La programmation logique avec contraintes revisitée en termes d'arbres de preuve et de squelettes

Michel Bergère, Gérard Ferrand, François Le Berre, Bernard Malfon, Alexandre Tessier
Laboratoire d'Informatique Fondamentale d'Orléans (LIFO)
Université d'Orléans

Résumé 24 juin 1994

L'objectif de ce travail est de redéfinir les bases de la programmation logique avec contraintes en tirant tout le bénéfice des notions d'*arbre de preuve* et de *squelette* d'arbre de preuve développées dans le cadre de la programmation logique par Deransart et Maluszynski [3].

Ceci présente le remarquable avantage de donner une *définition intrinsèque* aux réponses qu'un programme peut donner à une question qui lui est posée: ce sont les squelettes finis solubles ayant pour racine le but question. La solubilité, qui va de soi en programmation logique, dépend ici d'un paramètre qui apparaît sous la forme d'un domaine ou d'une théorie. Ainsi l'indépendance de nombreux résultats par rapport à la règle de sélection d'atome est clairement expliquée. De nouvelles questions peuvent aussi être formulées, par exemple la caractérisation des branches non échecs dans les arbres de recherche équitables.

Une autre idée importante est l'utilisation d'une *théorie* qui axiomatise le fonctionnement d'un résolveur de contraintes.

Il apparaît que la notion essentielle n'est pas le domaine sur lequel le programme est censé travailler : les entiers, les réels, les arbres finis ou infinis sur tel ou tel alphabet... C'est plutôt *l'incidence de ce domaine sur la satisfiabilité des contraintes*, ce qui est nettement mieux exprimé par une théorie $\mathcal{T}$: une contrainte est réputée satisfiable si sa clôture existentielle est conséquence logique de $\mathcal{T}$. Le domaine n'est pas manipulé en tant que tel mais à travers ses propriétés exprimées par la théorie.

La définition de la notion de réponse comme squelette fini complet ayant pour racine le but question $\leftarrow b$ permet d'élargir le champ d'investigation : tout squelette fini complet est une bonne réponse, en ce sens que, si r est la contrainte associée, alors

$$P \models r \rightarrow b.$$

C'est ce que nous appelons une *réponse généralisée*. Mais parmi les implications $r \rightarrow b$ conséquences logiques du programme, certaines sont trivialement vraies parce que la contrainte r est toujours fausse (insatisfiable). D'un point de vue théorique, on peut *éliminer* les réponses triviales.

D'un point de vue opérationnel, il est essentiel d'éliminer les explorations inutiles qui ne peuvent conduire qu'à des réponses triviales.

Tous les interprètes ont un résolveur de contraintes *qui ne sert qu'à éliminer* les recherches dont on sait pertinemment qu'elles sont infructueuses. Souvent, parce que c'est impossible ou pour des raisons d'efficacité, le résolveur n'est pas complet : il ne répond pas toujours OUI ou NON à la question de savoir si telle contrainte (ou tel ensemble de contraintes) est satisfiable ; il répond parfois PEUT-ETRE. C'est par exemple le cas du résolveur de CLP(R) qui gèle les contraintes non linéaires dans l'attente de contraintes supplémentaires qui débloqueraient la situation ; s'il reste des contraintes en attente à la fin du calcul, l'interprète donne une réponse assortie du commentaire PEUT-ETRE. Un tel résolveur est représenté par une *théorie qui n'est pas satisfaction-complète.*

La théorie ayant pour but d'éliminer, elle ne rejette que ce qui est certainement faux. Nous sommes donc amenés à donner une nouvelle notion de réponse: *une réponse faible* est un squelette fini dont la contrainte n'est pas réfutée par $\mathcal{T}$ (sa négation n'est pas conséquence logique de $\mathcal{T}$). Il est intéressant de voir que la correspondance entre les buts d'échec fini et les conséquences négatives du complété du programme s'exprime naturellement en termes de conséquences faibles d'une théorie non satisfaction-complète.

