Sémantique des programmes logiques avec contraintes fondée sur la relation de couverture

Gérard Ferrand et Alexandre Tessier

LIFO – Université d'Orléans – BP 6759 – 45067 Orléans Cedex 2

{ferrand,tessier}@lifo.univ-orleans.fr

ftp://ftp-lifo.univ-orleans.fr/pub/Users/tessier/

Résumé: Il n'est pas possible d'adapter simplement à la programmation logique avec contraintes les techniques de diagnostic déclaratif d'erreurs connues pour la programmation logique classique. De nouveaux fondements théoriques sont nécessaires. Il n'y a plus comme en programmation logique couverture d'une réponse par une réponse calculée plus générale. La relation de couverture pallie cette absence. Toute réponse est couverte par un ensemble (éventuellement infini) de réponses calculées.

L'article redéfinit la sémantique déclarative des programmes logiques avec contraintes en terme d'arbres de preuve utilisant une relation de couverture. [?, ?] montrent l'intérêt d'un cadre basé sur une relation abstraite de conséquence ("entailment" en anglais) plutôt que sur un domaine ou une théorie. La relation de couverture généralise la relation de conséquence.

Les arbres de preuve permettent de donner à la notion de réponse une définition intrinsèque indépendante de toute règle de calcul. La relation de couverture permet de capturer l'idée qu'une contrainte est couverte par un ensemble (éventuellement infini) de contraintes. Les arbres de preuve sont construits à partir de règles provenant soit des clauses du programme soit de la relation de couverture.

1 Introduction

La motivation de ce travail est le diagnostic déclaratif d'erreurs en programmation logique avec contraintes. Le diagnostic déclaratif d'erreurs en programmation logique fut introduit par E. Y. Shapiro [?] sous le nom de mise au point algorithmique. D'autres techniques, développées dans [?, ?, ?, ?, ?, ?, ?, ?] se sont inspirées de la méthode de Shapiro.

Rappelons brièvement quelques unes des différences entre la programmation logique classique et la programmation logique avec contraintes qui font que de nouveaux fondements théoriques sont nécessaires pour étudier le diagnostic déclaratif d'erreurs.

En programmation logique avec contraintes les interprétations de Herbrand ne représentent plus la sémantique des programmes. Lors de la mise au point on souhaite rester dans le langage du programme, mais certains éléments du domaine ne sont pas exprimables de façon finie dans le langage. Par exemple, le nombre π en CLP($\mathcal{R}$) [?], ou encore, l'ensemble des entiers formels pour un solveur sur les ensembles [?].

En programmation logique, chaque réponse qui est une conséquence du programme est couverte par *une* réponse calculée plus générale. Il n'y a plus couverture unique en programmation logique avec contraintes. Par exemple, en CLP($\mathcal{R}$), le programme:

$$p(x) \leftarrow x < 0$$
$$p(x) \leftarrow x \geq 0$$

a pour conséquence $true \rightarrow p(x)$, pourtant il n'existe pas de réponse calculée la plus générale. Néanmoins, lors du diagnostic déclaratif d'erreurs, on ne peut pas considérer que $true \rightarrow p(x)$ est un symptôme d'insuffisance car cette réponse est couverte par les deux réponses calculées $x < 0 \rightarrow p(x)$ et $x \geq 0 \rightarrow p(x)$. Une condition suffisante pour la propriété de couverture unique est l'indépendance des contraintes négatives (INC)

Journées du GDR Programmation.
22, 23 et 24 novembre 1995.
Grenoble.

[?, ?] qui n'est malheureusement vérifiée que par peu de domaines de contraintes intéressants. Parfois, en programmation logique avec contraintes, il n'y a même pas couverture finie si la sémantique des programmes est définie non pas par une théorie mais par un modèle [?].

Les algorithmes de mise au point déclarative habituellement proposés en programmation logique utilisent fortement ces propriétés des modèles de Herbrand. On ne peut donc pas adapter simplement les méthodes classiques à la programmation logique avec contraintes.

