

# Correction et complétude des sémantiques PLC revisitée par (co-)induction

**Gérard Ferrand**

LIFO, Université d'Orléans, BP 6759, 45067 Orléans Cedex 2  
E-mail : Gerard.Ferrand@lifo.univ-orleans.fr

**Alexandre Tessier**

INRIA, Domaine de Voluceau, BP 105, 78150 Le Chesnay Cedex  
E-mail : Alexandre.Tessier@inria.fr

---

**Résumé** : Cet article propose une reformulation de la sémantique des programmes logiques avec contraintes en termes de sémantique positive et sémantique négative dans un cadre inductif uniforme. Dans ce cadre, les résultats de correction et complétude s'expriment de manière naturelle et élégante. En particulier, nous montrons un résultat de complétude de la sémantique négative en utilisant des ensembles infinis de contraintes. Ce cadre théorique est une extension originale de la « vision grammaticale de la programmation logique ».

**Mots-clés** : sémantique, programmation logique avec contraintes, correction, complétude, induction, co-induction.

---

## 1 Introduction

À cause de l'indéterminisme de la *programmation logique avec contraintes* (PLC), il y a deux manières de lire la suite des réponses obtenues pour un but. Deux niveaux de calcul peuvent être considérés :

- Un premier niveau est la dérivation SLD. Une dérivation SLD finie pour un but  $\leftarrow g$  calcule une contrainte réponse  $c$ .

Par exemple si le programme  $P$  est formé des trois clauses (Prolog pur<sup>1</sup>)

$$\begin{aligned} p(X) &\leftarrow q(X) \\ q(a) &\leftarrow \\ q(b) &\leftarrow \end{aligned}$$

---

1. Rappelons que Prolog est un cas particulier de PLC. Ici, un programme logique pur suffit à l'explication.

alors pour le but  $\leftarrow p(X)$  l'une des contraintes réponses calculées est  $X = a$ .

La *correction du calcul* à ce niveau s'exprime par le fait que si  $c$  est une réponse à  $\leftarrow g$  alors  $c \rightarrow g$  est vrai dans la sémantique du programme.

Dans l'exemple il s'agit de la formule  $X = a \rightarrow p(X)$  c'est-à-dire  $p(a)$ .  $p(a)$  est vrai dans le *plus petit modèle de Herbrand* de  $P$ ,  $p(a)$  est conséquence logique de  $P$ .  $X = a \rightarrow p(X)$  est conséquence logique de  $P$  au sens du calcul des prédicats avec égalité.

À ce niveau, la *complétude du calcul* s'exprime par le fait que si  $c \rightarrow g$  est vrai dans la sémantique du programme alors  $c \rightarrow \bigvee c_i$  est vrai dans la sémantique des contraintes, où les  $c_i$  sont les réponses au but  $\leftarrow g$ .

Dans l'exemple, on peut considérer que la contrainte  $c$  est une conjonction d'égalités et la formule  $c \rightarrow \bigvee c_i$  est la formule  $c \rightarrow X = a \vee X = b$ .

La sémantique des contraintes peut être formalisée par le domaine de Herbrand muni de la seule égalité, ou bien par la *théorie complète de l'égalité de Clark* (connue sous le sigle *CET*), tout ceci en précisant le langage sous-jacent ([Fag 96]).

Par exemple ici, si les seules constantes du langage sous-jacent sont  $a$  et  $b$  la formule *vrai*  $\rightarrow p(X)$  (ici  $c$  est la conjonction vide), c'est-à-dire  $\forall X p(X)$ , est vraie dans le plus petit modèle de Herbrand de  $P$  et la formule *vrai*  $\rightarrow X = a \vee X = b$  est vraie dans le domaine de Herbrand. Si au contraire il y a d'autres constantes alors il y a moins de contraintes  $c$  telles que  $c \rightarrow p(X)$  est vrai dans le plus petit modèle de Herbrand de  $P$ . De plus, à cause de propriétés particulières à ce domaine, pour chacune de ces contraintes  $c$ , l'une des formules  $c \rightarrow X = a$  ou  $c \rightarrow X = b$  est vraie dans le domaine de Herbrand.

En termes de substitutions on retrouve que les seuls termes  $t$  pour lesquels  $p(t)$  est conséquence de  $P$  sont les termes  $a$  et  $b$ .

- Un deuxième niveau est l'arbre SLD (ou arbre de recherche). Un arbre SLD fini pour un but  $\leftarrow g$  calcule une réponse  $C$  qui est la disjonction des réponses du premier niveau.

Dans l'exemple,  $C$  est  $X = a \vee X = b$ . Cette manière de lire la suite des réponses obtenues au premier niveau comme une seule réponse du deuxième niveau est comparable à ce qui peut être aussi obtenu en Prolog ISO en utilisant convenablement une primitive comme `findall/3`.

La *correction du calcul* à ce deuxième niveau s'exprime par le fait que si  $C$  est une réponse à  $\leftarrow g$  alors  $g \rightarrow C$  est vrai dans la sémantique du programme.

Dans l'exemple, la formule  $p(X) \rightarrow X = a \vee X = b$  est bien vraie dans le plus petit modèle de Herbrand de  $P$ . Cependant cette formule n'est pas conséquence de  $P$ . Elle est conséquence de l'ensemble de formules  $\{p(X) \rightarrow q(X), q(X) \rightarrow X = a \vee X = b\}$  et donc aussi de l'ensemble de formules  $\{p(X) \leftrightarrow q(X), q(X) \leftrightarrow X = a \vee X = b\}$  noté usuellement  $P^*$ .

En général  $P^*$  ne suffit pas, il faut lui ajouter une *théorie des contraintes* (par exemple *CET*) et l'on obtient alors le *complété* de  $P$ .

À ce deuxième niveau, un cas particulier est  $C = \text{faux}$ ,  $g \rightarrow C$  est  $\neg g$ , c'est l'échec fini.

Dans l'exemple, avec le nouveau but  $\leftarrow p(f(X))$ ,  $C$  est la disjonction vide, c'est à dire  $\text{faux}$ .  $\forall X \neg p(f(X))$  est conséquence de  $\{p(X) \rightarrow q(X), q(X) \rightarrow X = a \vee X = b\}$  augmenté de *CET* puisque  $\neg(f(X) = a \vee f(X) = b)$  est conséquence de *CET*.

De façon plus générale, en un certain sens, la réponse  $C$  au deuxième niveau signifie qu'il n'y a pas d'autres réponses du premier niveau que celles qui sont dans  $C$ .

Dans l'exemple, avec le but  $\leftarrow p(X)$  la réponse de deuxième niveau ( $X = a \vee X = b$ ) signifie évidemment qu'il n'y a pas d'autres réponses de premier niveau que  $X = a$  et  $X = b$ .

C'est pourquoi nous qualifions de *positif* le premier niveau et de *négatif* le deuxième niveau. Malheureusement, il n'existe pas de complétude quand il n'existe pas d'arbre SLD fini.

L'exemple le plus trivial est obtenu en prenant  $C = \text{vrai}$  et le programme réduit à  $p \leftarrow p$ .

On a seulement un résultat, connu sous le nom de *complétude de la négation par l'échec*, quand  $\neg g$  est dans la sémantique du programme.

Cette distinction entre calcul positif et calcul négatif est une clarification utile motivée par les problèmes de validation et en particulier le débogage déclaratif [Sha 82] où l'on considère deux sortes de symptômes d'erreur : une réponse fautive (symptôme positif), une réponse manquante (symptôme négatif) [FT 97].

