ($1 \leq j \leq n$) et telle que chaque atome apparaissant dans R_i soit une feuille dans T_i^j pour un entier j . Puisque la dérivation de départ est une SLD-preuve sur le domaine des termes finis, il existe un entier k à partir duquel chaque T_k^j est un arbre de preuve de A_j pour $\lceil P \rceil$. ◀

La propriété de complétude s'obtient en deux étapes. La dénotation d'un programme étant définie en termes de définitions co-inductives, nous montrons tout d'abord comment à partir d'un arbre de preuve, il est possible de construire une dérivation.

Lemme 1 *Soit P un programme défini et A un atome. Si $A \in \text{Colnd}(P)$, alors il existe une SLD-preuve sur le domaine des termes finis à partir de A avec le programme P telle qu'à chaque étape i , l'unificateur θ_i mis en jeu est un renommage, injectif sur son domaine, de toutes les variables apparaissant dans la tête de la clause utilisée.*

PREUVE. La preuve de ce lemme repose sur un parcours de l'arbre de preuve durant lequel la dérivation est construite. Ce parcours s'effectue en largeur d'abord afin de garantir l'équité de la dérivation construite. Pour rester fidèle à nos travaux antérieurs [JAU 99], cette construction a été effectuée en explicitant complètement les aspects liés au renommage. Si $A \in \text{Colnd}(P)$, alors A est racine d'un arbre de preuve T pour P . Les noeuds de cet arbre peuvent être indicés par des éléments de \mathbb{N}^* : l'indice d'un noeud correspond aux étiquettes rencontrées sur le chemin qui mène de la racine à ce noeud ; les arcs sont étiquetés de gauche à droite par des entiers successifs à partir de 1 (ε désigne le mot vide). Le parcours en largeur de T fournit la liste de noeuds \mathcal{L} . Par définition, pour chaque noeud $A_{\bar{i}}$ il existe une clause $C_{T, \bar{i}} \in P$ qui s'écrit $A_{\bar{i}} \leftarrow A_{\bar{i}1}, \dots, A_{\bar{i}n_{\bar{i}}}$. Les indices de T peuvent être ordonnés par : $\bar{i} \prec \bar{j}$ ssi $A_{\bar{i}}$ apparaît avant $A_{\bar{j}}$ dans \mathcal{L} . $Z_T = \cup \text{var}(C_{T, \bar{i}})$ est l'ensemble des variables apparaissant dans les noeuds de T . On montre qu'étant donnés une clause $C_{T, \bar{i}} \in P$, une substitution de renommage $r_0^{\bar{i}}$, telle que $\text{range}(r_0^{\bar{i}}) \cap \text{var}(C_{T, \bar{i}}) = \emptyset$, et un ensemble de variables $Z_{\bar{i}}$, il existe une substitution $\theta_{\bar{i}}$, une clause $C_{\bar{i}}$ et une substitution de renommage $r_1^{\bar{i}} = r_0^{\bar{i}} r_1^{\bar{i}}$ tels que⁷ :

$$\begin{aligned} \text{var}(C_{\bar{i}}) \cap (\text{var}(r_0^{\bar{i}} C_{T, \bar{i}}) \cup Z_{\bar{i}}) &= \emptyset & r_1^{\bar{i}} C_{T, \bar{i}}^- &= \theta_{\bar{i}} C_{\bar{i}}^- \\ \text{dom}(\theta_{\bar{i}}) &= \text{var}(C_{\bar{i}}^+) & \text{range}(r_1^{\bar{i}}) &= \text{var}(C_{\bar{i}}^-) \setminus \text{var}(C_{\bar{i}}^+) \end{aligned}$$