L'approche de la sémantique des programmes logiques avec contraintes en terme d'arbres de preuve et de squelettes, paramétrée par un critère de rejet [?], est une extension de la vision grammaticale de la programmation logique introduite par P. Deransart et J. Maluszinski [?]. La relation entre la sémantique déclarative et la sémantique opérationnelle est ainsi mieux expliquée. La notion de domaine ou de théorie, habituelle pour la sémantique des programmes logiques avec contraintes [?, ?, ?, ?, ?, ?], devient un paramètre du critère de rejet. Le rôle du critère de rejet est d'éliminer certaines réponses dont la contrainte n'est jamais satisfaite.

Cette sémantique permet de rendre compte de l'incomplétude de certains solveurs de contraintes qui ne peut pas s'exprimer en terme de logique. Par exemple, le solveur de contraintes de CLP($\mathcal{R}$) fournit trois types de réponses: `yes` la contrainte est alors satisfiable, `no` la contrainte est alors insatisfiable et `maybe` on ne peut pas alors statuer sur la satisfiabilité de la contrainte. Le solveur répond `yes` pour $x \times x = 1 \wedge x = 1$ et répond `maybe` pour $x \times x = 1$, néanmoins $\models \exists x(x \times x = 1 \wedge x = 1) \rightarrow \exists x(x \times x = 1)$; le solveur répond `no` pour $x = 1 \wedge x = 0$ et répond `maybe` pour $x \times x = 1 \wedge x \times x = 0$, néanmoins $\models \neg\exists x(x = 1 \wedge x = 0) \rightarrow \neg\exists x(x \times x = 1 \wedge x \times x = 0)$.

Le diagnostic déclaratif d'incorrection a pu être traité dans ce cadre [?] mais l'abstraction introduite par le critère de rejet ne permet pas d'exprimer la notion de couverture des réponses par des réponses calculées qui est nécessaire pour le diagnostic d'insuffisance. Pour cette raison nous introduisons une relation de couverture entre une contrainte c et un ensemble de contraintes C , noté $c \vdash C$ et lue: c est couverte par C. On retrouve le critère de rejet quand l'ensemble C est vide et la relation de conséquence proposée à l'origine par V. Saraswat [?] quand l'ensemble C est un singleton. Nous généralisons ces deux notions et retrouvons les résultats connus.

Cet article est organisé comme suit: dans la section ?? nous définissons les programmes logiques avec contraintes après avoir introduit le langage des programmes et quelques notations; la section ?? présente la relation de couverture et ses propriétés; les arbres de preuve sont définis dans la section ?? qui fixe la sémantique des programmes logiques avec contraintes, on y trouvera également un résultat essentiel exprimant le lien entre la sémantique opérationnelle abstraite et la sémantique déclarative. L'article se termine par une conclusion récapitulant les idées principales.

2 Terminologie, notations

Considérons une fois pour toute quatre ensembles qui déterminent le langage du programme:

- un ensemble infini de *variables* V ;
- un ensemble de *symboles de fonctions* Σ ;
- un ensemble de *symboles de prédicats de contraintes* Π_c (ces prédicats sont dit *pré-définis*) ;
- un ensemble de *symboles de prédicats définis par programme* Π_p (ces prédicats sont appelés *prédicats de programme*).

L'ensemble des *termes* est construit, comme d'habitude, sur V et Σ. Les formules atomiques du langage du premier ordre construit sur V, Σ et Π_p de la forme $p(x_1, \ldots, x_n)$, où p est un prédicat de programme d'arité n et $x_1, \ldots, x_n$ sont n variables distinctes, sont appelées *atomes*. Le langage des contraintes CONST est un sous-ensemble du langage du premier ordre construit à partir de V, Σ et Π_c. On suppose qu'il est clos par conjonction et quantification existentielle. Une *contrainte* est une formule de CONST.

Notations :

- $\tilde{x}$ désigne une séquence de variables distinctes $x_1, \ldots, x_n$;

- si c est une contrainte et $\tilde{x}$ une séquence de variables $x_1, \ldots, x_n$ alors $\exists_{\tilde{x}}\, c$ désigne la contrainte $\exists x_1 \cdots \exists x_n\, c$;
- si c est une contrainte et a est un atome alors $\exists_{-a}\, c$ est la contrainte $\exists_{\tilde{x}}\, c$, où $\tilde{x}$ sont les variables libres de c qui n'apparaissent pas dans a.

