Sémantique Abstraite et Instrumentale pour Prolog

K. Musumbu
LaBRI (URA 1304 du C.N.R.S.),
Université de Bordeaux I
33405 Talence Cedex, France
e-mail: musumbu@labri.u-bordeaux.fr

Résumé

Le fait de proposer un modèle d'interprétation abstraite nécessite de prouver sa correction par rapport au modèle sémantique standard. Or, la plupart des modèles d'interprétation abstraite définis pour Prolog utilisent des opérations abstraites qui n'ont pas de contrepartie explicite dans la sémantique opérationnelle classique (SLD-resolution). Ce qui rend encore plus difficile la preuve de leur correction. Nous proposons, dans cet article, une sémantique opérationnelle plus proche de ces modèles. Nous prouvons son équivalence avec la sémantique classique et nous l'utilisons ensuite pour prouver la consistance de la sémantique abstraite calculée par les algorithmes présentés dans [7, 8].

Mots clefs: analyse statique, interprétation abstraite, sémantique opérationnelle.

1 Introduction

Le caractère déclaratif de la programmation logique (et, en particulier, de Prolog pur) va de pair avec de nombreuses possibilités d'optimisations. Celles-ci doivent reposer sur des méthodes d'analyse statique automatisables et formellement correctes. L'interprétation abstraite [5] est la principale de ces méthodes. Son application à la programmation logique a été proposée par de nombreux chercheurs (par exemple [2, 12, 13, 4]). Ces travaux proposent des modèles sémantiques dits "abstraits" qui définissent des propriétés globales des programmes (valables pour une infinité d'exécutions différentes). Un modèle sémantique doit être prouvé consistant, c'est-à-dire qu'il ne permet de déduire que des propriétés correctes (mais éventuellement imprécises) des programmes. La preuve de consistance d'un modèle sémantique abstrait doit se faire par rapport à une sémantique de référence (ou standard). En programmation logique la sémantique de référence est appelée SLD-resolution (voir par exemple [11]). Toutefois, pour des raisons d'applicabilité pratique, de nombreux modèles d'interprétation abstraite pour Prolog utilisent des opérations primitives qui n'ont pas de contrepartie explicite dans la sémantique de référence. La principale de ces opérations est appelée *extension* dans [2] et *meet* dans [12]. Cette divergence rend difficile une preuve de consistance directe des modèles d'interprétation abstraite. Dans le but de réaliser de telles preuves de manière simple et convaincante nous proposons dans cet article une sémantique opérationnelle non standard pour Prolog pur[1], qui permet de faire aisémement le lien avec l'opération d'extension. Nous prouvons l'équivalence de cette sémantique avec la SLD-resolution. Enfin, nous l'appliquons à une preuve de consistance de la sémantique abstraite proposée dans [14] sur laquelle se basent les algorithmes d'interprétation abstraite décrits dans [7, 8]. Elle est applicable aussi aux modèles proposés dans [2, 3, 12, 4].

La suite de cet article est organisée comme suit. Le paragraphe 2 contient un rappel de la sémantique de référence de Prolog pur et introduit quelques notations. Au paragraphe 3, nous présentons la sémantique abstraite dont il faut prouver la consistance. Au paragraphe 4, nous définissons la sémantique opérationnelle instrumentale et nous prouvons son équivalence avec celle du paragraphe 2. Enfin, nous énonçons, au paragraphe 5, les principales propriétés de la sémantique instrumentale que nous utilisons ensuite pour prouver la consistance de la sémantique abstraite.

2 Sémantique Opérationnelle de référence(SLD-resolution)

2.1 Notations

Nous supposons connues les notions classiques de variable, terme, atome, clause et procédure logique. Les programmes logiques considérés sont supposés purs : sans littéraux négatifs et sans prédicats extra-logiques. La sémantique opérationnelle de référence associe à toute suite d'atomes SA un ensemble de substitutions (noté $Cas(SA)$). Les notions de

[1] En fait, pour une variante syntaxique de Prolog pur, appelée *ensemble des programmes normalisés*.

substitution (notée σ) et d'unificateur général sont également supposées connues. Nous notons $mgu(t_1, t_2)$ l'ensemble des unificateurs généraux de t_1, t_2. On notera τ les substitutions inversibles (dites aussi de renommage).