7. C^+ (resp. C^-) dénote la tête (resp. le corps) de la clause C

où $\theta_{\bar{\tau}}$ est un renommage idempotent qui est un unificateur principal de $C_{\bar{\tau}}^+$ et $r_0^{\bar{\tau}}C_{T,\bar{\tau}}^+$. A partir de \mathcal{L} , on définit alors la suite de transitions :

$$t_\varepsilon = \mathcal{T}(C_{T,\varepsilon}, s_{id}, Z_T), \quad t_{\bar{\tau}k} = \mathcal{T}(C_{T,\bar{\tau}k}, r_0^{\bar{\tau}k}, Z_{\bar{\tau}k}) \quad \left\{ \begin{array}{l} 1 \leq k \leq n_{\bar{\tau}} \\ r_0^{\bar{\tau}k} = r_1^{\bar{\tau}} \\ Z_{\bar{\tau}k} = Z_T \cup \bigcup_{\bar{\tau} \prec \bar{\tau}k} \text{var}(C_{\bar{\tau}}) \end{array} \right.$$

où $\mathcal{T}(C_{T,\bar{\tau}}, r_0^{\bar{\tau}}, Z_{\bar{\tau}})$ dénote la transition $r_0^{\bar{\tau}}C_{T,\bar{\tau}}^+ \xrightarrow{C_{\bar{\tau}}, \theta_{\bar{\tau}}, Z_{\bar{\tau}}} \theta_{\bar{\tau}}C_{\bar{\tau}}^-$ et où s_{id} est la substitution identité. Cette définition est correcte, puisque l'on montre que $\forall \bar{\tau}, \text{range}(r_0^{\bar{\tau}}) \cap \text{var}(C_{T,\bar{\tau}}) = \emptyset$, et vérifie bien les propriétés désirées (voir [JAU 98]). ◀

Ce lemme permet de donner une caractérisation d'une classe bien particulière de dérivations : puisqu'à chaque transition l'unificateur utilisé est un renommage, ces dérivations ne « calculent » rien. $\text{Colnd}(P)$ correspond à l'ensemble défini co-inductivement à partir des (variantes de) clauses de P non instanciées vues comme des constructeurs. Aussi, puisque nous définissons la dénotation d'un programme P par l'ensemble $\text{Colnd}(\lceil P \rceil)$, le lemme 1 nous permettra seulement d'obtenir des dérivations à partir du programme $\lceil P \rceil$. C'est pourquoi, dans un deuxième temps, nous montrons le lemme suivant qui montre comment obtenir une dérivation avec le programme P à partir d'une dérivation avec le programme $\lceil P \rceil$. A partir de maintenant, afin d'éviter toute confusion, on indicera la relation de transition \rightarrow par le programme utilisé.

Lemme 2 (Program lifting lemma) *Si il existe une SLD-preuve :*

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

telle que pour tout $i \geq 1$, θ_i est une substitution de renommage idempotente et injective sur son domaine vérifiant $\text{dom}(\theta_i) = \text{var}(C_i^+)$, alors il existe une SLD-preuve sur le domaine des termes finis : $A_0 \xrightarrow{C_{P,1}, \sigma_1}_{\lceil P \rceil} R'_1 \rightarrow_P \cdots \rightarrow_P R'_{i-1} \xrightarrow{C_{P,i}, \sigma_i}_{\lceil P \rceil} R'_i \rightarrow_P \cdots$ telle que pour tout $i \geq 1$, $\sigma_i A_0 = A_0$, $C_i = \mu_i C_{P,i}$ et $R_i = \rho_i R'_i$ où ρ_i est la restriction de $\theta_i \mu_i \theta_{i-1} \mu_{i-1} \cdots \theta_1 \mu_1$ aux variables de R'_i .

PREUVE. La preuve est assez technique : elle requiert un calcul sur les substitutions (unificateurs et renommages). On construit un ensemble $\{C_{P,1}, \dots, C_{P,i}, \dots\}$ de variantes de clauses de P tel que chaque clause $C_{P,i}$ vérifie $C_i = \mu_i C_{P,i}$ où μ_i est une substitution idempotente telle que $\text{dom}(\mu_i) = \text{var}(C_{P,i})$ et