Une *clause* est un $n+2$-uplet ($0 \leq n$) noté $a_0 \leftarrow c \,\square\, a_1, \ldots, a_n$, où les a_i sont des atomes et c est une contrainte. Quand $n = 0$ la clause est appelée un *fait*. Ces définitions sont identiques à celles de [?, ?, ?, ?, ?] et se motivent de la même façon.

Un *but* est une clause sans tête de la forme $\leftarrow c \,\square\, a_1, \ldots, a_n$. Comme dans [?], pour formaliser la notion de réponse à un but, on peut se ramener à une *question* sur un atome, de la forme $\leftarrow a$, en introduisant une clause fictive dont le corps est le but et la tête l'atome.

Un *programme* est un ensemble de clauses.

Un *atome contraint* est un couple noté $a[c]$ où a est un atome et c est une contrainte.

3 Relation de couverture

Une *relation de couverture* $\vdash$ est une relation binaire sur $\mathrm{CONST} \times 2^{\mathrm{CONST}}$. Elle a, pour toute contrainte c, les trois propriétés suivantes :

1. si il existe n contraintes $c_1, \ldots, c_n$ et n ensembles de contraintes $C_1, \ldots, C_n$ tels que $c_i \vdash C_i$, pour tout $i = 1, \ldots, n$, alors $c \wedge c_1 \wedge \cdots \wedge c_n \vdash \{c \wedge c'_1 \wedge \cdots \wedge c'_n \mid c'_i \in C_i, \text{ pour tout } i = 1, \ldots, n\}$;
   (cas particulier si $n = 0$ on en déduit $c \vdash \{c\}$)
2. si il existe un ensemble de contraintes C tel que $c \vdash C$ alors $\exists_{\tilde{x}}\, c \vdash \{\exists_{\tilde{x}}\, c' \mid c' \in C\}$, pour toute séquence de variables $\tilde{x}$;
3. si il existe un ensemble de contraintes C tel que $c \vdash C$ et si il existe une famille d'ensembles de contraintes $(C'_{c'})_{c' \in C}$ telle que $c' \vdash C'_{c'}$, pour toute contrainte $c' \in C$, alors $c \vdash \bigcup_{c' \in C} C'_{c'}$;

On définit ainsi un système de contraintes $(\mathrm{CONST}, \vdash)$ en un sens analogue à [?].

La relation de couverture sera le plus souvent définie par :

- un domaine $\mathcal{D}$, elle est notée $\vdash_{\mathcal{D}}$, dans ce cas $c \vdash_{\mathcal{D}} C$ si pour toute valuation dans $\mathcal{D}$ qui satisfait c il existe une contrainte de C satisfaite par cette valuation ;
- une théorie $\mathcal{T}$ (non nécessairement "satisfaction-complète"), elle est notée $\vdash_{\mathcal{T}}$, dans ce cas $c \vdash_{\mathcal{T}} C$ si pour tout modèle $\mathcal{D}$ de $\mathcal{T}$ $c \vdash_{\mathcal{D}} C$;
- un algorithme de satisfaction de contrainte $\mathcal{A}$, elle est notée $\vdash_{\mathcal{A}}$ et définie par la plus petite relation verifiant les trois propriétés et telle que $c \vdash_{\mathcal{A}} \emptyset$ si l'algorithme répond non pour la satisfiabilité de c.
- un critère de rejet RC [?], elle est notée $\vdash_{RC}$ et définie par la plus petite relation verifiant les trois propriétés et telle que $c \vdash_{RC} \emptyset$ si le critère de rejet RC rejette c.

4 Arbres de preuve

Dans cette section nous supposons qu'un programme P et qu'une relation de couverture $\vdash$ sont fixés.

Nous nous dotons de deux types de règles qui permettent de construire des *arbres de preuve* [?]. Les premières sont issues des clauses du programme et les secondes sont définies par la relation de couverture.