Cet article propose un cadre uniforme pour formaliser ces deux sémantiques<sup>2</sup> et exprimer clairement les résultats de correction et complétude. En particulier, il formule une complétude de la sémantique négative en utilisant des ensembles infinis de contraintes. D'un point de vue opérationnel, ces ensembles infinis sont naturellement introduits par les calculs infinis. D'un point de vue dénotationnel, ils peuvent être définis par *co-induction*.

L'utilité de l'induction (et plus petit point fixe) pour la sémantique est particulièrement bien connue pour la PLC (un programme peut être vu comme une définition inductive de relations [PS 89]). La co-induction (et le plus grand point fixe) est utilisée, en général, pour rendre compte d'objets non bien-fondés (par exemple [MT 91, CC 92] utilisent ce principe dans un autre contexte). [Nai 87] utilise une sémantique "plus

---

2. La *sémantique positive* (celle du premier niveau) qui donne une information positive sur le programme et la *sémantique négative* (celle du second niveau) qui fournit une information négative sur le programme.

grand point fixe” en programmation logique pour les calculs infinis. [LP 88] traite également les calculs infinis.

Cet article, en utilisant l’induction et la co-induction, montre combien la sémantique positive et la sémantique négative ont des définitions et des propriétés naturelles et élégantes. Les deux sémantiques sont étudiées parallèlement de la même manière. On retrouve ainsi facilement l’ensemble des résultats classiques [Mah 87, JM 94, Fag 96] de correction et complétude du niveau positif en précisant clairement les hypothèses nécessaires sur les solveurs de contraintes. Les résultats de correction et complétude négative que nous ajoutons en utilisant des ensembles infinis de contraintes sont obtenus aussi facilement et apparaissent naturels dans ce cadre.

Une autre contribution de cet article est l’extension de la *vision grammaticale de la programmation logique* [DM 93] non seulement à la PLC, en prenant en compte l’incomplétude des vrais solveurs de contraintes, mais aussi pour traiter la sémantique négative par les squelettes infinis. Les squelettes finis sont une notion venant de l’analyse syntaxique. Ils sont utiles pour décrire les calculs et obtenir immédiatement la confluence. D’un point de vue théorique on peut considérer que la “vision grammaticale” consiste en l’usage systématique des squelettes (finis ou non) en relation avec des considérations inductives (ou co-inductives).

## 2 Préliminaires

Considérons une fois pour toute quatre ensembles disjoints qui déterminent le langage des programmes : un ensemble infini de *variables*  $V$  ; un ensemble de *symboles de fonction*  $\Sigma$  ; un ensemble de *symboles de prédicat de contrainte*  $\Xi$  ; un ensemble de *symboles de prédicat de programme*  $\Pi$ . Chaque symbole est muni d’une arité.

Un *atome pur* (appelé ici *atome*) est une formule atomique  $p(\overline{X})$  construite sur le langage du premier ordre  $\mathcal{L}(V, \emptyset, \Pi)$ , où  $\overline{X}$  est une séquence de variables distinctes.

L’ensemble des *contraintes basiques* est un sous-ensemble du langage du premier ordre  $\mathcal{L}(V, \Sigma, \Xi)$  clos par renommage des variables. Une *contrainte* est une formule du plus petit ensemble qui contient l’ensemble des contraintes basiques et qui est clos par conjonction. Dans la suite, on identifie une conjonction de contraintes basiques avec l’ensemble des contraintes basiques de la conjonction. Par conséquent, une contrainte est vue comme un ensemble fini de contraintes basiques.

Dans  $V$  on distingue l’ensemble des variables de programme :  $V_P = \{X_{i,j} \mid i \geq 0, j \geq 1\} \cup \{Y_j \mid j \geq 1\}$ .

Une *clause* est une formule  $p_0(\overline{X}_0) \leftarrow C \wedge p_1(\overline{X}_1) \wedge \cdots \wedge p_n(\overline{X}_n)$ , où

– pour tout  $i = 0, \dots, n$  :  $n_i$  étant l’arité de  $p_i \in \Pi$ ,  $\overline{X}_i = X_{i,1}, \dots, X_{i,n_i}$  ;

- $C$  est une contrainte telle que l'ensemble de ses  $n_C$  variables libres qui ne sont pas dans  $\bigcup_{i=0,\dots,n} \{X_{i,j} \mid 1 \leq j \leq n_i\}$  est  $\{Y_j \mid 1 \leq j \leq n_C\}$ .

Le fait que l'on suppose que toutes les variables de tous les atomes sont distinctes n'est pas une restriction puisque tous les liens entre elles peuvent être exprimés dans la contrainte (par exemple par des égalités).  $V_P$  n'est qu'une manière de normaliser les noms des variables dans les clauses. Tout programme peut s'écrire sous cette forme.

Un *programme* est une famille de clauses. Dans la suite, *on suppose un programme  $P$  fixé*. L'ensemble des indices de  $P$  est noté  $\delta(P)$ .

Un *nom de clause* est un élément de  $\delta(P)$ . La *définition* de  $p \in \Pi$  dans  $P$  est la sous-famille de clauses de  $P$  dont le symbole de prédicat de tête est  $p$ ; elle est indiquée par le sous-ensemble  $\delta_p(P) \subseteq \delta(P)$ . On impose que  $\delta_p(P)$  soit fini pour tout  $p \in \Pi$ . La clause dont le nom (l'indice) est  $f$  est notée  $clause(P, f)$ .

On utilisera les notations suivantes, où  $F$  est une formule du premier ordre construite sur le langage  $\mathcal{L}(V, \Sigma, \Pi \cup \Xi)$  et  $\tilde{X} = \{X_1, \dots, X_n\} \subseteq V$ :  $var(F)$  est l'ensemble des variables libres de  $F$ ;  $\forall F$  est la clôture universelle de  $F$ ;  $\exists F$  est la clôture existentielle de  $F$ ;  $\exists_{\tilde{X}} F$  est la formule  $\exists X_1 \dots \exists X_n F$ ; et  $\exists_{- \tilde{X}} F$  est la formule  $\exists_{var(F) \setminus \tilde{X}} F$ .

Pour tout  $p \in \Pi$ , on définit :

- $IF(P, p) = \forall(p(\overline{X_0}) \leftarrow \bigvee_{f \in \delta_p(P)} \exists_{-\overline{X_0}} B_f)$ ;
- $FI(P, p) = \forall(p(\overline{X_0}) \rightarrow \bigvee_{f \in \delta_p(P)} \exists_{-\overline{X_0}} B_f)$ ;
- $IFF(P, p) = IF(P, p) \wedge FI(P, p)$ ;

où pour chaque  $f \in \delta_p(P)$ :  $clause(P, f) = p(\overline{X_0}) \leftarrow B_f$ .

On remarque que lorsque la définition de  $p$  est vide:  $FI(P, p) = \forall(\neg p(\overline{X_0}))$ .

On définit maintenant la version *si* de  $P$ , la version *seulement-si* de  $P$  et enfin la version *si-et-seulement-si* de  $P$  (le *complété* sans théorie des contraintes) comme suit :

- $IF(P) = \{IF(P, p) \mid p \in \Pi\}$ ;
- $FI(P) = \{FI(P, p) \mid p \in \Pi\}$ ;
- $IFF(P) = \{IFF(P, p) \mid p \in \Pi\}$ .

Un système de programmation logique avec contraintes utilise une adaptation de la SLD-résolution pour calculer l'ensemble des réponses à un but comme décrit dans [JL 87, JM 94, Fag 96]. Notre propos est de donner un sens logique à cet ensemble de réponses (ou à chaque réponse isolée).