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

• *Pour la première transition.* Par définition $\theta_1 A_0 = \theta_1 C_1^+ = \theta_1 \mu_1 C_{P,1}^+$. De plus, puisque $\text{var}(C_{P,1}) \cap \text{var}(A_0) = \emptyset$, et $\text{var}(C_1) \cap \text{var}(A_0) = \emptyset$, on a $\theta_1 \mu_1 A_0 = A_0 = \theta_1 \mu_1 C_{P,1}^+$. On montre alors que la restriction σ_1 de $\theta_1 \mu_1$ aux variables de $\text{var}(C_{P,1}^+)$ est un unificateur principal de A_0 et $C_{P,1}^+$. On a alors la transition $A_0 \xrightarrow{C_{P,1}, \sigma_1}_{\lceil P \rceil} R'_1$. $\sigma_1 A_0 = A_0$ découle de $\theta_1 \mu_1 A_0 = A_0$. Soit ρ_1 la restriction de $\theta_1 \mu_1$ à $\text{var}(R'_1)$ et

montrons que $\rho_1 R'_1 = \rho_1 \sigma_1 C_{P,1}^- = \theta_1 \mu_1 C_{P,1}^- = \theta_1 C_1^- = R_1$. Soit $v \in \text{var}(C_{P,1}^-)$, deux cas sont possibles. Si $v \in \text{var}(C_{P,1}^+)$, alors $\sigma_1 v = \theta_1 \mu_1 v$ et on peut conclure puisque $\theta_1 \mu_1 \theta_1 \mu_1 v = \theta_1 \mu_1 v$. Sinon, si $v \notin \text{var}(C_{P,1}^+)$, alors on a $\rho_1 \sigma_1 v = \rho_1 v = \theta_1 \mu_1 v$ et on peut aussi conclure.

• *Pour la i -ème transition.* Si A est l'atome sélectionné dans R_{i-1} à la position k , alors il existe un atome A' apparaissant à la position k dans R'_{i-1} tel que $A = \rho_{i-1} A'$. On a alors $\theta_i \rho_{i-1} A' = \theta_i \mu_i C_{P,i}^+$. De $\text{dom}(\rho_{i-1}) \subseteq \text{var}(R'_{i-1})$ et $\text{var}(R'_{i-1}) \subseteq (\cup_{1 \leq j < i} \text{var}(C_{P,j}) \cup \text{var}(A_0))$, il vient $\rho_{i-1} C_{P,i} = C_{P,i}$ et donc on a $\theta_i \rho_{i-1} A' = \theta_i \mu_i \rho_{i-1} C_{P,i}^+$. De plus, $\text{dom}(\mu_i) = \text{var}(C_{P,i})$ et on a $\mu_i A = A$. Aussi, $\theta_i \mu_i \rho_{i-1} A' = \theta_i \mu_i \rho_{i-1} C_{P,i}^+$, et puisque $\theta_i \mu_i \dots \theta_1 \mu_1 A_0 = A_0$, il existe un unificateur principal σ_i de A' et $C_{P,i}^+$ tel que $\sigma_i A_0 = A_0$. Pour une substitution η_i , on a donc $\eta_i \sigma_i = \theta_i \mu_i \rho_{i-1}$. On obtient alors la transition $R'_{i-1} \xrightarrow{C_{P,i}, \sigma_i} R'_i$. Puisque σ_i est idempotente, on a $\theta_i \mu_i \rho_{i-1} \sigma_i = \eta_i \sigma_i \sigma_i = \eta_i \sigma_i = \theta_i \mu_i \rho_{i-1}$ et il vient $\theta_i \mu_i \rho_{i-1} R'_i = R_i$. Pour terminer, on montre par induction que $\forall n \geq i + 1, \rho_n \sigma_n \sigma_{n-1} \dots \sigma_{i+1} R'_i = R_i$. Aussi, puisque R_i ne contient que des atomes finis et puisque $\forall n \geq i + 1$, on a $\sigma_n \sigma_{n-1} \dots \sigma_{i+1} R'_i \leq R_i$, la condition (6) établit que la dérivation obtenue est bien une SLD-preuve sur le domaine des termes finis. ◀

Tout comme le lemme de généralisation « standard » (*lifting lemma*) pour les requêtes, ce lemme permet de « généraliser » sur les programmes et permet, avec le lemme 1, d'obtenir directement la propriété de complétude.