Définition 4.1 Règles de programme et règles de couverture

Pour toute clause renommée $a \leftarrow c \,\square\, a_1, \ldots, a_n$ du programme et toutes contraintes $c_1, \ldots, c_n$, nous avons la règle de programme :

$$\frac{a_1[c_1] \;\; \cdots \;\; a_n[c_n]}{a[\exists_{-a}\, c \wedge c_1 \wedge \cdots \wedge c_n]}$$

(cas particulier si $n = 0$ on en déduit l'axiome $\frac{}{a[\exists_{-a}\, c]}$)

Pour tout atome a, toute contrainte c et tout ensemble de contraintes C tels que $c \vdash C$, nous avons la règle de couverture :

$$\frac{\{a[c'] \mid c' \in C\}}{a[c]}$$

(cas particulier si $C = \emptyset$ on en déduit l'axiome $\frac{}{a[c]}$)

Ces règles fournissent la notion classique d'arbres de preuve. Un arbre de preuve avec hypothèses est appelé *arbre de preuve partiel*, les autres sont appelés *arbre de preuve complet*.

Si il existe un arbre de preuve (partiel ou complet) dont la conclusion est l'atome contraint $a[c]$ et les hypothèses constituent l'ensemble d'atomes contraints A, alors nous notons en résumé $A \rightsquigarrow_{\vdash} a[c]$.

Si $\emptyset \rightsquigarrow_{\vdash} a[c]$ alors $a[c]$ est appelé une *réponse* à la question $\leftarrow a$ selon $\vdash$. Cette définition est équivalente à celle donnée dans [?] si $\vdash$ est définie par un domaine $\mathcal{D}$.

Remarque : si $c \vdash \emptyset$ alors $\emptyset \rightsquigarrow_{\vdash} a[c]$, pour tout atome a. $a[c]$ est alors appelé une *réponse triviale* à la question $\leftarrow a$ selon $\vdash$. Le système cherchera, si possible, à éliminer ces réponses triviales qui sont indépendantes du programme et sans intérêt pour l'utilisateur. Exemple en CLP($\mathcal{R}$) : soit le programme P

$$\begin{array}{l} p(x,y) \leftarrow x > y \ \square \\ q(x,y) \leftarrow v = x \times x \wedge w = y \times y \ \square \ p(v,w) \\ r(x) \leftarrow x = y \ \square \ p(x,y) \\ r(x) \leftarrow x = y \ \square \ q(x,y) \end{array}$$

pour la question $\leftarrow r(x)$ nous obtenons par exemple les réponses $r(x)[x > x]$ et $r(x)[x \times x > x \times x]$. Relativement au domaine $\mathcal{R}$, les deux contraintes de ces réponses sont insatisfiables. Le système CLP($\mathcal{R}$) élimine bien la première, par contre il n'élimine pas la seconde mais la qualifie de "`maybe`" (le solveur est incomplet).

Si il existe un arbre de preuve utilisant seulement des règles de programme dont la conclusion est $a[c]$ et dont les hypothèses sont A, alors nous notons en résumé $A \rightsquigarrow a[c]$.

Si $\emptyset \rightsquigarrow a[c]$ alors $a[c]$ est appelé une *réponse calculée* à la question $\leftarrow a$.

Lemme 4.1 Correction/complétude des réponses selon $\vdash_{\mathcal{D}}$ et $\vdash_{\mathcal{T}}$

$a[c]$ est une réponse à la question $\leftarrow a$ selon $\vdash_{\mathcal{D}}$ si et seulement si $P \models_{\mathcal{D}} c \rightarrow a$ ($P \models_{\mathcal{D}} F$ signifie F est vraie dans tout $\mathcal{D}$-modèle de P).

$a[c]$ est une réponse à la question $\leftarrow a$ selon $\vdash_{\mathcal{T}}$ si et seulement si $P, \mathcal{T} \models c \rightarrow a$.