L'ensemble des variables de preuve<sup>3</sup> est le sous-ensemble de  $V : V_S = \{X_{i,j}^\nu \mid i \geq 0, j \geq 1, \nu \in \mathbb{N}_+^*\} \cup \{Y_j^\nu \mid j \geq 1, \nu \in \mathbb{N}_+^*\}$ . Cet ensemble est utilisé afin de normaliser le renommage de variables durant le calcul.

Un *but* est un atome  $p(X_{0,1}^\varepsilon, \dots, X_{0,n}^\varepsilon)$ . De nouveau, ceci n'est pas une restriction par rapport aux systèmes réels qui autorisent des buts plus généraux, puisqu'on peut toujours ajouter un nouveau symbole de prédicat à  $\Pi$  et une clause à  $P$  (dont le corps est le but et la tête est construite à partir du nouveau symbole de prédicat et des variables libres du but).

Un *store* est un ensemble (fini ou infini<sup>4</sup>) de contraintes basiques dont les variables libres sont dans  $V_S$ . Un store  $S$  est satisfiable quand il existe une valuation  $v$  telle que pour chaque  $c \in S : v(c) = \text{vrai}$ . Dans ce cas,  $v$  est appelée une solution de  $S$ .

Une *pré-interprétation*  $\mathcal{D}$  est composée de :

- un ensemble non vide  $\mathbb{D}$  (le domaine);
- pour chaque symbole de fonction  $\varphi \in \Sigma$  d'arité  $n$ , une fonction  $\varphi_{\mathcal{D}}$  de  $\mathbb{D}^n$  dans  $\mathbb{D}$ ;
- pour chaque symbole de prédicat de contrainte  $\rho \in \Xi$  d'arité  $n$ , une relation  $\rho_{\mathcal{D}}$  sur  $\mathbb{D}^n$ .

Un  *$\mathcal{D}$ -atome* est un "pseudo-atome" noté  $p(d_1, \dots, d_n)$ , où  $p \in \Pi$  d'arité  $n$  et chaque  $d_i \in \mathbb{D}$ . La  *$\mathcal{D}$ -base* est l'ensemble des  $\mathcal{D}$ -atomes. Une  *$\mathcal{D}$ -interprétation*  $I$  est un sous-ensemble de la  $\mathcal{D}$ -base. Elle est identifiée à une interprétation qui prolonge<sup>5</sup>  $\mathcal{D}$ .

Une *valuation*  $v$  est une application de  $V$  dans  $\mathbb{D}$ , étendue, comme d'habitude, en des applications également notées  $v$  :

- définie sur l'ensemble des termes et à valeur dans  $\mathbb{D}$ ;
- définie sur l'ensemble des contraintes et à valeur dans  $\{\text{vrai}, \text{faux}\}$ ;
- définie sur l'ensemble des atomes et à valeur dans la  $\mathcal{D}$ -base.

3.  $\mathbb{N}_+^*$  est l'ensemble des suite finies d'entiers strictement positifs.

4. Dans la suite on s'intéresse à d'éventuelles "réponses infinies".

5. Une interprétation du langage tout entier qui est une *expansion* de  $\mathcal{D}$  c'est-à-dire qui ajoute à  $\mathcal{D}$ , pour chaque symbole de prédicat de programme  $p \in \Pi$ , d'arité  $n$ , une relation  $\rho_I$  sur  $\mathbb{D}^n$ .

On utilise les notions classiques de  $\mathcal{D}$ -modèle et  $\mathcal{D}$ -conséquence (notée par  $\models_{\mathcal{D}}$ ).

Soit  $T_P^{\mathcal{D}}$  l'opérateur de conséquence immédiate sur l'ensemble des  $\mathcal{D}$ -interprétations défini par :  $T_P^{\mathcal{D}}(I) = \{v(a) \mid v \text{ est une valuation, il existe } a \leftarrow C \wedge a_1 \wedge \dots \wedge a_n \in P, v(C) = \text{vrai et chaque } v(a_i) \in I\}$ .

$I$  étant une  $\mathcal{D}$ -interprétation, on rappelle que :

- $T_P^{\mathcal{D}}(I) \subseteq I$  si et seulement si  $I$  est un  $\mathcal{D}$ -modèle de  $\text{IF}(P)$  ;
- $T_P^{\mathcal{D}}(I) \supseteq I$  si et seulement si  $I$  est un  $\mathcal{D}$ -modèle de  $\text{FI}(P)$  ;
- $T_P^{\mathcal{D}}(I) = I$  si et seulement si  $I$  est un  $\mathcal{D}$ -modèle de  $\text{IFF}(P)$ .

$T_P^{\mathcal{D}}$  est croissant (et continu), donc il a un plus petit et un plus grand point fixe :

- $\text{pppf}(T_P^{\mathcal{D}})$  est le plus petit  $\mathcal{D}$ -modèle de  $\text{IF}(P)$  (et  $\text{IFF}(P)$ ) ;
- $\text{pgpf}(T_P^{\mathcal{D}})$  est le plus grand  $\mathcal{D}$ -modèle de  $\text{FI}(P)$  (et  $\text{IFF}(P)$ ).

Dans cet article il s'agit de formaliser les relations logiques entre un but et les contraintes réponses à ce but. Cela conduit à étudier les conséquences de  $\text{IFF}(P)$  de la forme  $a \leftrightarrow F$  où  $a$  est une atome et  $F$  est une formule construite sur le langage des contraintes  $\mathcal{L}(V, \Sigma, \Xi)$ . Par ailleurs, on rappelle que  $\models \forall(a \leftrightarrow F) \leftrightarrow \forall((a \leftarrow F) \wedge (a \rightarrow F))$  et  $\models \forall(a \leftarrow \bigvee_{1 \leq i \leq n} F_i) \leftrightarrow \bigwedge_{1 \leq i \leq n} \forall(a \leftarrow F_i)$ .

En utilisant  $\text{pppf}(T_P^{\mathcal{D}})$  et  $\text{pgpf}(T_P^{\mathcal{D}})$  on obtient les équivalences suivantes, où  $F$  est une formule sur le langage des contraintes<sup>6</sup>  $\mathcal{L}(V, \Sigma, \Xi)$ ,  $a$  est un atome,  $\mathcal{D}$  est une pré-interprétation et  $\mathcal{T}$  est une théorie des contraintes<sup>7</sup> :

- $\text{IFF}(P) \models_{\mathcal{D}} \forall(a \leftarrow F)$  si et seulement si  $\text{IF}(P) \models_{\mathcal{D}} \forall(a \leftarrow F)$ , et, puisque cela est vrai pour tout  $\mathcal{D}$ , on a donc :  $\text{IFF}(P) \models \forall(a \leftarrow F)$  si et seulement si  $\text{IF}(P) \models \forall(a \leftarrow F)$ , et de même :  $\mathcal{T}, \text{IFF}(P) \models \forall(a \leftarrow F)$  si et seulement si  $\mathcal{T}, \text{IF}(P) \models \forall(a \leftarrow F)$  ;
- $\text{IFF}(P) \models_{\mathcal{D}} \forall(a \rightarrow F)$  si et seulement si  $\text{FI}(P) \models_{\mathcal{D}} \forall(a \rightarrow F)$ , et, puisque cela est vrai pour tout  $\mathcal{D}$ , on a donc :  $\text{IFF}(P) \models \forall(a \rightarrow F)$  si et seulement si  $\text{FI}(P) \models \forall(a \rightarrow F)$ , et de même :  $\mathcal{T}, \text{IFF}(P) \models \forall(a \rightarrow F)$  si et seulement si  $\mathcal{T}, \text{FI}(P) \models \forall(a \rightarrow F)$ .