2.2 Arbre de dérivation associé à une suite d'atome: $Au(SA)$

Soit SA une suite d'atomes. On suppose fixé un programme de référence.
$Au(SA)$ est défini comme suit:

- Chaque sommet est étiqueté par une suite d'atomes. Chaque arc est étiqueté par une substitution.
- La racine de $Au(SA)$ est étiquetée par SA. Le niveau de la racine est égal à 1.
- Soit un sommet de niveau $i \geq 1$, étiqueté par SA'. Les descendants immédiats de ce sommet sont définis comme suit:
  1. Si SA' est vide, il n'y en a pas.
  2. Si SA' est de la forme $A.SA''$, soient $C_1, \ldots, C_m$ les clauses dont la tête est unifiable à A. Alors, le sommet considéré possède m descendants immédiats(de niveau $i+1$). La suite d'atomes étiquetant le j-ème de ces sommets s'obtient comme suit:

     Soient
     - H_j, SA_j la tête et le corps de C_j
     - τ_j, une substitution de renommage des variables de C_j, au moyen de nouvelles variables,
     - σ_j, un mgu de A et $H_j\tau_j$, n'utilisant pas de variables autres que celles de A et $H_j\tau$.

     Alors, $((SA_j\tau_j)SA'')\sigma_j$ est la suite d'atomes en question.
- Les arcs joignant le sommet considéré à ses descendants directs sont étiquetés par les substitutions σ_j.

2.3 Noeuds de succès et d'échec, Substitution résultat correcte

Un **noeud de succès de** $Au(SA)$ est un sommet étiqueté par une suite d'atomes non vide.
Soit i, le niveau de ce noeud et $\sigma_2, \sigma_3 \ldots, \sigma_i$ les substitutions étiquetant les arcs aboutissant aux ancêtres de ce sommet, de niveaux $2, 3, \ldots, i$ respectivement. On appelle **substitution résultat correcte** associée à ce noeud de succès, la substitution: $(\sigma_2 \ldots \sigma_i)_{/var(SA)}$.
Un **noeud d'échec** est un sommet sans descendant étiqueté par une suite d'atome vide.
$Cas(SA)$ dénote, par définition, l'ensemble des substitutions résultats correctes associées aux noeuds de succès de SA.

3 Sémantique Abstraite

3.1 Programmes normalisés, variables de programme

En pratique, l'interprétation abstraite des programmes logiques est simplifiée si on suppose les programmes mis sous forme *normalisée*. Cette idée a été présentée d'abord dans [2]. Notre notion de programme normalisé diffère de celle de [2] par l'introduction de la notion de *variable de programme*. Nous appellerons *variables standard* et *substitutions standard* les variables et substitutions utilisées par la sémantique de référence. Pour les programmes normalisés, nous postulons l'existence d'une séquence infinie de variables : $x_1, \ldots, x_i, \ldots$ distinctes des variables standard et appelées variables de programme. Un programme normalisé est tel que toute tête de clause est de la forme $p(x_1, \ldots, x_n)$ et tout atome d'une des formes $x_{i_1} = x_{i_2} (i_1 \neq i_2)$, $x_{i_1} = f(x_{i_2}, \ldots, x_{i_n})$ ou $p(x_{i_1}, \ldots, x_{i_n})$ $(n \geq 0)$. Une *substitution de programme*, notée θ, est un ensemble de la forme

$$\{x_{i_1} \leftarrow t_1, \ldots, x_{i_n} \leftarrow t_n\}$$

où les t_i sont des termes standard. On définit comme suit l'application d'une substitution standard σ à une substitution de programme θ : $\theta\sigma = \{x_{i_1} \leftarrow t_1\sigma, \ldots, x_{i_n} \leftarrow t_n\sigma\}$.
On note $Csub_D$ l'ensemble des substitutions de programme de domaine D (ci-dessus: $D = \{x_{i_1}, \ldots, x_{i_n}\}$).

3.2 Substitutions Abstraites, Domaine Abstrait

Nous appellerons *substitutions abstraites* des objets représentant des propriétés de substitutions de programme.

On définit donc un *domaine abstrait* en associant à tout ensemble fini,D de variables de programme un ensemble inductif, qu'on note $Asub_D$. On utilise la lettre β pour dénoter les substitutions abstraites. Formellement, β représente un ensemble de substitutions de programme, noté $Cc(\beta)$. La fonction

$$Cc : Asub_D \rightarrow \mathcal{P}(Csub_D)$$

est appelée fonction de concrétisation.

3.3 Opérations Abstraites

Soit p, une procédure d'un programme normalisé, soit β_{in}, une substitution abstraite d'entrée pour p, décrivant une certaine classe d'appels de p.

Le problème essentiel à résoudre est de déterminer pour chaque clause:

$$p(x_1, \ldots, x_n) \leftarrow B_1, \ldots, B_m.$$

une liste de substitutions abstraites:

$$\beta_0, \beta_1, \ldots, \beta_m$$

représentant les propriétés des variables de la clause aux différents points précédants et suivants chaque atome $B_1, \ldots, B_m$, ce qu'on notera:

$$p(x_1, \ldots, x_n) \leftarrow \beta_0\ B_1\ \beta_1,\ \ldots,\ \beta_{m-1}\ B_m\ \beta_m.$$

Nous allons faire momentanément la supposition que nous disposons d'un oracle qui pour toute substitution abstraite d'entrée et toute procédure nous fournit une substitution abstraite de sortie décrivant correctement les résultats.

