Analyse de flot de données de programmes récursifs à l'aide de grammaires hors-contexte

Albert Cohen

Analyse de flot de donnes de programmes rcursifs l'aide de grammaires hors-contexte

Albert Cohen[a]

[a]PRiSM, Universit de Versailles, 45 Avenue des tats-Unis, 78035 Versailles, France.

Rsum
Cet article prsente une application originale des langages algbriques l'analyse de flot de donnes de programmes rcursifs. La technique prsente autorise une description globale et prcise du flot, en montrant d'importantes proprits smantiques.

Mots cls Analyse de flot, paralllisation, compilation, rcursivit, langages algbriques.

1. Introduction

Les techniques de paralllisation automatique pour programmes impratifs reposent presque toujours sur des analyses statiques d'accs la mmoire. En effet, l'excution dans un ordre quelconque de deux accs la mmoire ncessite que ceux-ci ne soient pas en conflit.

Ainsi les *analyses d'alias* recherchent des couples d'accs la mme case mmoire, et les *analyses de dpendance* expriment les accs conflictuels comme relations entre instances d'instructions [1]. L'*analyse de flot de donnes* raffine ces relations en dterminant, pour une lecture d'une cellule mmoire — le *puits* — la *dernire* criture dans cette mme cellule — la *source*.

Sur les nids de boucles, on distingue les programmes *contrle statique* dont l'excution fixe ds la compilation est propice une analyse *exacte* [2], des programmes *contrle dynamique* dont l'ensemble des oprations est imprdictible statiquement, on parle d'analyse *floue* [3].

La distinction s'tend la rcursion et dans un travail prcdent [4] des restrictions garantissent le contrle statique. On tudie ici l'analyse *floue* de programmes rcursifs contrle dynamique. Les analyses abstraites classiques calculent un flot de donnes trop imprcis pour valider les transformations de la paralllisation automatique. L'approche de ce papier est diffrente : on prsente une modlisation par langages algbriques puis une discussion des rsultats.

2. Exemples

On utilisera une syntaxe Pascal mais la mthode prsente ici est indpendante du langage. Considrons la procdure `F` fig. 1, initie par l'appel `F (0, 0)`. l'aide d'un tiquetage l,s,r, on identifie l'instance d'une instruction au *mot de contrle* form de la concatnation des tiquettes des instructions menant cette instance dans l'*arbre d'appel* : e.g. l'instance $\bigcirc$ de s fig. 2.a est excute aprs l'appel r, l, r et l ; on la dnote par $rlrls$. $\lhd$ dnote l'ordre textuel des instructions : $l \lhd s \lhd r$ (infixe). L'ordre d'excution est alors l'ordre *lexicographique* $\ll$ des mots de contrle. On ne fait *aucune hypothse sur les instructions conditionnelles* (P_l et P_r).

```
      procedure F (l, r) begin
l       if (P_l) then F (l+1, r);
s       A[l+r] := ··· A[l+r-1] ···;
r       if (P_r) then F (l, r+1)
      end
```

```
      procedure G (l, r) begin
l       if (P_l) then G (l+1, r);
s       A[l-r] := ··· A[l-r]+1 ···;
r       if (P_r) then G (l, r+1)
      end
```

FIG. 1 – *Procdures `F` et `G`.*

Puisque l'argument `l` (resp. `r`) est incrment chaque appel suivant l (resp. r), l'instance $rlrls$ lit la valeur `A[3]`. Les sources de $rlrls$ possibles sont donc toutes les critures *prcdentes* dans `A[3]`, non oblitres entre

temps par une autre criture dans `A[3]`. Ne pouvant garantir l'excution de ces instances la compilation, toutes les critures dans `A[3]` sont sources potentielles ; dsignes par ■ fig. 2.a (les instances de mme profondeur accdent au mme lment).

rllrs lit aussi `A[3]`, et les instances prcdentes assignant `A[3]` sont les mmes que pour *rlrls*. Mais dsormais, *rlls* (prcdant le puits *rllrs*) *est excute*, en tant qu'*anctre* [1] de *rllrs* :

Lemme 1 *Soit s une instruction d'une procdure `F` n'appartenant pas une conditionnelle. Chaque appel `F` excutant s, l'excution d'une instance implique celle de tous ses anctres* [1].

Les critures prcdant *rlls* sont donc «crases» (notes □, fig. 2.b). La source de *rllrs* est *rlls*.