6. Le résultat est faux, en général, quand  $F$  est une formule sur  $\mathcal{L}(V, \Sigma, \Pi \cup \Xi)$ .

7. Une théorie des contraintes est une théorie sur le langage des contraintes.

### 3 Deux définitions inductives

Cette section donne des définitions préliminaires, utilisées dans la section 5 pour montrer la correction et la complétude de la sémantique opérationnelle (positive et négative) des systèmes de PLC. Il n’y a pas de notion de calcul ici.

Dans un premier temps on s’intéresse à un ensemble de règles sur les couples  $(t : p(\bar{d}))$ , où  $t$  est un arbre<sup>8</sup> étiqueté par  $\delta(P)$  et  $p(\bar{d})$  est un  $\mathcal{D}$ -atome. Pour tout *nom de clause*  $f \in \delta(P)$ , soit  $clause(P, f) = p_0(\bar{X}_0) \leftarrow C \wedge p_1(\bar{X}_1) \wedge \cdots \wedge p_n(\bar{X}_n)$ , pour toute valuation  $v$  solution de  $C$ , pour tous arbres  $t_1, \dots, t_n$ , on définit la règle :

$$\frac{t_1 : v(p_1(\bar{X}_1)) \quad \cdots \quad t_n : v(p_n(\bar{X}_n))}{f(t_1, \dots, t_n) : v(p_0(\bar{X}_0))}$$

L’opérateur associé à l’ensemble de règles précédent est noté  $\mathcal{T}_P^{\mathcal{D}}$ . Il est croissant (et continu), donc a un plus petit et un plus grand point fixe.

Un élément  $t : p(\bar{d})$  du plus petit point fixe de  $\mathcal{T}_P^{\mathcal{D}}$  est tel que  $t$  est fini ( $\mathcal{T}_P^{\mathcal{D}}$  est continu et aux “règles faits” correspond un arbre réduit à un seul nœud).  $t$  correspond à la notion de squelette [DM 93] et, exactement, à la notion de squelette complet non rejeté défini par des termes sur  $\delta(P)$  dans [FT 97].

Un élément  $t : p(\bar{d})$  du plus grand point fixe de  $\mathcal{T}_P^{\mathcal{D}}$  est tel que  $t$  peut être infini.  $t$  correspond à la notion plus générale de squelette infini.

L’opérateur  $\mathcal{T}_P^{\mathcal{D}}$  est très fortement relié à l’opérateur  $T_P^{\mathcal{D}}$  :

**Lemme 1** Lien entre  $\mathcal{T}_P^{\mathcal{D}}$  et  $T_P^{\mathcal{D}}$   
 $\{p(\bar{d}) \mid \text{il existe } t \text{ tel que } t : p(\bar{d}) \in \text{pppf}(\mathcal{T}_P^{\mathcal{D}})\} = \text{pppf}(T_P^{\mathcal{D}})$  (le plus petit  $\mathcal{D}$ -modèle de  $\mathbf{IF}(P)$ );  
 $\{p(\bar{d}) \mid \text{il existe } t \text{ tel que } t : p(\bar{d}) \in \text{pgpf}(\mathcal{T}_P^{\mathcal{D}})\} = \text{pgpf}(T_P^{\mathcal{D}})$  (le plus grand  $\mathcal{D}$ -modèle de  $\mathbf{FI}(P)$ ).

**Preuve** Simple application des définitions inductives. On a *internalisé*<sup>9</sup> un codage des preuves : dans  $t : p(\bar{d})$ ,  $t$  code une preuve de  $p(\bar{d})$ .  $\square$

Les  $\mathcal{D}$ -atomes ne sont pas des notions syntaxiques. En fait, les calculs ne font appel qu’aux notions syntaxiques que sont les contraintes et les atomes. La définition inductive précédente a été donnée à titre pédagogique et parce qu’elle sert aux preuves

8.  $f(t_1, \dots, t_n)$  désigne l’arbre enraciné par  $f \in \delta(P)$  dont les sous-arbres sont  $t_1, \dots, t_n$ .

9. C’est comparable à la théorie constructive des types où les  $\lambda$ -termes codent les preuves de formules, mais ici basé uniquement sur le modus ponens.

des résultats des sections suivantes. On introduit maintenant une seconde définition inductive pour prouver des “pseudo-formules” du langage. Mais un formalisme plus élaboré sera utile pour tenir compte d’éventuelles “réponses infinies” :

Un *séquent positif* est un couple  $S \vdash p$ , où  $S$  est un store et  $p \in \Pi$ . Un séquent positif  $S \vdash p$  est *valide* dans une interprétation  $I$  si pour toute solution  $v$  de  $S$  :  $v(p(X_{0,1}^\varepsilon, \dots, X_{0,n}^\varepsilon)) \in I$  ( $n$  est l’arité de  $p$ ). Quand  $S$  est fini, c’est équivalent à  $I \models \bigwedge_{c \in S} c \rightarrow p(X_{0,1}^\varepsilon, \dots, X_{0,n}^\varepsilon)$ .

Un *séquent négatif* est un couple  $p \vdash \mathcal{Z}$ , où  $\mathcal{Z}$  est un *ensemble* de stores et  $p \in \Pi$ . Un séquent négatif  $p \vdash \mathcal{Z}$  est *valide* dans une interprétation  $I$  si pour toute valuation  $v$  telle que  $v(p(X_{0,1}^\varepsilon, \dots, X_{0,n}^\varepsilon)) \in I$  : il existe  $S \in \mathcal{Z}$  et il existe une solution  $v'$  de  $S$  tels que pour tout  $1 \leq i \leq n$  :  $v(X_{0,i}^\varepsilon) = v'(X_{0,i}^\varepsilon)$ . Quand  $\mathcal{Z}$  est un ensemble fini de stores finis, c’est équivalent à  $I \models p(X_{0,1}^\varepsilon, \dots, X_{0,n}^\varepsilon) \rightarrow \bigvee_{S \in \mathcal{Z}} \exists_{-\{X_{0,1}^\varepsilon, \dots, X_{0,n}^\varepsilon\}} \bigwedge_{c \in S} c$ .

Soient  $F$  une formule telle que  $\text{var}(F) \subseteq V_P$  (l’ensemble des variables de programme) et  $\nu \in \mathbb{N}_+^*$ .  $F^\nu$  désigne la variante de  $F$  où les  $X_{i,j}$  sont remplacés par  $X_{i,j}^\nu$  et les  $Y_j$  sont remplacés par  $Y_j^\nu$ .

Soient  $F$  une formule telle que  $\text{var}(F) \subseteq V_S$  (l’ensemble des variables de preuve) et  $k \in \mathbb{N}_+$ .  $F^{+k}$  désigne la variante de  $F$  où les  $X_{i,j}^\nu$  sont remplacés par<sup>10</sup>  $X_{i,j}^{\nu k}$  et les  $Y_j^\nu$  sont remplacés par  $Y_j^{\nu k}$ . Si  $S$  est un ensemble de formules alors  $S^{+k} = \{F^{+k} \mid F \in S\}$ .

On donne maintenant la seconde définition inductive qui prouve des séquents positifs (et négatifs).