Le calcul de β_0, à partir de β_{in}, se fera grâce à une fonction d'*"extension"*:

$$Ext_{D,D'} : Asub_D \rightarrow Asub_{D'}$$

(où $D = \{x_1, \ldots, x_n\}$ et D' est l'ensemble des variables de la clause.)

Ensuite, le calcul de β_i peut se décomposer comme suit: d'abord, calculer une substitution abstraite qui réduit le domaine de β_{i-1} (soit D) aux variables figurant dans B_i (soit D'). Pour cela on devra posséder une fonction de *"restriction"* :

$$Restr_{D,D'} : Asub_D \rightarrow Asub_{D'}$$

Il faut, ensuite, remplacer par $x_1, \ldots, x_n$, les variables de programme $x_{i_1}, \ldots, x_{i_n}$, figurant dans B_i, pour obtenir une substitution abstraite d'entrée pour la procédure correspondant au sous-but. Ceci se fera au moyen d'une fonction de *"changement de variables"* :

$$Chgvar_\gamma : Asub_D \rightarrow Asub_{D'}$$

(où $D = \{x_{i_1}, \ldots, x_{i_n}\}, D' = \{x_1, \ldots, x_n\}$ et $\gamma : D \rightarrow D'$ est une bijection)

L'oracle nous permettra alors de connaître la substitution β_{out} représentant les substitutions concrètes de sortie.

On sait que les instanciations apportées aux variables dans $t_{i_1}, \ldots, t_{i_n}$ lors de l'exécution de B_i seront simultanément propagées dans les autres termes. On devra donc avoir une opération d'extension:

$$GExt_{D,D'} : Asub_D \times Asub_{D'} \rightarrow Asub_D$$

qui permettra de reporter ces instanciations sur les autres variables. Si B_i est un *built-in*, le calcul de β_i pourra se faire selon le même schéma, sauf qu'il est inutile de recourir à l'oracle car on se donnera deux fonctions capables d'exécuter *abstraitement* les unifications. Il y a deux cas possibles: $x_{i_1} = x_{i_2}$ ou $x_{i_1} = f(x_{i_2}, \ldots, x_{i_n})$. Selon le cas, l'oracle sera remplacé par:

$$AI\text{-}Var : Asub_D \rightarrow Asub_D \text{ où } D = \{x_1, x_2\}$$

$$AI\text{-}Func : Asub_D \times F_{n-1} \rightarrow Asub_D \text{ où } D = \{x_1, \ldots, x_n\}$$ [2]

[2] F_n est l'ensemble des foncteurs d'arité n.

3.4 Ensemble de triplets abstraits

Pour compléter cette présentation, il nous reste à résoudre le problème de l'oracle, c'est-à-dire trouver un moyen de le calculer. Remarquons d'abord que cet oracle est équivalent à un ensemble de triplets abstraits de la forme:

$$(\beta_{in}, p, \beta_{out})$$

tel que $\beta_{in}, \beta_{out} \in Asub_D$ et $D = \{x_1, \ldots, x_n\}$ où n est l'arité de p et p un symbole de prédicat figurant dans le programme de référence, et qui à chaque couple (β_{in}, p) formé d'une substitution abstraite d'entrée et d'un nom de procédure, associe la substitution abstraite de sortie correspondante β_{out}. Nous noterons Eta un tel ensemble. On peut remarquer que le procédé d'interprétation abstraite décrit plus haut permet pour un Eta donné de calculer un autre ensemble de triplets abstraits Eta': Soient $C_1, \ldots, C_s$, les clauses de la procédure p. Pour chaque substitution abstraite β_{in} on peut exécuter abstraitement chaque clause $C_1, \ldots, C_s$, ce qui donnera s substitutions abstraites:

$$\beta_1, \ldots, \beta_s$$

On restreint ces substitutions abstraites aux variables de p, grâce à l'opération $Restr_{D,D'}$. Ensuite, on prend la *"réunion"* de ces substitutions grâce à une nouvelle fonction:

$$Union : Asub_D \times \cdots \times Asub_D \rightarrow Asub_D$$

Ce qui donne une autre substitution abstraite de sortie β_{out}. Il est intuitivement évident que si l'oracle (Eta) propose des substitutions abstraites de sortie correctes et si les différentes opérations calculent également des résultats consistants, le nouvel ensemble Eta' de triplets calculés sera aussi consistant. Notre procédé d'interprétation abstraite défini une *transformation d'ensembles de triplets abstraits* et nous allons prouver que tout *point fixe* de cette transformation prédit des propriétés correctes des substitutions de sortie.