Proposition 2 (Complétude) *Etant donné un programme P , si $A \in \text{Colnd}([P])$, alors il existe une SLD-preuve sur le domaine des termes finis à partir de A avec le programme $P : A \xrightarrow{C_{P,1}, \sigma_1} R'_1 \rightarrow_P \dots \rightarrow_P R'_{i-1} \xrightarrow{C_{P,i}, \sigma_i} R'_i \rightarrow_P \dots$ telle que pour tout $i \geq 1, \sigma_i A = A$.*

La sémantique qui vient d'être définie repose sur la notion d'arbre de preuve. La propriété de complétude a été obtenue en considérant l'arbre de preuve associé aux éléments de $\text{Colnd}([P])$, en « linéarisant » cet arbre pour obtenir une dérivation avec le programme $[P]$, puis en « généralisant » sur $[P]$ pour obtenir une dérivation avec le programme P .

3.2. Un cas particulier

Les dérivations que nous avons considérées jusqu'à présent satisfont les conditions mentionnées dans la définition 1. Dans ces dérivations, un « calcul » peut être effectué (i.e., les unificateurs peuvent instancier la requête initiale). Il est possible de restreindre encore cette classe de dérivations pour ne considérer que l'aspect « preuve » des dérivations.

Définition 2 *Une SLD-preuve directe est une SLD-preuve de la forme :*

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

telle que pour tout $i \geq 1$, $\text{dom}(\theta_i) \subseteq \text{var}(C_i^+)$. En particulier, une SLD-réfutation directe est une SLD-preuve directe finie se terminant par la requête vide.

Pour cette classe de dérivations, il est possible de définir une sémantique en termes de modèles dont la définition diffère légèrement de celle utilisée pour la \mathcal{C} -sémantique dans [FAL 93, FAL 89]. Pour cela, la notion de \mathcal{C}^+ -vérité d'une clause est définie relativement à une \mathcal{C} -interprétation I par : une clause $A \leftarrow B_1, \dots, B_q$ est \mathcal{C}^+ -vraie dans I si et seulement si pour toute substitution θ vérifiant $\text{dom}(\theta) \subseteq \text{var}(A)$, si les atomes θB_i ($1 \leq i \leq q$) sont \mathcal{C} -vrais dans I , alors θA est \mathcal{C} -vrai dans I . Il est alors possible de définir une sémantique de manière classique pour les SLD-réfutations directes : plus petit \mathcal{C}^+ -modèle, validité, complétude ... (voir [JAU 98]). Si l'on se place dans le cadre des définitions (co-)inductives, cette sémantique peut être directement obtenue en identifiant un programme P à l'ensemble de constructeurs défini par :

$$[P]^+ = \{\theta C \mid C \in P, \theta : X \rightarrow T_\Sigma[X], \text{dom}(\theta) \subseteq \text{var}(C^+)\}$$

auquel peut être associé un opérateur $T_{[P]^+}$ sur les \mathcal{C} -interprétations, comme indiqué en (4). Ce cadre est adéquat pour étudier la sémantique des programmes dont les clauses ne contiennent pas de variables existentielles (i.e., telles que chaque variable apparaissant dans le corps d'une clause apparaît aussi dans sa tête) puisqu'alors on a $[P]^+ = [P]$. De tels programmes vérifient de plus l'égalité $\text{gfp}(T_{[P]^+}) = T_{[P]^+}^{\downarrow\omega}$ et font partie des programmes canoniques étudiés dans [JAF 86a]. Les résultats concernant les SLD-preuves directes infinies sont similaires à ceux présentés dans la sous-section 3.1.