Preuve du lemme 4.1 Voir les théorèmes 4.2, 4.3, 4.8 et 4.9 de [?]. ■

Les arbres de preuve n'utilisant que des règles de programme correspondent aux réponses calculées (grâce à la notion de squelette [?]).

Définition 4.2 Ensembles succès

L'ensemble succès associé au programme P est

$$SS(P) = \{a[c] \mid \emptyset \rightsquigarrow a[c]\}$$

L'ensemble succès associé au programme P et à une relation de couverture $\vdash$ est

$$SS_{\vdash}(P) = \{a[c] \mid \emptyset \rightsquigarrow_{\vdash} a[c]\}$$

$SS(P)$ (resp. $SS_{\vdash}(P)$) est l'ensemble défini inductivement par les règles de programme (resp. par les règles de programme et les règles de couverture).

Remarque : si l'on considère le sous-ensemble $SS_{\vdash\emptyset}(P)$ de $SS(P)$ défini par $SS_{\vdash\emptyset}(P) = \{a[c] \mid a[c] \in SS(P)$ et non$(c \vdash \emptyset)\}$ alors $SS_{\vdash_{RC}\emptyset}(P) = SS_{RC}(P)$ de [?], $SS_{\vdash_{\mathcal{T}}\emptyset}(P) = SS(P, \mathcal{T})$ de [?], $SS_{\vdash_{\mathcal{D}}\emptyset}(P) = lfp(S_P^{\mathcal{D}})$ de [?], $SS_{\vdash_{\mathcal{D}}\emptyset}(P) = SS_3(P, \mathcal{D})$ de [?] (voir la preuve du théorème 5.2 de [?] et les définitions de [?, ?, ?, ?]).

Définition 4.3 Opérateurs de conséquence immédiate

Soit T_P, $T_\vdash$ et $T_{P,\vdash}$ les trois opérateurs de conséquences immédiates (des ensembles d'atomes contraints dans les ensembles d'atomes contraints), associés aux règles de programmes et de couvertures, définis par :

- $T_P(I) = \{a[c] \mid$ *il existe une règle de programme* $\dfrac{a_1[c_1], \ldots, a_n[c_n]}{a[c]}$ *telle que* $a_i[c_i] \in I$, *pour tout* $i = 1, \ldots, n\}$
- $T_\vdash(I) = \{a[c] \mid$ *il existe une règle de couverture* $\dfrac{\{a[c'] \mid c' \in C\}}{a[c]}$ *telle que* $a[c'] \in I$, *pour toute contrainte* $c' \in C\}$
- $T_{P,\vdash}(I) = T_P(I) \cup T_\vdash(I)$

Lemme 4.2 Plus petit point fixe

1. $pppf(T_P) = T_P \uparrow \omega = SS(P)$.
2. $pppf(T_{P,\vdash}) = SS_\vdash(P)$.

Preuve du lemme 4.2 D'après le théorème de Knaster-Tarski. ■

Lemme 4.3 Correction/complétude des réponses selon $\vdash$

$SS_\vdash(P) = \{a[c] \mid$ *il existe* C *tel que* $c \vdash C$, *et* $a[c'] \in SS(P)$, *pour toute contrainte* $c' \in C\}$

Corollaire 4.1

1. $pppf(T_{P,\vdash}) = T_\vdash(pppf(T_P)) = T_\vdash(SS(P)) = SS_\vdash(P)$.
2. $pppf(T_{P,\vdash}) = T_{P,\vdash} \uparrow \omega + 1$.
3. *Pour tout atome contraint* $a[c]$, *si* $\emptyset \rightsquigarrow_\vdash a[c]$ *alors il existe un ensemble de contraintes* C *tel que* $c \vdash C$ *et* $\emptyset \rightsquigarrow a[c']$, *pour toute contrainte* $c' \in C$.

   C'est-à-dire, pour tout arbre de preuve complet, il existe un arbre de preuve complet de même conclusion qui utilise exactement une règle de couverture, et cette règle de couverture est utilisée à la racine de l'arbre de preuve.

   On retrouve ainsi les résultats bien connus dans le cas où $\vdash$ *est déduite d'un domaine* $\mathcal{D}$ *ou d'une théorie* $\mathcal{T}$.