3.5 Définition de la Sémantique Abstraite

Formellement, le problème de *"calculer l'oracle"* sera posé comme suit. On définit, à l'aide des opérations abstraites, une transformation:

$$Teta : E\text{-}eta \rightarrow E\text{-}eta,$$

où $E\text{-}eta$ désigne l'ensemble des ensembles de triplets abstraits. La définition de $Teta$ utilise 3 familles de fonctions auxiliaires indicées respectivement par un symbole de prédicat du programme de référence, une clause et une suite de buts:

$$T_p(\circledast, p, \circledast) : Asub_D \times E\text{-}eta \rightarrow Asub_D,$$

$$T_C(\circledast, C, \circledast) : Asub_D \times E\text{-}eta \rightarrow Asub_D,$$

$T_{SB}(\bullet, SB, \bullet) : Asub_D \times E\text{-}eta \rightarrow Asub_D$,

On utilise aussi 4 familles d'opérations abstraites dérivées des opérations identifiées plus haut:

$ExtC(C, \beta) = CExt_{D,D'}(\beta)$
(où D est l'ensemble des variables de la tête et D' de toutes les variables de C),

$RestrC(C, \beta) = Restr_{D',D}(\beta)$,

$ExtB(B, \beta, \beta') = GExt_{D',D''}(\beta, Chgvar_{\gamma^{-1}}(\beta'))$
(où D'' est l'ensemble des variables dans B),

$RestrB(B, \beta) = Chgvar_{\gamma}(Restr_{D',D''}(\beta))$.

La transformation $Teta$ est définie par les équations suivantes:

$Teta(Eta) = \{(\beta, p, \beta') : \beta' = T_p(\beta, p, Eta)\}$

$T_p(\beta, p, Eta) = Union(\beta_1, \ldots, , \beta_n)$,

où $\beta_i = T_C(\beta, C_i, Eta)$,
et $C_i, \ldots, C_n$ sont les clauses de p.

$T_C(\beta, C, Eta) = RestrC(C, \beta')$

où $\beta' = T_{SB}(ExtC(C, \beta), SB, Eta)$,
et SB est le corps de C.

$T_{SB}(\beta, (), Eta) = \beta$,

$T_{SB}(\beta, B.SB, Eta) = T_{SB}(\beta_3, SB, Eta)$

$$
\begin{array}{lll}
avec & \beta_1 = RestrB(B, \beta), & \\
 & \beta_2 = Eta(\beta_1, p) & si\ B\ est\ de\ la\ forme\ p(\cdots), \\
 & \quad = AI\_Var(\beta_1) & si\ B\ est\ de\ la\ forme\ x_{i_1} = x_{i_2}, \\
 & \quad = AI\_Func(\beta_1) & si\ B\ est\ de\ la\ forme\ x_{i_1} = f(\cdots), \\
et & \beta_3 = ExtB(B, \beta, \beta_2). &
\end{array}
$$

La sémantique abstraite du programme est, par définition, le plus petit point fixe de $Teta$. Son existence est assurée si l'on impose que les ensembles $Asub_D$ soient inductifs et que les opérations primitives soient monotones. Cependant, ces propriétés ne sont pas nécessaires pour prouver la *consistance* de la sémantique. Nous ne nous y attarderons donc pas, ici.

4 Sémantique opérationnelle instrumentale

4.1 Motivation

Selon la sémantique opérationnelle de référence toute exécution d'un programme revient à *"poser une question Q"* de la forme:

$$p(t_1, \ldots, t_n)$$

où p est le nom d'une procédure du programme et $t_1, \ldots, t_n$ sont des termes pouvant contenir des variables standard. Les résultats du programme constituent un ensemble de substitutions ayant pour domaine l'ensemble des variables figurant dans $t_1, \ldots, t_n$.
Intuitivement, l'objectif des algorithmes d'interprétation abstraite est de calculer des *"propriétés"* des résultats à partir des certaines suppositions sur la forme de la question. Ces propriétés et suppositions sont formalisées par la notion de substitutions abstraites. Que signifie la *consistance* de la sémantique abstraite? Selon cette sémantique, un programme Prolog définit un ensemble de triplets abstraits: Eta. D'une part, la condition:

$$(\beta, p, \beta') \in Eta$$

signifie: si θ est une substitution qui vérifie la *"propriété"* β et si σ est un résultat pour la question $p\theta$, alors, $\theta\sigma$ vérifie la *"propriété"* β'.