On s’intéresse à un ensemble de règles sur les couples  $(t : S \vdash p)$ , où  $t$  est un arbre étiqueté par  $\delta(P)$  et  $S \vdash p$  est un séquent positif. Pour tout *nom de clause*  $f \in \delta(P)$ , soit  $\text{clause}(P, f) = p_0(\overline{X_0}) \leftarrow C \wedge p_1(\overline{X_1}) \wedge \dots \wedge p_n(\overline{X_n})$ , pour tous arbres  $t_1, \dots, t_n$  et pour tous stores  $S_1, \dots, S_n$ , on définit la règle :

$$\frac{t_1 : S_1 \vdash p_1 \quad \dots \quad t_n : S_n \vdash p_n}{f(t_1, \dots, t_n) : S_0 \vdash p_0}$$

où  $S_0 = C^\varepsilon \cup S_1^{+1} \cup \dots \cup S_n^{+n} \cup S'$ , avec  $S' = \bigcup_{1 \leq i \leq n} \bigcup_{1 \leq j \leq n_i} \{X_{i,j}^\varepsilon = X_{0,j}^i\}$ , où chaque  $n_i$  est l’arité de  $p_i$ .

L’opérateur associé à cet ensemble de règles est noté  $\mathcal{T}_P$ . Il est croissant (et continu), donc a un plus petit et un plus grand point fixe.

Un élément  $t : S \vdash p$  de  $\text{pppf}(\mathcal{T}_P)$  est tel que  $t$  et  $S$  sont finis.  $t$  code une preuve de  $S \vdash p$  (on remarque que  $S \vdash p$  ne dépend que de  $t$ ),  $S$  est équivalent à la notion classique de contrainte réponse [JM 94] en laissant de coté les problèmes de satisfiabilité.

10.  $\nu k$  est la concaténation de  $\nu$  et  $k$ .

Un élément  $t : S \vdash p$  de  $pppf(\mathcal{T}_P)$  est tel que  $t$  et  $S$  peuvent être infinis. À nouveau,  $S \vdash p$  ne dépend que de  $t$ .  $S \vdash p$  est le séquent dont le squelette est  $t$ .

Les opérateurs  $\mathcal{T}_P^D$  et  $\mathcal{T}_P$  sont reliés par le fait que si  $t : S \vdash p \in pppf(\mathcal{T}_P)$  (resp.  $pppf(\mathcal{T}_P)$ ) et s'il existe une solution  $v$  de  $S$  alors  $t : v(p(\overline{X}_0^\varepsilon)) \in pppf(\mathcal{T}_P^D)$  (resp.  $pppf(\mathcal{T}_P^D)$ ). Réciproquement, si  $t : v(p(\overline{X}_0^\varepsilon)) \in pppf(\mathcal{T}_P^D)$  (resp.  $pppf(\mathcal{T}_P^D)$ ) alors il existe un store  $S$  et une solution  $v'$  de  $S$ ,  $v(p(\overline{X}_0^\varepsilon)) = v'(p(\overline{X}_0^\varepsilon))$ , tels que  $t : S \vdash p \in pppf(\mathcal{T}_P)$  (resp.  $pppf(\mathcal{T}_P)$ ).

#### 4 Vue logique des définitions inductives

Dans cette section, on donne un sens logique (correction/complétude) à la définition inductive de la section précédente. On commence par sa correction, du point de vue positif comme du point de vue négatif.

**Lemme 2** Correction positive.

Soit  $I$  un  $\mathcal{D}$ -modèle de  $\text{IF}(P)$ .

Si  $t : S \vdash p \in pppf(\mathcal{T}_P)$  alors  $S \vdash p$  est valide dans  $I$  (i.e.  $\text{IF}(P) \models_{\mathcal{D}} \bigwedge_{c \in S} c \rightarrow p(\overline{X}_0^\varepsilon)$  parce que  $S$  est fini).

**Preuve** Par induction.  $\square$

**Corollaire 3** Correction positive.

Si  $t : S \vdash p \in pppf(\mathcal{T}_P)$  alors  $\mathcal{T}, \text{IF}(P) \models \bigwedge_{c \in S} c \rightarrow p(\overline{X}_0^\varepsilon)$ , où  $\mathcal{T}$  est une théorie des contraintes.

Si  $t : S \vdash p \in pppf(\mathcal{T}_P)$  alors  $\text{IF}(P) \models \bigwedge_{c \in S} c \rightarrow p(\overline{X}_0^\varepsilon)$ .

On note  $\mathcal{Z}^-(p)$  l'ensemble  $\{S \mid \text{il existe } t \text{ tel que } t : S \vdash p \in pppf(\mathcal{T}_P)\}$ . Le séquent négatif associé à  $p \in \Pi$  est  $p \vdash \mathcal{Z}^-(p)$ .

**Lemme 4** Correction négative.

Soit  $I$  un  $\mathcal{D}$ -modèle de  $\text{FI}(P)$ .

$p \vdash \mathcal{Z}^-(p)$  est valide dans  $I$ .

**Preuve** Utilise le lemme 1.  $\square$

**Corollaire 5** Correction négative.

Si  $\mathcal{Z}^-(p)$  est un ensemble fini de stores finis alors :

$$\text{FI}(P) \models_{\mathcal{D}} p(\overline{X_0^\varepsilon}) \rightarrow \bigvee_{S \in \mathcal{Z}^-(p)} \exists_{-\overline{X_0^\varepsilon}} \bigwedge_{c \in S} c;$$

$$\mathcal{T}, \text{FI}(P) \models p(\overline{X_0^\varepsilon}) \rightarrow \bigvee_{S \in \mathcal{Z}^-(p)} \exists_{-\overline{X_0^\varepsilon}} \bigwedge_{c \in S} c, \text{ où } \mathcal{T} \text{ est une théorie des contraintes ;}$$

$$\text{FI}(P) \models p(\overline{X_0^\varepsilon}) \rightarrow \bigvee_{S \in \mathcal{Z}^-(p)} \exists_{-\overline{X_0^\varepsilon}} \bigwedge_{c \in S} c.$$

Pour généraliser le corollaire précédent, on définit la notion de séquent négatif *conséquence* d'un ensemble de formules. Un séquent négatif  $p \vdash \mathcal{Z}$  est conséquence de  $\Gamma$  quand il est valide dans tous les modèles de  $\Gamma$ .

**Corollaire 6** Correction négative.

$p \vdash \mathcal{Z}^-(p)$  est conséquence de  $\mathcal{T} \cup \text{FI}(P)$ , où  $\mathcal{T}$  est une théorie des contraintes (en particulier  $p \vdash \mathcal{Z}^-(p)$  est conséquence de  $\text{FI}(P)$ ).

Pour exprimer la complétude de la définition inductive, on introduit une notion similaire à la notion de séquent, mais qui ne fait référence qu'à des formules sur le langage des contraintes.

Une *couverture négative* est un couple  $F \vdash \mathcal{Z}$ , où  $F$  est une formule du langage des contraintes et  $\mathcal{Z}$  est un ensemble de stores. Une couverture négative  $F \vdash \mathcal{Z}$  est *valide* dans une pré-interprétation  $\mathcal{D}$  si pour toute solution  $v$  de  $F$  : il existe un store  $S \in \mathcal{Z}$  dont  $v$  est une solution. Le cas fini correspond à  $F \rightarrow \bigvee_{S \in \mathcal{Z}} \bigwedge_{c \in S} c$  valide dans  $\mathcal{D}$ . Une couverture négative est conséquence d'une théorie des contraintes  $\mathcal{T}$  si elle est valide dans tous les modèles de  $\mathcal{T}$ . La notion de couverture négative quand tous les stores de  $\mathcal{Z}$  sont finis correspond à la notion habituelle de couverture dans les théorèmes de complétude classiques [Mah 87, JM 94, Tes 96a].