Proposition 3 *Soit P un programme défini et A_0 un atome. Il existe une SLD-preuve directe : $A_0 \xrightarrow{C_1, \theta_1}_P R_1 \rightarrow_P \dots \rightarrow_P R_{i-1} \xrightarrow{C_i, \theta_i}_P R_i \rightarrow_P \dots$ si et seulement si $A_0 \in \text{Colnd}([P]^+)$.*

Toutefois, lorsque la dérivation infinie considérée n'est pas équitable, il est possible de caractériser un ensemble d'hypothèses suffisantes pour que l'atome « partiellement prouvé » par la dérivation appartienne à $\text{Colnd}([P]^+)$. Pour ce faire, nous définissons la requête résiduelle d'une dérivation :

$$R_0 \xrightarrow{C_1, \theta_1}_P R_1 \rightarrow_P \dots \rightarrow_P R_{i-1} \xrightarrow{C_i, \theta_i}_P R_i \rightarrow_P \dots$$

par $R_\infty = \bigcup_{p \geq 0} \bigcap_{p \leq n} R_n$. Signalons que même si la dérivation infinie est équitable, R_∞ n'est pas nécessairement l'ensemble vide (par exemple, pour la dérivation infinie équitable que l'on peut obtenir à partir de la requête $p(z)$ avec le programme (3), on a $R_\infty = \{p(z)\}$). La proposition 3 ne doit donc pas être vue comme un corollaire de la proposition suivante mais comme une autre façon d'interpréter une dérivation infinie qui ne construit pas de termes infinis : ces dérivations sont vues ici comme des preuves partielles, et expriment donc une implication. Cette proposition peut être comparée au lemme standard, établissant l'implication $P \models R \Rightarrow P \models \theta R_0$ à partir de la dérivation finie $R_0 \xrightarrow{\theta, *}_P R$.

Proposition 4 *Soit P un programme défini et A_0 un atome. Si il existe une dérivation infinie (non nécessairement équitable):*

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

telle que pour tout $i > 0$, $\text{dom}(\theta_i) = \text{var}(C_i^+)$, alors :

$$R_\infty \subseteq \text{Colnd}(\lceil P \rceil^+) \Rightarrow A_0 \in \text{Colnd}(\lceil P \rceil^+)$$

PREUVE. Supposons que $\cup_{p \geq 0} \cap_{p \leq n} R_n \subseteq \text{Colnd}(\lceil P \rceil^+)$ et montrons que $A_0 \in \text{Colnd}(\lceil P \rceil^+)$. Pour cela, il suffit d'exhiber un arbre de preuve T de A_0 pour $\lceil P \rceil^+$. On définit la suite T_1, \dots, T_i, \dots d'arbres de preuve partiels (i.e., dont les feuilles ne correspondent pas nécessairement à une clause unité) telle que chaque atome de R_i soit une feuille de T_i . T_1 est obtenu en considérant la première transition: sa racine est $A_0 = \theta_1 A_0$ et a pour fils les feuilles qui sont dans $R_1 = \theta_1 C_1^-$. Montrons à présent comment obtenir T_n à partir de T_{n-1} . On sait que les atomes de R_{n-1} sont des feuilles de T_{n-1} . Soit A l'atome sélectionné dans R_{n-1} . Puisque $\text{dom}(\theta_n) \subseteq \text{var}(C_n^+)$, T_n est obtenu en ajoutant les atomes de $\theta_n C_n^-$ comme fils de A . Puisque⁸, $R_n = \theta_n R_{n-1}[k \leftarrow C_n^-] = R_{n-1}[k \leftarrow \theta_n C_n^-]$, T_n est un arbre de preuve partiel de A_0 pour $\lceil P \rceil^+$ tel que chaque atome de R_n est une feuille de T_n . En itérant ce processus, on obtient un arbre de preuve partiel T_∞ dont les feuilles sont soit des têtes de clauses unités de $\lceil P \rceil^+$ soit des atomes apparaissant dans R_∞ , qui sont, par hypothèse, dans $\text{Colnd}(\lceil P \rceil^+)$ et correspondent donc à des racines d'arbres de preuve pour $\lceil P \rceil^+$. Aussi, en ajoutant dans T_∞ ces arbres aux feuilles correspondantes, on obtient un arbre de preuve de A_0 pour $\lceil P \rceil^+$. ◀