Le fait de vérifier une propriété est formalisé par la fonction de concrétisation Cc. D'autre part, nous sommes convenus de noter $Cas(Q)$ l'ensemble des résultats pour une question Q. Donc, la correction de notre sémantique s'énoncera précisement comme suit:

$$\theta \in Cc(\beta) \text{ et } \sigma \in Cas(p(x_1, \ldots, x_n)\theta) \Rightarrow \theta\sigma \in Cc(\beta').$$

Pour être rigoureux nous baserons notre démonstration sur la sémantique opérationnelle de référence rappelée au paragraphe 2, dans une formulation un peu différente de celle de [11], destinée à fournir une terminologie et des notations utiles pour la suite. Une difficulté de la démonstration vient du fait que la sémantique de référence est assez éloignée du modèle procedural sur lequel est calqué notre sémantique abstraite. L'idée de notre démonstration est de montrer que pour tout nombre k, entier positif, si on limite le nombre d'appels de procédures imbriquées à k au plus, les réponses fournies vérifient les propriétés décrites par la sémantique abstraite. Or, dans la sémantique de référence, la structure des appels de procédures en cours n'apparaît pas. Pour résoudre ce problème, nous proposons maintenant, une sémantique opérationnelle *instrumentale* que nous prouvons équivalente à celle du paragraphe 2. Le paragraphe 5 sera alors consacré à démontrer la consistance de la sémantique abstraite par rapport à la sémantique instrumentale.

Terminons cette section en présentant intuitivement cette sémantique. La sémantique de référence définit un arbre de recherche pour toute question Q. Chaque noeud de cet arbre est étiqueté par une suite d'atomes SA et le calcul des résultats consiste à parcourir cet arbre "en profondeur d'abord" . D'une autre manière, on peut voir, le calcul des résultats

comme un appel de procédures déclenchant d'autres appels de procédures. Supposons que la question initiale soit de la forme $p(x_1, \ldots, x_n)\theta$, et fixons un instant de l'exécution. Selon la première sémantique, on se trouve sur un noeud courant étiqueté par SA. Selon la seconde, on a déjà déclenché, sans les avoir terminés, k appels de procédures: $p_1, p_2, \ldots, p_k$. Pour chacun de ces appels, une clause est en cours d'exécution et il reste à exécuter une suite de sous-buts après exécution du sous-but courant. De plus les variables de la clause sont "instanciées" à certaines valeurs ce qui peut être représenté par des substitutions de programme: $\theta_1, \ldots, \theta_k$. Le lien entre les deux points de vue est que SA peut s'obtenir en concaténant les suites de sous-buts:

$$SB_k\theta_k, SB_{k-1}\theta_{k-1}, \ldots, SB_1\theta_1.$$

Cette suite de suites de sous-buts définit une structuration du noeud courant qui n'est pas prise en compte par la définition traditionnelle de Lloyd. Notre "nouvelle" définition correspond à incorporer cette structure à celle de Lloyd. Le but SA y sera remplacé par un objet de la forme:

$$((SB_k, \theta_k), (SB_{k-1}, \theta), \ldots, (SB_1\theta_1)).$$

que l'on notera SE et qu'on appellera *résolvant.*

4.2 Préliminaires

La sémantique instrumentale est définie pour les programmes normalisés. On appelle *élément de résolvant*, tout couple de la forme (SB, θ) où SB est une suite d'atomes normalisés et θ est une substitution de programme. Un *résolvant* est une suite *non vide* d'éléments de résolvant. On notera E et SE les éléments de résolvant et les résolvants. Un *résolvant vide* est un résolvant de la forme $((), \theta)$.

4.3 Arbre de dérivation associé à un résolvant:$An(SE)$

Soit SE un résolvant. On suppose fixé un programme normalisé de référence.
$An(SE)$ est défini comme suit:

- Chaque sommet est étiqueté par un résolvant est possède un niveau. Les arcs ne sont pas étiquetés.
- La racine de $An(SE)$ est étiquetée par SE. Son niveau est égale á 1.
- Soit un sommet de niveau $i \geq 1$, étiqueté par SE', les descendants immédiats de ce sommet sont définis comme suit:

  1. Si SE' est vide, il n'y en a pas.
  2. Si $SE' = ((), \theta).SE''$ et SE'' est un résolvant (suite non vide),
     il y a un seul successeur, étiqueté par SE'', son niveau est i (et non $i - 1$).

3. Si $SE' = (B.SB, \theta).SE''$ et SE'' est un résolvant ou une suite vide, il y a deux sous-cas, à envisager.

   (a) B est un "built-in" de la forme $t_1 = t_2$.
   Si $t_1\theta$ et $t_2\theta$ ne sont pas unifiables, il n'y a pas de descendants.
   Sinon, soit σ un mgu de t_1, t_2 n'utilisant que des variables de t_1, t_2. Il y a un seul successeur, de niveau $i+1$, étiqueté par $((SB, \theta).SE'')\sigma$ où la notation $SE\sigma$ est définie par:
   $()\sigma = (); ((SB, \theta).SE)\sigma = (SB, \theta\sigma).SE\sigma$.