Une *couverture positive* est un couple  $S \vdash F$ , où  $F$  est une formule du langage des contraintes et  $S$  est un store. Une couverture positive  $S \vdash F$  est *valide* dans une pré-interprétation  $\mathcal{D}$  si pour toute solution  $v$  de  $S$  :  $v$  est solution de  $F$ . Le cas fini correspond à  $\bigwedge_{c \in S} c \rightarrow F$  valide dans  $\mathcal{D}$ . Une couverture positive est conséquence d'une théorie des contraintes  $\mathcal{T}$  si elle est valide dans tout modèle de  $\mathcal{T}$ .

On note  $\mathcal{Z}^+(p)$  l'ensemble  $\{S \mid \text{il existe } t \text{ tel que } t : S \vdash p \in \text{pppf}(\mathcal{T}_P)\}$ .

**Lemme 7** Complétude positive (couverture positive, interpolation positive).

Soit  $F$  une formule du langage des contraintes.

Si  $\text{IF}(P) \models_{\mathcal{D}} F \rightarrow p(\overline{X_0^\varepsilon})$  alors la couverture négative  $\exists_{-\overline{X_0^\varepsilon}} F \vdash \mathcal{Z}^+(p)$  est valide dans  $\mathcal{D}$ .

**Preuve** Utilise le plus petit  $\mathcal{D}$ -modèle de  $\text{IF}(P)$  et le lemme 1.  $\square$

D'un point de vue logique, cette propriété peut être vue comme une *interpolation* :  $\mathcal{Z}^+(p)$  est une formule intermédiaire (formule d'interpolation) entre  $\exists_{-\overline{X_0^\varepsilon}} F$  et  $p(\overline{X_0^\varepsilon})$ .

**Corollaire 8** Complétude positive et compacité.

Soit  $F$  une formule du langage des contraintes.

Si  $\mathcal{T}, \text{IF}(P) \models F \rightarrow p(\overline{X_0^\varepsilon})$  alors la couverture négative  $\exists_{-\overline{X_0^\varepsilon}} F \vdash \mathcal{Z}^+(p)$  est conséquence de  $\mathcal{T}$ , et il existe une partie finie<sup>11</sup>  $\mathcal{Z} \subseteq \mathcal{Z}^+(p)$  telle que  $\exists_{-\overline{X_0^\varepsilon}} F \vdash \mathcal{Z}$  est conséquence de  $\mathcal{T}$ , i.e.  $\mathcal{T} \models \exists_{-\overline{X_0^\varepsilon}} F \rightarrow \bigvee_{S \in \mathcal{Z}} \bigwedge_{c \in S} c$  (compacité de la logique du premier ordre et chaque  $S \in \mathcal{Z}^+(p)$  est fini).

**Lemme 9** Complétude négative (couverture négative, interpolation négative).

Soit  $F$  une formule du langage des contraintes.

Si  $\text{FI}(P) \models_{\mathcal{D}} p(\overline{X_0^\varepsilon}) \rightarrow F$  alors pour tout  $S \in \mathcal{Z}^-(p)$  : la couverture positive  $S \vdash \exists_{-\overline{X_0^\varepsilon}} F$  est valide dans  $\mathcal{D}$ .

**Preuve** Utilise le plus grand  $\mathcal{D}$ -modèle de  $\text{FI}(P)$  et le lemme 1.  $\square$

À nouveau,  $\mathcal{Z}^-(p)$  est une formule d'interpolation entre  $p(\overline{X_0^\varepsilon})$  et  $\exists_{-\overline{X_0^\varepsilon}} F$ .

**Corollaire 10** Complétude négative et compacité.

Soit  $F$  une formule du langage des contraintes.

Si  $\mathcal{T}, \text{FI}(P) \models p(\overline{X_0^\varepsilon}) \rightarrow F$  alors pour tout  $S \in \mathcal{Z}^-(p)$  : la couverture positive  $S \vdash \exists_{-\overline{X_0^\varepsilon}} F$  est conséquence de  $\mathcal{T}$  et il existe une partie finie  $S'$  de  $S$  telle que  $S' \vdash \exists_{-\overline{X_0^\varepsilon}} F$  est conséquence de  $\mathcal{T}$ , i.e.  $\mathcal{T} \models \bigwedge_{c \in S} c \rightarrow \exists_{-\overline{X_0^\varepsilon}} F$ .

À travers les lemmes de correction et complétude nous avons donné le sens logique des séquents positifs et négatifs. Cependant, les systèmes réels de programmation logique avec contraintes ne calculent pas tous ces séquents. Les systèmes stoppent le calcul quand ils détectent que le store courant est insatisfiable, grâce au “solveur de contraintes”. Par conséquent, un grand nombre de séquents positifs ne sont jamais calculés. En fait, si  $S \vdash p \in \text{pppf}(\mathcal{T}_P)$  et  $S$  est insatisfiable alors  $S \rightarrow p(\overline{X_0^\varepsilon})$  est toujours vrai et sans intérêt car indépendant du programme  $P$ . Pour formaliser le solveur de contraintes nous introduisons, dans la section suivante, le *critère de rejet*.

<sup>11</sup>. Réduite à un singleton quand le domaine à la propriété d'Indépendance des Contraintes Négatives [Mah 93], comme dans l'exemple donné en introduction.

## 5 Correction et complétude (de la sémantique opérationnelle)

Cette section étudie la correction et la complétude de la sémantique opérationnelle théorique classique des systèmes de PLC.

Un *critère de rejet* est un ensemble de stores RC. Un store  $S$  est *rejeté* si  $S \in \text{RC}$ . Les systèmes réels ne manipulent que des stores finis, aussi, dans un premier temps, on définit la relation RC sur les stores finis. Le critère de rejet vérifie les propriétés suivantes, où  $S$  est un store fini : si  $S \in \text{RC}$  alors pour tout renommage de variables  $\theta$  :  $S\theta \in \text{RC}$  ; si  $S \in \text{RC}$  alors pour tout store fini  $S'$  :  $S \cup S' \in \text{RC}$  ;  $\emptyset \notin \text{RC}$  (le store vide  $\emptyset$  est la constante logique *vrai*).

Un critère de rejet RC est *correct* par rapport à la pré-interprétation  $\mathcal{D}$  (resp. la théorie  $\mathcal{T}$ ) quand  $S \in \text{RC}$  implique qu'il n'existe aucune solution de  $S$  dans  $\mathcal{D}$  (resp. il n'existe aucune solution de  $S$  dans tout modèle de  $\mathcal{T}$ ).

Un critère de rejet RC est *complet* par rapport à la pré-interprétation  $\mathcal{D}$  (resp. la théorie  $\mathcal{T}$ ) quand s'il n'existe aucune solution de  $S$  dans  $\mathcal{D}$  (resp. s'il n'existe aucune solution de  $S$  dans tout modèle de  $\mathcal{T}$ ) alors cela implique que  $S \in \text{RC}$ .

Par exemple, le critère de rejet peut être défini par une relation *consistent* et une fonction *infer* (voir [JM 94]) de la manière suivante : un store  $S$  est rejeté si  $\text{infer}(\emptyset, S) = (S_1, S_2)$  et  $S_1 \notin \text{consistent}$ .

On suppose le lecteur familier avec les notions de dérivation SLD et d'arbre SLD par rapport à un "solveur de contraintes" [JM 94] formalisé ici par un critère de rejet. On peut voir une dérivation SLD comme la construction d'un squelette, elle est alors définie comme une suite de squelettes partiels finis<sup>12</sup> [Tes 97] ; un arbre SLD regroupe l'ensemble des dérivations pour un but selon une règle de calcul [Tes 96b].