4. Conclusion

Tandis que les résultats classiques de la programmation logique concernent la sémantique des calculs finis, certains programmes logiques peuvent donner lieu à des dérivations infinies pour lesquelles ces résultats ne s'appliquent pas. Dans la littérature consacrée aux SLD-dérivations infinies, il n'existe pas une « sémantique classique » mais plusieurs approches non équivalentes. Dans la plupart d'entre elles, la dénotation d'un programme défini P est obtenue en considérant le plus grand point fixe d'un opérateur associé à P . Hélas, lorsque l'univers du discours contient des éléments infinis, l'utilisation d'un plus grand point fixe conduit à définir une sémantique valide mais non complète (i.e., certains atomes dans la dénotation de P ne sont pas « constructibles » par une SLD-dérivation à partir de P). En effet, en présence d'éléments infinis, seul un sous-ensemble du plus grand point fixe de l'opérateur associé à un programme pourrait caractériser les objets infinis calculables à l'infini à partir de ce programme. D'ailleurs, l'approche développée par W.G. Golson [GOL 88]

8. $R[k \leftarrow C^-]$ dénote la requête R dans laquelle le k -ième atome est remplacé par C^- .

(complétion par idéaux) permet de caractériser les objets « calculés » par une certaine classe de dérivations en termes d'objets minimaux présents dans le plus grand point fixe de l'opérateur T_P : tous les éléments de cet ensemble ne sont pas considérés. Cependant, de manière intuitive, c'est l'approche développée par G. Levi et C. Palamidessi [LEV 88] (fermeture topologique du plus petit point fixe d'un opérateur associé à une version modifiée du programme), qui semble le mieux capturer la notion d'atomes infinis calculables à l'infini : les dérivations infinies qui construisent de tels objets procèdent par approximations successives et la notion de limite d'une suite d'approximations finies semble plus adéquate pour définir la dénotation d'un programme. C'est du moins ce que suggère la comparaison, effectuée dans la section 2, des termes de preuve par co-induction portant sur des objets infinis avec les dérivations infinies qui construisent effectivement ces objets. Comme point de départ de cette étude, les preuves SLD, finies ou infinies, ont été comparées à celles que l'on peut obtenir, par induction ou par co-induction, à partir des clauses d'un programme logique vues comme des règles d'inférence (les dérivations infinies ont été interprétées selon l'isomorphisme de Curry-Howard en identifiant ces dérivations infinies à des λ -termes d'un type co-inductif). Cette lecture sémantique permet de considérer un programme, non plus comme une théorie du premier ordre, mais comme un ensemble de définitions (co-)inductives. Dans le cas fini la correspondance entre preuves et SLD réfutations est complète (« ce que calcule un programme est prouvable et *vice versa* ») tandis que dans le cas infini, certains objets non calculables (par une SLD dérivation) sont toutefois prouvables (par co-induction). Les approches existantes sont exclusivement dédiées à la caractérisation d'objets infinis. L'approche que nous avons développée dans cet article correspond à une démarche « opposée » : l'univers du discours considéré ne contient pas d'éléments infinis et seules les dérivations ne construisant aucun terme infini sont considérées. Il s'agit là d'une restriction sur les dérivations qui permet de considérer le plus grand point fixe de l'opérateur $T_{\lceil P \rceil}$ tout entier. En envisageant uniquement les dérivations qui ne calculent pas à l'infini, il a alors été possible de définir une sémantique valide et complète pour les dérivations qui « prouvent à l'infini ». Quoi qu'il en soit, la définition d'une sémantique pour (toutes) les dérivations infinies ne connaît pas encore de solution générale. Néanmoins, les dérivations infinies sur le domaine des termes finis semblaient constituer un obstacle à la complétude des approches existantes et les résultats présentés ici, même s'ils ne paraissent pas surprenants, contribuent à une meilleure compréhension du problème.

Remerciements

Je remercie vivement René Lalement et Catherine Dubois pour leurs précieux conseils et leur soutien sur ce travail.