   (b) B est un atome de la forme $p(x_{i_1}, \ldots, x_{i_n})$.
   Soient $C_1, \ldots, C_m$, les clauses de p. Il y a m successeurs. Le j-ème d'entre eux est étiqueté par:
   $(SB_j, \theta_j).(SB, \theta).SE''$,
   où SB_j est le corps de C_j et
   $\theta_j = \{x_1 \leftarrow xi_1\theta, \ldots, x_n \leftarrow x_{i_n}\theta, x_{n+1} \leftarrow y_1, \ldots, x_{n+p} \leftarrow y_p\}$
   $var(C_j) = x_1, \ldots, x_{n+p}$,
   $y_1, \ldots, y_p$ sont des variables de renommage fraîches.
   Tous ces successeurs sont de niveau $i+1$.

4.4 Définitions

Un *noeud de succès de* $An(SE)$ est un sommet étiqueté par un résolvant vide, soit $((), \theta)$. θ est la *substitution résultat* associée à ce noeud de succès. Un *noeud d'échec* est un sommet sans descendant étiqueté par un résolvant non vide. On note $Cas(SE)$, l'ensemble des substitutions résultats de $An(SE)$.

4.5 Arbre de dérivation limité associé à un résolvant: $An_k(SE)$ $(k \geq 1)$

On définit $An_k(SE)$ comme $An(SE)$ avec la modification suivante:
"Si SE' contient k éléments, il n'y a pas de successeurs."

4.6 Propriétés

On démontre aisément les propriétés suivantes:

1. $An_k(SE)$ est fini, pour tout k.
2. Soit A un arbre. On appelle souche de A, tout arbre A' tel que:
   - la racine de A' est celle de A;
   - si $s \in A'$, on a aussi $s \in A$ et le chemin de la racine à s est le même dans A et dans A'.

   On a :

- $An_k(SE)$ est une souche de $An(SE)$,
- $An_k(SE)$ est une souche de $An_{k'}(SE)$, si $k \leq k'$.

3. $Cas_k(SE) \subseteq Cas_{k'}(SE) \subseteq Cas(SE)$ (si $1 \leq k \leq k'$).
4. $Cas(SE) = \bigcup_{k=1}^{\infty} Cas_k(SE)$.

4.7 Équivalence des deux sémantiques opérationnelles

4.7.1 Définition: *Valeur d'un résolvant* ($v(SE)$)

On définit par induction sur la structure des résolvants, la fonction $v(SE)$ qui donne la suite d'atomes représentée par le résolvant:

$$v((SB, \theta)) = SB\theta,$$

$$v((SB, \theta).SE) = SB\theta.v(SE),$$

4.7.2 Lemme

Il existe une fonction surjective de l'ensemble des sommets de $An(SB, \theta)$ dans ceux de $Au(SA)$, vérifiant la propriété suivante:
Pour tout couple(N,N') de sommets mis en correspondance:

- N et N' ont même niveau,
- si N est étiqueté par SE, N' l'est par $v(SE)$,
- si i est le niveau de N et N',

  $SE = SE'.(SB'\theta')$,

  $\sigma_2, \sigma_3 \ldots, \sigma_i$ sont les substitutions étiquetant les arcs menant de la racine à N',

  alors, $\theta' = \theta\sigma_2\sigma_3 \ldots \sigma_i$.

démonstration Soit i, un entier quelconque, on montre par récurrence sur i, qu'une telle fonction existe pour les sommets de niveau i. On trouvera une démonstration complète dans [14].

4.7.3 Théorème

Supposons fixé un programme de référence normalisé. Soient SA, une suite d'atomes ne contenant que des variables standard, SB, une suite d'atomes normalisés, θ, une substitution de programme telle que $SA = SB\theta$ et $dom(\theta) = var(SB)$.
On a:

$$Cas(SA) = \{\sigma : dom(\sigma) \subseteq var(SA) \text{ et } \theta\sigma \in Cas(SB, \theta)\},$$

$Cas(SB, \theta) = \{\theta\sigma : \sigma \in Cas(SA)\}$.

démonstration Conséquence du lemme (voir [14]).

5 Consistance de la sémantique abstraite

Par induction sur k,nous démontrons, que toutes les substitutions résultats figurant dans un arbre de dérivation $An_k(p(x_1, \ldots, x_n), \theta)$ vérifient la propriété $Eta(\beta, p)$ si θ vérifie β et si Eta est un point fixe de $Teta$. Pour cela il est utile d'établir d'abord un certain nombre de propriétes de la sémantique opérationnelle. Nous ne pouvons qu' énoncer ces propriétés, sans démonstration (faute de place). Les preuves complètes figurent dans [14].