Un *calcul positif* est une dérivation SLD succès. À toute dérivation SLD succès pour le but  $p(\overline{X_0^\varepsilon})$  correspond un élément  $t$  :  $S \vdash p$  de  $\text{pppf}(\mathcal{T}_P)$  tel que  $S \notin \text{RC}$ . Le séquent positif  $S \vdash p$  est alors appelé un *séquent positif RC-calculé*. Le *squelette fini*  $t$  caractérise les clauses utilisées par la dérivation et  $S$  est le store calculé par la dérivation. Ceci permet de justifier facilement la propriété appelée *indépendance (positive) de la règle de calcul* : les branches succès de deux arbres SLD pour le but  $p(\overline{X_0^\varepsilon})$  sont en bijection car chaque branche succès est caractérisée par un squelette fini  $t$ .

Comme cas particulier du lemme 2 on a :

**Théorème 11** Correction positive.

Soient RC un critère de rejet et  $I$  un  $\mathcal{D}$ -modèle de  $\text{IF}(P)$ .

Si  $S \vdash p$  est un séquent positif RC-calculé alors  $S \vdash p$  est valide dans  $I$ .

<sup>12</sup> Un squelette partiel est un squelette dans lequel pour certains nœuds on a remplacé le sous-arbre greffé par une feuille indéfinie.

On peut également énoncer un corollaire comme pour le corollaire 3.

Soit  $\mathcal{Z}^+(p, \text{RC}) = \{S \mid \text{il existe } t \text{ tel que } t : S \vdash p \in \text{pppf}(\mathcal{T}_P), S \notin \text{RC}\}$ , c'est-à-dire  $\{S \mid S \vdash p \text{ est un séquent positif RC-calculé}\}$ .

**Théorème 12** Complétude positive.

Soient  $\text{RC}$  un critère de rejet correct par rapport à  $\mathcal{D}$  et  $F$  une formule du langage des contraintes.

Si  $\text{IF}(P) \models_{\mathcal{D}} F \rightarrow p(\overline{X_0^\varepsilon})$  alors la couverture négative  $\exists_{-\overline{X_0^\varepsilon}} F \vdash \mathcal{Z}^+(p, \text{RC})$  est valide dans  $\mathcal{D}$ .

**Preuve** D'après le lemme 7 parce que les éléments de  $\mathcal{Z}^+(p) \setminus \mathcal{Z}^+(p, \text{RC})$  n'ont pas de solution dans  $\mathcal{D}$ .  $\square$

De la même façon, on peut formuler une version du corollaire 8 avec un critère de rejet *correct* par rapport à  $\mathcal{T}$ .

Un calcul *négatif* correspond au calcul d'un arbre SLD.

Pour pouvoir formaliser les dérivations infinies (*équitable*), le critère de rejet est étendu aux stores infinis de la manière suivante : un store  $S$  est rejeté si et seulement si il existe une partie finie de  $S$  rejetée. C'est une forme de compacité du critère de rejet pour exprimer le fait que quand une dérivation SLD est infinie le calcul n'a jamais rencontré de store rejeté.

À toute dérivation SLD équitable (finie ou infinie) et non échec pour le but  $p(\overline{X_0^\varepsilon})$  correspond un élément  $t : S \vdash p$  de  $\text{pppf}(\mathcal{T}_P)$  tel que  $S \notin \text{RC}$ . Le squelette  $t$  caractérise les clauses utilisées par la dérivation et  $S$  est l'ensemble des contraintes accumulées par la dérivation. Ceci permet de justifier une propriété que nous appelons *indépendance négative de la règle de calcul équitable* : les branches non échecs de deux arbres SLD équitables<sup>13</sup> pour le but  $p(\overline{X_0^\varepsilon})$  sont en bijection car chaque branche est caractérisée par un squelette  $t$ .

Soit  $\mathcal{Z}^-(p, \text{RC}) = \{S \mid \text{il existe } t \text{ tel que } t : S \vdash p \in \text{pppf}(\mathcal{T}_P), S \notin \text{RC}\}$ .

Le séquent négatif associé à  $p$  selon  $\text{RC}$  est  $p \vdash \mathcal{Z}^-(p, \text{RC})$ . Le séquent négatif est *RC-calculé* quand  $\mathcal{Z}^-(p, \text{RC})$  est un ensemble fini de stores finis (i.e. quand il existe un arbre SLD fini).

**Théorème 13** Correction négative.

Soient  $\text{RC}$  un critère de rejet correct par rapport à  $\mathcal{D}$  et  $I$  un  $\mathcal{D}$ -modèle de  $\text{FI}(P)$ .  $p \vdash \mathcal{Z}^-(p, \text{RC})$  est valide dans  $I$ .

13. On rappelle que tout arbre SLD fini est équitable (indépendamment de la règle de calcul).

**Preuve** D'après le lemme 4 parce que les éléments de  $\mathcal{Z}^-(p) \setminus \mathcal{Z}^-(p, \text{RC})$  n'ont pas de solution dans  $\mathcal{D}$ .  $\square$

On remarque que quand il y a échec fini pour le but  $p(\overline{X_0^\varepsilon})$  on a  $\mathcal{Z}^-(p, \text{RC}) = \emptyset$ , donc  $p \vdash \emptyset$ , i.e.  $\neg p(\overline{X_0^\varepsilon})$  valide dans  $I$ .

De même, on peut formuler une version du corollaire 5 quand RC est *correct* par rapport à  $\mathcal{T}$ .

**Théorème 14** Complétude négative.

Soient RC un critère de rejet,  $\mathcal{D}$  une pré-interprétation et  $F$  une formule du langage des contraintes.

Si  $\text{FI}(P) \models_{\mathcal{D}} p(\overline{X_0^\varepsilon}) \rightarrow F$  alors pour tout  $S \in \mathcal{Z}^-(p, \text{RC})$ : la couverture positive  $S \vdash \exists_{\overline{X_0^\varepsilon}} F$  est valide dans  $\mathcal{D}$ .

**Preuve** D'après le lemme 9 parce que  $\mathcal{Z}^-(p, \text{RC})$  est un sous-ensemble de  $\mathcal{Z}^-(p)$ .  $\square$

À nouveau, on peut formuler une version du corollaire 10 qui amène très simplement au résultat bien connu de *complétude de la négation par l'échec*:

**Théorème 15** Complétude de la négation par l'échec.

Soit RC un critère de rejet complet par rapport à une théorie des contraintes  $\mathcal{T}$ .

Si  $\mathcal{T}, \text{FI}(P) \models \neg p(\overline{X_0^\varepsilon})$  alors tout arbre SLD (selon RC) équitable pour le but  $p(\overline{X_0^\varepsilon})$  est un arbre SLD d'échec fini.

**Preuve** Il suffit de montrer que  $\mathcal{Z}^-(p, \text{RC})$  est vide (i.e. il n'y a pas de branche non échec dans les arbres SLD équitables). Mais pour tout  $S \in \mathcal{Z}^-(p, \text{RC})$ :  $S \vdash \text{faux}$  d'après le corollaire du théorème 14 et si  $\mathcal{Z}^-(p, \text{RC})$  contient un store  $S$  alors  $S$  n'est pas rejeté et, parce que le critère de rejet est complet par rapport à  $\mathcal{T}$ , on a une contradiction (à cause de la compacité de la logique du premier ordre).  $\square$

On aurait aimé avoir un résultat de complétude négative finie (qui étende le résultat de complétude de la négation par l'échec), comme par exemple: si  $\mathcal{T}, \text{FI}(P) \models p(\overline{X_0^\varepsilon}) \rightarrow F$  alors il existe un arbre SLD fini pour le but  $p(\overline{X_0^\varepsilon})$  tel que, pour tout  $S \in \mathcal{Z}^-(p, \text{RC})$ ,  $\bigwedge_{c \in S} c \rightarrow \exists_{\overline{X_0^\varepsilon}} F$  est valide. Mais, on voit la raison pour laquelle il n'y a pas de résultat plus général de complétude négative finie. Supposons que  $\mathcal{T}, \text{FI}(P) \models p(\overline{X_0^\varepsilon}) \rightarrow F$ . Montrer qu'il existe un arbre SLD fini pour le but  $p(\overline{X_0^\varepsilon})$  tel que chaque succès implique  $F$  revient à montrer que  $\mathcal{Z}^-(p, \text{RC})$  est un ensemble fini de stores finis. Même si RC est complet par rapport à  $\mathcal{T}$ , le seul cas où l'on peut le garantir est quand  $F$  est *faux*, i.e.  $\mathcal{Z}^-(p, \text{RC}) = \emptyset$ .