5. Bibliographie

[ABD 84] ABDALLAH M. N., « On the Interpretation of Infinite Computations in Logic Programming », PAREDAENS J., Ed., *11th International Colloquium on Automata, Languages*

- and Programming, ICALP'84*, vol. 172 de LNCS, Springer-Verlag, 1984, p. 358–370.
- [ABD 85] ABDALLAH M. N., VAN EMDEN M., « Top-Down Semantics of Fair Computations of Logic Programs », *Journal of Logic Programming*, vol. 2, n° 1, 1985, p. 67–76.
- [ACZ 77] ACZEL P., « An Introduction to Inductive Definitions », BARWISE K., Ed., *Handbook of Mathematical Logic*, Studies in Logic and Foundations of Mathematics, North Holland, 1977.
- [BOE 95] DE BOER F., PIERRO A. D., PALAMIDESSI C., « Nondeterminism and infinite computations in constraint programming », *Theoretical Computer Science*, vol. 151, n° 1, 1995, p. 37–78.
- [COQ 94] COQUAND T., « Infinite Objects in Type Theory », BARENDREGT H., NIPKOW T., Eds., *1st International Workshop on Types for Proofs and Programs, TYPES'93*, vol. 806 de LNCS, p. 62–78, Springer-Verlag, 1994.
- [DER 93] DERANSART P., MALUSZYNSKI J., *A Grammatical View of Logic Programming*, The MIT Press, 1993.
- [FAL 89] FALASCHI M., LEVI G., PALAMIDESSI C., MARTELLI M., « Declarative Modeling of the Operational Behavior of Logic Languages », *Theoretical Computer Science*, vol. 69, n° 3, 1989, p. 289–318.
- [FAL 93] FALASCHI M., LEVI G., MARTELLI M., PALAMIDESSI C., « A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs », *Information and Computation*, vol. 103, n° 1, 1993, p. 86–113.
- [GOL 88] GOLSON W., « Toward a Declarative Semantics for Infinite Objects in Logic Programming », *Journal of Logic Programming*, vol. 5, n° 2, 1988, p. 151–164.
- [HEI 92] HEIN J., « Completions of perpetual logic programs », *Theoretical Computer Science*, vol. 99, n° 1, 1992, p. 65–78.
- [HUE 90] HUET G., « *A Uniform Approach to Type Theory* », vol. Logical Foundations of Functional Programming, p. 337–398, Addison-Wesley, 1990.
- [JAF 86a] JAFFAR J., STUCKEY P., « Canonical Logic Programs », *Journal of Logic Programming*, vol. 3, n° 2, 1986, p. 143–155.
- [JAF 86b] JAFFAR J., STUCKEY P., « Semantics of Infinite Tree Logic Programming », *Theoretical Computer Science*, vol. 46, n° 2-3, 1986, p. 141–158.
- [JAU 98] JAUME M., « Logic programming and co-inductive definitions », Research Report n° 98-140, 1998, ENPC-CERMICS.
- [JAU 99] JAUME M., « A full formalisation of SLD-resolution in the calculus of inductive constructions », *Journal of Automated Reasoning*, vol. 23, n° 3-4, 1999, p. 347–371.
- [LAL 93] LALEMENT R., *Computation as Logic*, Prentice Hall International Series in Computer Science, 1993.
- [LEV 88] LEVI G., PALAMIDESSI C., « Contributions to the Semantics of Logic Perpetual Processes », *Acta Informatica*, vol. 25, n° 6, 1988, p. 691–711.
- [LLO 87] LLOYD J., *Foundations of Logic Programming*, Springer-Verlag, 1987.
- [PAU 89] PAULSON L., SMITH A., « Logic Programming, Functional Programming, and Inductive Definitions », SCHROEDER-HEISTER P., Ed., *International Workshop on Extensions of Logic Programming*, vol. 475 de LNCS, Springer-Verlag, 1989, p. 283–310.
- [POD 99] PODELSKI A., CHARATONIK W., MÜLLER M., « Set-based Failure Analysis for Logic Programs and Concurrent Constraint Programs », SWIERSTRA S. D., Ed., *8th European Symposium on Programming, ESOP'99*, LNCS, Springer-Verlag, 1999, p. 177–192.