5.1 Conventions

1. Soient E et E' deux ensembles de substitutions. Nous les considérons comme égaux et notons $E = E'$ si et seulement si pour toute substitution $\sigma \in E$ il existe une substitution inversible τ telle que $\sigma\tau \in E'$ et réciproquement.

2. On définit les trois nouvelles notations suivantes:

$$\begin{array}{lcl} Cas_k(SE) & = & \{\theta : ((), \theta) \text{ est un sommet de } An_k(SE)\}, \\ Cas_k(SB, \theta) & = & Cas_k((SB, \theta)), \\ Cas_k(p, \theta) & = & Cas_k(p(x_1, \ldots, x_n), \theta). \end{array}$$

5.2 Propriétés de la sémantique instrumentale

1. Soient $k \geq 2$ et SE, un résolvant de longueur m $(1 \leq k < m)$. On a :

$$Cas_k((SB, \theta).SE) = \bigcup_{\theta\sigma \in Cas_{k-m}(SB,\theta)} Cas_k(SE\sigma).$$

2. Soient B et SB, atome et suite d'atomes normalisés, θ telle que $var(B.SB) \subseteq dom(\theta)$. On a:

$$Cas_k(B.SB, \theta) = \bigcup_{\theta' \in Cas_k(B,\theta)} Cas_k(SB, \theta').$$

3. Soient B, un atome de la forme $p(x_{i_1}, \ldots, x_{i_n})$, θ une substitution telle que $dom(\theta) = \{x_1, \ldots, x_m\} = D$, θ' une substitution telle que $dom(\theta') = \{x_{i_1}, \ldots, x_{i_n}\} = D' \subseteq D$, on définit:

$$\begin{array}{lcl} RestrB(B, \theta) & = & \{x_1 \leftarrow x_{i_1}\theta, \ldots, x_n \leftarrow x_{i_n}\theta\}, \\ ExtB(B, \theta, \theta') & = & \theta\sigma \\ & & \text{où} \left\{ \begin{array}{l} dom(\sigma) \subseteq codom(\theta_{/D'}), \\ codom(\sigma) \bigcap (codom(\theta) - codom(\theta_{/D'})) = \{\}, \\ (\theta\sigma)_{/D'} = \{\ldots, x_{i_j} \leftarrow x_j\theta', \ldots\} \end{array} \right. \end{array}$$

Avec ces notations:

$$Cas_k(B,\theta) = \{ExtB(B,\theta,\theta') : \theta' \in Cas_k(p, RestrB(B,\theta))\}.$$

4. Soient C, une clause de la forme: $p(x_1,\ldots,x_n) \leftarrow SB$ et θ telle que $dom(\theta) = \{x_1,\ldots,x_n\}$. On définit:

$$ExtC(C,\theta) = \{\ldots, x_i \leftarrow x_i\theta, \ldots, x_{n+i} \leftarrow y_i, \ldots\}$$

où $var(C) = \{\ldots, x_i, \ldots, x_{n+i}, \ldots\}$ et les variables y_i sont fraîches et distinctes.
Soit θ telle que $dom(\theta) = var(C)$. On définit:

$$RestrC(C,\theta) = \{x_1 \leftarrow x_1\theta, \ldots, x_n \leftarrow x_n\theta\}.$$

On pose aussi: $\begin{cases} Cas_0(C,\theta) = \{\}, \\ Cas_k(C,\theta) = \{RestrC(C,\theta') : \theta' \in Cas_k(SB, ExtC(C,\theta))\}\ (k \geq 1). \end{cases}$

Avec ces notations, on a:

$$Cas_k(p,\theta) = \bigcup_{j=1}^{m} Cas_{k-1}(Cj,\theta)\ (k \geq 1),$$

où les C_j sont les clauses de p.

5.3 Consistance des opérations abstraites

La consistance de la sémantique abstraite repose sur celle des opérations primitives définies au paragraphe 3.5. De façon générale, une opération abstraite

$$o_a : Asub_{D_1} \times \ldots \times Absub_{D_n} \rightarrow Asub_D$$

est consistante par rapport à l'opération concrète

$$o_c : Csub_{D_1} \times \ldots \times Cbsub_{D_n} \rightarrow Csub_D$$

si et seulement si on a :

$$(\forall i : 1 \leq i \leq n : \theta_i \in Cc(\beta_i)) \Rightarrow o_c(\theta_1,\ldots,\theta_n) \in Cc(o_a(\beta_1,\ldots,\beta_n))$$

(pour tous $\theta_i \in Csub_{D_i}$ et $\beta_i \in Asub_{D_i}$).