1 Principales définitions

Le *langage du programme* est défini à partir de quatre ensembles: V (variables), Σ (fonctions), Π_c (prédicats de contraintes) et Π_p (prédicats définis par le programme).
Les *contraintes* constituent une partie du langage du premier ordre $\mathcal{L}(V, \Sigma, \Pi_c)$.
Un *programme contraint* est un ensemble de clauses de la forme $A \leftarrow c \,\square\, B_1, \cdots, B_n$ où A et les B_i sont des atomes (formules atomiques de $\mathcal{L}(V, \Sigma, \Pi_p)$), c est une contrainte.
Une *pré-interprétation* $\mathcal{D}$ est une interprétation de $\mathcal{L}(V, \Sigma, \Pi_c)$. On appelle $\mathcal{D}$*-atome* un objet abstrait $p(d_1, \cdots, d_n)$ où p est un symbole de prédicat de programme et les d_i des éléments du domaine D de la pré-interprétation $\mathcal{D}$.
Toute valuation dans $\mathcal{D}$ s'étend naturellement aux termes (en donnant des éléments de D), aux contraintes (en donnant des valeurs booléennes), aux atomes (en donnant des $\mathcal{D}$-atomes).
Une interprétation dans $\mathcal{D}$ (une $\mathcal{D}$*-interprétation*) est assimilable à un ensemble de $\mathcal{D}$-atomes.
On appelle $\mathcal{D}$-modèle du programme P une interprétation dans $\mathcal{D}$ qui est modèle des clauses de P.
Les $\mathcal{D}$*-règles* associées au programme P, du type $a \leftarrow b_1, \cdots, b_n$ où a et les b_i sont des $\mathcal{D}$-atomes, sont données par les valuations des clauses de P pour lesquelles la contrainte prend la valeur *vrai*. On en déduit un *opérateur de conséquence immédiate* $T_{\mathcal{D}}^P$ opérant sur les $\mathcal{D}$-interprétations.
Un $\mathcal{D}$*-arbre de preuve* est étiqueté par des $\mathcal{D}$-atomes de telle sorte que si un nœud (d'étiquette a) a n fils (d'étiquettes $b_1, \cdots, b_n$) alors $a \leftarrow b_1, \cdots, b_n$ est une $\mathcal{D}$-règle. Une forêt de preuve est un n-uplet d'arbres de preuve.
Les *squelettes* sont étiquetés par des clauses de P, à l'exception éventuelle de

- la racine qui peut être étiquetée par un but (une clause sans tête)
- certaines feuilles qui peuvent être indéfinies (étiquetées par ?).

Le squelette est complet s'il n' a pas de feuille indéfinie.
A tout squelette S on associe une collection de contraintes $cont(S)$ (assimilable à une contrainte unique si S est fini) qui est satisfiable ssi S correspond à au moins une forêt de preuve. C'est cela qu'on appelle *squelette soluble* selon $\mathcal{D}$.
Une *réponse* pour le but $B = \leftarrow b$ est un squelette fini complet soluble de racine B.
Les dérivations se définissent comme des suites de squelettes solubles de racine B.
La *contrainte réponse* associée à un squelette réponse est la clôture existentielle sur les variables autres que celles de la question de la conjonction des contraintes de $cont(S)$.
Une *théorie* $\mathcal{T}$ est un ensemble de formules closes de $\mathcal{L}(V, \Sigma, \Pi_c)$ fermé par conséquence logique. $\mathcal{T}$ est satisfaction-complète si pour toute contrainte c, soit $\mathcal{T} \models \tilde{\exists} c$, soit $\mathcal{T} \models \neg c$ ($\tilde{\exists} c$ est la clôture existentielle de c). c est (fortement) satisfiable selon $\mathcal{T}$ si $\mathcal{T} \models \tilde{\exists} c$.
Une *réponse forte* selon $\mathcal{T}$ est un squelette fini complet fortement soluble : sa contrainte est fortement satisfiable.
Lorsque $\mathcal{T}$ n'est pas satisfaction-complète, c est faiblement satisfiable si $non(\mathcal{T} \models \neg c)$, c'est-à-dire si c n'est pas réfutée par $\mathcal{T}$. Cela correspond à la notion de *réponse faible*: squelette fini complet de racine B dont la contrainte est faiblement satisfiable (squelette faiblement soluble). La notion de squelette faiblement soluble s'étend aux squelettes infinis.
B est *d'échec fini* si toutes les branches d'un arbre de recherche (ou de tous les arbres de recherche équitables) sont des échecs: tous les prolongements conduisent à des contraintes réfutées par $\mathcal{T}$.