Preuve du lemme 4.3 On prouve le lemme par induction sur les règles en distinguant les deux cas associés aux deux types de règles.

1. Soit $\dfrac{a_1[c_1] \ \cdots \ a_n[c_n]}{a[\exists_{-a}\, c \wedge c_1 \wedge \cdots \wedge c_n]}$ une règle de programme.

   Supposons que, pour tout $i = 1, \ldots, n$, $a_i[c_i] \in SS_\vdash(P)$ et il existe un ensemble de contraintes C_i tel que $c_i \vdash C_i$ et $a_i[c_i'] \in SS(P)$, pour toute contrainte $c_i' \in C_i$.

   Montrons qu'alors il existe un ensemble de contraintes C tel que $\exists_{-a}\, c \wedge c_1 \wedge \cdots \wedge c_n \vdash C$ et $a[c'] \in SS(P)$, pour toute $c' \in C$.

   La règle de programme est issue de la clause renommée $a \leftarrow c \,\square\, a_1, \ldots, a_n$.

   Donc, pour tout n-uplet de contraintes $c_1', \ldots, c_n'$ tel que $c_i' \in C_i$, pour tout $i = 1, \ldots, n$, on a la règle de programme $\dfrac{a_1[c_1'] \ \cdots \ a_n[c_n']}{a[\exists_{-a}\, c \wedge c_1' \wedge \cdots \wedge c_n']}$, par conséquent $a[\exists_{-a}\, c \wedge c_1' \wedge \cdots \wedge c_n'] \in SS(P)$.

   La propriété 1 de la relation de couverture assure que

   $$c \wedge c_1 \wedge \cdots \wedge c_n \vdash \{c \wedge c_1' \wedge \cdots \wedge c_n' \mid c_i' \in C_i, \text{ pour tout } i = 1, \ldots, n\}$$

   La propriété 2 de la relation de couverture assure que

   $$\exists_{-a}\, c \wedge c_1 \wedge \cdots \wedge c_n \vdash \{\exists_{-a}\, c \wedge c_1' \wedge \cdots \wedge c_n' \mid c_i' \in C_i, \text{ pour tout } i = 1, \ldots, n\}$$

   Donc, il suffit de prendre $C = \{\exists_{-a}\, c \wedge c_1' \wedge \cdots \wedge c_n' \mid c_i' \in C_i, \text{ pour tout } i = 1, \ldots, n\}$.

2. Soit $\frac{\{a[c'] \mid c' \in C'\}}{a[c]}$ une règle de couverture.

   Supposons que, pour toute $c' \in C'$, $a[c'] \in SS_{\vdash}(P)$ et il existe un ensemble de contraintes $C'_{c'}$ tel que $c' \vdash C'_{c'}$ et $a[c''] \in SS(P)$, pour toute contrainte $c'' \in C'_{c'}$.

   Montrons qu'alors il existe un ensemble de contraintes C tel que $c \vdash C$ et $a[c''] \in SS(P)$, pour toute $c'' \in C$.

   La règle de couverture est issue de $c \vdash C'$.

   La propriété 3 de la règle de couverture assure que

$$c \vdash \bigcup_{c' \in C'} C'_{c'}$$

   Donc, il suffit de prendre $C = \bigcup_{c' \in C'} C'_{c'}$.

■

Preuve du corollaire 4.1 Il suffit d'appliquer les définitions. ■

5 Conclusion

Nous avons ainsi un cadre formel uniforme dans lequel sont reliées les notions de réponse calculée et de réponse au sens déclaratif, par l'intermédiaire d'une relation abstraite de couverture.

Ce cadre peut s'appliquer aussi bien au cas où la sémantique des programmes est définie en référence à un domaine $\mathcal{D}$, une théorie $\mathcal{T}$, et peut même rendre compte d'un algorithme incomplet de test de satisfaction de contraintes.

Ce cadre permet d'étudier (travail en cours) les problèmes de diagnostic déclaratif d'erreurs, en particulier, dans le cas de réponses manquantes (insuffisance).