## 6 Conclusion

Nous avons esquissé une reformulation de la sémantique de la PLC basée sur les définitions inductives. En ne considérant que les principes d'induction et de co-induction, les résultats de bases sont retrouvés d'une manière très naturelle et élégante. Par exemple, dans cette approche, on voit directement où interviennent les hypothèses sur le solveur de contraintes (correction, complétude) : dans les théorèmes 11 (correction positive) et 14 (complétude négative) il n'y a aucune hypothèse sur le critère de rejet alors que dans les théorèmes 12 (complétude positive) et 13 (correction négative) il est supposé correct. On remarque également qu'il est supposé complet dans le théorème 15 (complétude de la négation par l'échec). Peu de vrais solveurs sont complets, d'où l'intérêt limité de ce résultat dans le cadre de la PLC, et en revanche le grand intérêt des recherches autour d'autres types de négation (par exemple constructive [Fag 97]).

Dans [Lio 97] des résultats de correction et complétude en logique bivaluée sont aussi obtenus en faisant intervenir des ensembles infinis de formules pour des programmes logiques plus généraux (normaux, c'est-à-dire avec négation dans les corps de clause). Contrairement à [Lio 97] le présent article ne cherche qu'à étudier le sens logique des réponses obtenues conformément aux sémantiques opérationnelles usuelles. D'autre part les outils théoriques utilisés ici, (co-)induction et en particulier l'internalisation de la preuve sous la forme d'un arbre  $t$  (squelette) dans le séquent  $t : S \vdash p$ , sont beaucoup plus élémentaires et semblent bien adaptés à cet objectif.

Dans ce formalisme il est possible de donner une définition inductive qui correspond exactement à ce qui est calculé : il suffit de ne garder que les règles dont la conclusion a un store non rejeté. Dans ce cas, le plus petit point fixe de l'opérateur associé correspond à la  $S$ -sémantique du programme [BGLM 94].

Des notions inductives supplémentaires peuvent être utilisées pour formaliser le *calcul négatif fini* : dans [FT 97] une définition inductive est à la base d'une notion d'erreur (*non complètement couvert*) associée à un symptôme de réponse manquante.

Le cadre théorique esquissé dans cet article est une extension à la PLC de la vision grammaticale de la programmation logique [DM 93]. La nouveauté n'est pas seulement le paramétrage par un domaine de contraintes et un critère de rejet, c'est aussi l'utilisation des squelettes finis et infinis. Par exemple, ils fournissent une caractérisation simple des branches (finies ou infinies) non échec des arbres SLD équitables (*indépendance négative de la règle de calcul équitable*).

## Bibliographie

- [BGLM 94] BOSSI A., GABRIELLI M., LEVI G., et MARTELLI M., « The S-Semantics Approach: Theory and Applications », *Journal of Logic Programming*, vol. 19&20, :149–198, 1994.
- [CC 92] COUSOT P. et COUSOT R., « Inductive Definitions, Semantics and Abstract Interpretation », In *Conference Record of the 19<sup>th</sup> ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages*, pp. 83–94, ACM Press, 1992.
- [DM 93] DERANSART P. et MAŁUSZYŃSKI J., *A Grammatical View of Logic Programming*, MIT Press, 1993.
- [FT 97] FERRAND G. et TESSIER A., « Clarification of the bases of Declarative Diagnoser for CLP », Deliverable D.WP2.1.M1.1-1, Debugging Systems for Constraint Programming (ESPRIT 22532), 1997, version étendue de « Positive and Negative Diagnosis for Constraint Logic Programs in terms of Proof Skeletons », In KAMKAR M., éditeur, *International Workshop on Automated Debugging*, pp. 141–154, 1997.
- [Fag 96] FAGES F., *Programmation Logique par Contraintes*, Cours de l'École Polytechnique, ellipses, 1996.
- [Fag 97] FAGES F., « Constructive negation by pruning », *Journal of Logic Programming*, vol. 32, n° 2:85–118, 1997.
- [JL 87] JAFFAR J. et LASSEZ J.-L., « Constraint Logic Programming », In 14<sup>th</sup> *ACM Symposium on Principles of Programming Languages*, pp. 111–119, 1987.
- [JM 94] JAFFAR J. et MAHER M. J., « Constraint Logic Programming: a survey », *Journal of Logic Programming*, vol. 19-20, :503–581, 1994.
- [LP 88] LEVI G. et PALAMIDESSI C., « Contributions to the Semantics of Logic Perpetual Processes », *Acta Informatica*, vol. 25, :691–711, 1988.
- [Lio 97] LIOGIER K., « Négation Constructive et Modèles Bivalués », In BENHAMOU F., éditeur, *Journées Francophones de Programmation Logique et Programmation par Contraintes*, pp. 85–99, HERMES, 1997.
- [MT 91] MILNER R. et TOFTE M., « Co-induction in relational semantics », *Theoretical Computer Science*, vol. 87, :209–220, 1991.
- [Mah 87] MAHER M. J., « Logic Semantics for a class of Committed-Choice Programs », In *International Conference on Logic Programming*, pp. 858–876, 1987.
- [Mah 93] MAHER M. J., « A Logic Programming view of CLP », In WARREN, éditeur, *International Conference on Logic Programming*, pp. 737–753, MIT Press, 1993.
- [Nai 87] NAIT ABDALLAH M.-A., « Fair derivations in logic programming : operational and greatest fixpoint semantics », *Fundamenta Informaticae*, vol. X, n° 3:247–308, 1987.

- [PS 89] PAULSON L. C. et SMITH A. W., « Logic Programming, Functional Programming, and Inductive Definitions », In *Extensions of Logic Programming*, volume 475 de *Lecture Notes in Computer Science*, pp. 283–309, Springer-Verlag, 1989.
- [Sha 82] SHAPIRO E. Y., *Algorithmic Program Debugging*, ACM Distinguished Dissertation, MIT Press, 1982.
- [Tes 96a] TESSIER A., « Declarative Debugging in Constraint Logic Programming », In JAFFAR J., éditeur, *Asian Computing Science Conference*, volume 1179 de *Lecture Notes in Computer Science*, pp. 64–73, Springer-Verlag, 1996.
- [Tes 96b] TESSIER A., « Une caractérisation des arbres SLD en programmation logique avec contraintes », In DEVIENNE P., éditeur, *actes du pôle Contraintes et Programmation Logique*, pp. 1–11, Journées du GDR Programmation du CNRS, 1996.
- [Tes 97] TESSIER A., Approche, en termes de squelettes de preuve, de la sémantique et du diagnostic d'erreur des programmes logiques avec contraintes, thèse de doctorat, LIFO, Université d'Orléans, 1997.