Une *réponse générale* est la contrainte associée à tout squelette fini complet de racine B.
Le *complété* du programme P, noté P^*, est obtenu de la même manière qu'en programmation logique classique.

2 Principaux résultats

Les $\mathcal{D}$-modèles de P sont les pré-points fixes de $T_{\mathcal{D}}^P$. Le plus petit $\mathcal{D}$-modèle de P est le plus petit point fixe de $T_{\mathcal{D}}^P$, identique à l'ensemble des racines des $\mathcal{D}$-arbres de preuve.
Les $\mathcal{D}$-modèles de P^* sont les points fixes de $T_{\mathcal{D}}^P$; P et P^* ont le même plus petit $\mathcal{D}$-modèle.

Dans une théorie $\mathcal{T}$,

- si $\leftarrow b$ a une réponse forte, alors $P, \mathcal{T} \models \tilde{\exists} b$
- si $\leftarrow b$ a une réponse faible, alors $non(P, \mathcal{T} \models \neg b)$
- si $P, \mathcal{T} \models \tilde{\exists} b$ alors $\leftarrow b$ a une réponse faible
- les branches succès des arbres de recherche pour le but B correspondent aux réponses faibles
- les branches non échec des arbres de recherche équitables pour B correspondent aux squelettes (éventuellement infinis) faiblement solubles de racine B.
- les cinq propriétés suivantes sont équivalentes :
  - $P, \mathcal{T} \models c \rightarrow b$
  - $\mathcal{T} \models c \rightarrow \bigvee_{r \in RG} r$ (réponses générales)
  - il existe une partie finie RG' de RG telle que $\mathcal{T} \models c \rightarrow \bigvee_{r' \in RG'} r'$
  - $\mathcal{T} \models c \rightarrow \bigvee_{r \in RF} r$ (réponses faibles)
  - il existe une partie finie RF' de RF telle que $\mathcal{T} \models c \rightarrow \bigvee_{r' \in RF'} r'$, et RF' peut se réduire à une seule contrainte réponse si la propriété d'indépendance des contraintes négatives est vérifiée ; c'est le cas pour les contraintes correspondant aux substitutions réponses en programmation logique classique.
- les conséquences positives de P et de P^* coïncident
- $\leftarrow b$ est d'échec fini ssi $P^*, \mathcal{T} \models \neg b$

Indépendamment de toute théorie, les trois propriétés sont équivalentes :

- $P \models c \rightarrow b$
- $\models c \rightarrow \bigvee_{r \in RG} r$ (réponses générales)
- il existe une partie finie RG' de RG telle que $\models c \rightarrow \bigvee_{r' \in RG'} r'$, et RG' peut se réduire à une seule contrainte si la propriété d'indépendance des contraintes négatives est vérifiée.

Bibliographie

[1] K. Apt, Logic Programming, Handbook of Theoretical Computer Science (J. Van Leeuwen ed.), Elsevier Science Publishers B. V., 1990.

[2] J. Cohen, Constraint Logic Programming Languages, CACM n° 33, 52-68, juillet 1990.

[3] P. Deransart & J. Maluszynski, A Grammatical View of Logic Programming, MIT Press 1993.

[4] M. Gabrielli & G. Levy, Modeling Answer Constraints in Constraint Logic Programming, Proc. 8th ICLP, 238-252, 1991.

[5] N. Heintze, N. Jaffar, S. Michaylov, P. Stuckey & R. Yap, The CLP(R) programmer's manual version 1.2, IBM T. J. Watson Research Center.

[6] J. Jaffar & J-L. Lassez, Constraint Logic Programming, Technical Report 86/73, Department of Computer Science, Monash University, 1986 (et 14th ACM Symposium on Principles of Programming Languages, Munich, janvier 1987).

[7] J. Jaffar & M. Maher, Constraint Logic Programming: a survey, Journal of Logic Programming 1994.

[8] J. W. Lloyd, Foundations of Logic Programming, Springer-Verlag, 1987.

[9] M. Maher, A Logic Programming View of CLP, IBM Research Report, T. J. Watson Research Center, 1993 (et 10th ICLP 1993).