5.3.1 Théorème

Si Eta est un point fixe de $Teta$, on a, avec les notations habituelles:

$$\begin{array}{rcl} \theta \in Cc(\beta) & \Rightarrow & Cas_k(p,\theta) \subseteq Cc(Tp(\beta,p,Eta)), \\ & & Cas_k(C,\theta) \subseteq Cc(TC(\beta,C,Eta)), \\ & & Cas_k(SB,\theta) \subseteq Cc(TSB(\beta,SB,Eta)). \end{array}$$

démonstration: (par induction sur k)
La démonstration complète se trouve dans [14]

5.4 Corollaire

Avec les notations du théorème précédent, on a:

$$Cas_k(p,\theta) \subseteq Cc(Eta(\beta,p)).$$

démonstration : conséquence immédiate du théorème précédent et du fait que Eta est un point fixe de $Teta$.

5.5 Corollaire

Sous les mêmes conditions, on a:

$$Cas(p,\theta) \subseteq Cc(Eta(\beta,p)).$$

démonstration : d'après la propriété 4.6.4.

5.6 Théorème

Soient p, une procedure normalisée et θ, une substitution de programme. On a:

$$\theta \in Cc(\beta) \text{ et } \sigma \in Cas(p(x_1,\ldots,x_n)\theta) \Rightarrow \theta\sigma \in Cc(Eta(\beta,p))$$

(lorsque Eta est un point fixe de $Teta$).
démonstration :

$\sigma \in Cas(p(x_1,\ldots,x_n)\theta)$

$\Rightarrow \quad \theta\sigma \in Cas(p,\theta)$ *(équivalence des deux semantiques*

$\Rightarrow \quad \theta\sigma \in Cc(Eta(\beta,p))$ *(corollaire ci – dessus).*

6 Conclusion

Nous avons proposé dans cet article une sémantique opérationnelle instrumentale permettant de justifier simplement et rigoureusement les modèles sémantiques d'interprétation abstraite de Prolog. Nous l'avons prouvée équivalente à la sémantique opérationnelle de référence [11] et l'avons utilisée pour prouver la consistance d'un modèle sémantique d'interprétation abstraite (utilisé par [7, 8]). Elle est aussi applicable aux modèles proposés dans [2, 4, 12]. L'abondance des recherches actuelles en analyse statique des programmes logiques conduira à définir de nouveaux modèles plus élaborés. Ces nouveaux modèles pourront être justifiés en suivant la même démarche.

7 Remerciements

Je tiens à remercie Baudouin Le Charlier pour les remarques pertinentes et les nombreuses discusions que nous avons eues sur ce sujet et d'avoir bien voulu relire le papier. La qualité finale de ce papier lui doit beaucoup.

References

[2] **M. Bruynooghe et al.** *Abstract interpretation: Towards the global optimization of logic programs*, In Proc. 1987 Symp. on Logic Programming, IEEE Society Press, pp 192-204.

[3] **M. Bruynooghe** *A practical framework for the abstract interpretation of logic programs*, Journal of Logic Programming, 1991.

[4] **P. Codognet, G. File'** *Computations, abstractions and constraints for Logic Programs*, In Proc ICCL 92, San Francisco, April 1992.

[5] **P. Cousot, R. Cousot** *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction of Approximation of Fixpoints*, POPL 1977, Sigact Sigplan, pp 238–252.

[7] **B. Le Charlier, K. Musumbu, P. Van Hentenryck** *Efficient and accurate algorithms for the Abstract Interpretation of Prolog Programs*, Rapport interne, Institut d'Informatique, F.U.N.D.P. Namur, Août 1990.

[8] **B. Le Charlier, K. Musumbu, P. Van Hentenryck** *A generic abstract interpretation algorithm and its complexity analysis*, In Proc ICLP 91, June 91.

[10] **C. Livercy** *Théorie des programmes: Schémas, Preuves, Sémantique*, Dunod Informatique, Paris, 1978.

[11] **J.W. Lloyd** *Foundation of Logic Programming*, Springer Verlag, 1987.

[12] **K. Marriott and H. Sondergaard** *Notes for a Tutorial on Abstract Interpretation of Logic Programs*, NACLP 89, Cleveland, 1989.

[13] **C.S. Mellish** *Abstract Interpretation of Prolog Programs*, In Abstract Interpretation of Declarative Languages, Ellis Horwood Limited, 1987.

[14] **K. Musumbu** *Interprétation abstraite des programmes Prolog* , Thèse de doctorat (Phd), Institut d'Informatique, F.U.N.D.P. Namur, sept. 90.