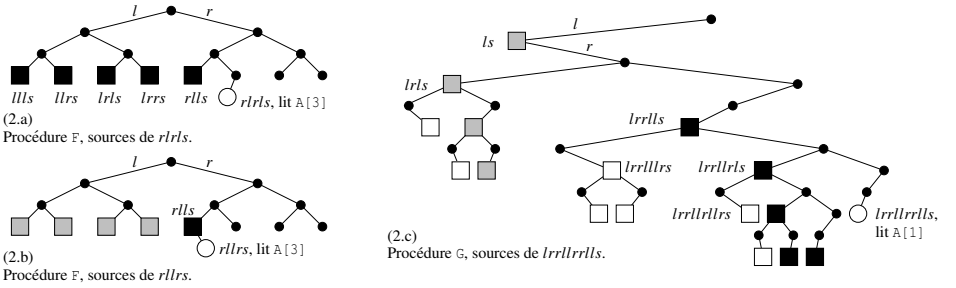

FIG. 2 – *Arbres d'appel partiels de `F` et `G`.*

Considrons prsent la procdure `G` fig. 1, initie par `G (0, 0)`. L'index de `A` tant `r-l`, les sources de *lrrllrrlls* se situent parmi les critures dans `A[1]`. Il apparat que *lrrlls*, anctre de *lrrllrrlls*, crit dans `A[1]` (fig. 2.c). Donc toutes les critures prcdant *lrrlls* sont crases (*ls*, *lrls*, etc.). De plus, certaines critures potentielles dans `A[1]` — reprsentes par □ — s'excutent *avant* un de leurs anctres crivant dans `A[1]` : ainsi lorsque *lrrllrllrs* est excute, *lrrllrls* aussi ; *lrrllrllrs* n'est donc pas source potentielle. Les sources possibles sont donc les ■.

L'objectif est de dcrire l'ensemble des ■ l'aide d'une expression *paramtre* par le mot de contrle du puits, *indpendante* de la taille des donnes ou des calculs.

3. Caractrisation des sources l'aide de langages algbriques

Lorsque l'instruction d'criture appartient une conditionnelle on ne peut en garantir l'excution, et toute instance conflictuelle prcdant le puits sera source potentielle. Sinon, on raffine l'analyse en distinguant trois types d'instances : ■ dsigne les sources potentielles ; □ les critures en conflit dont un anctre postrieur est source potentielle ; ▣ celles qui sont crases par un anctre du *puits* mais n'ont pas d'anctre postrieur source potentielle.

3.1. Modle

On considre un programme compos d'une unique procdure rcursive, les variables inductives indexant des tableaux sont translates lors des appels (pour `F` : l et r sont incrmentes), les tableaux sont unidimensionnels et les indexations affines.

Les exemples prcdents indiquent que les sources sont caractrises par des quations affines et des critres lis l'ordre lexicographique. D'o l'ide d'un automate dot d'un *compteur* (simul par une pile [5, 6]) capable de reprer les conflits et de garantir l'ordre lexicographique.

Notations : sur la procdure `G` on note $\Sigma = \{l,r\}$, w le mot de contrle du puits et w' celui d'une source potentielle ; $w,w' \in \Sigma^* \cdot s$. L'index de `A` s'exprime en fonction de w comme «le nombre de l : $|l|_w$, moins le nombre de r : $|r|_w$» . La fonction d'accs est donc $\varphi(w) = |l|_w - |r|_w$. On crit que w et w' accdent au mme lment : $\varphi(w) = \varphi(w')$ [2], c'est l'*quation de conflit.*

1. La notion d'anctre diffre de celle de prfixe : si u est *prfixe* d'un mot de contrle w, us est *anctre* de w.

3.2. Automate

Un compteur requiert trois symboles de pile : le symbole Z de fond de pile dtecte l'annulation, I code en unaire les nombres positifs, D les nombres ngatifs. Mais on ne peut garantir l'ordre d'excution avec un automate reconnaissant ww', on doit dcomposer l'ordre lexicographique :

$$w' \ll w \quad \Longleftrightarrow \quad w = u\alpha v, \quad w' = u\alpha' v', \quad \alpha' \lhd \alpha, \quad u \in \Sigma^*, \quad \alpha v, \alpha' v' \in \Sigma^* s. \tag{1}$$

Ainsi on cherche caractriser des mots de la forme $u \bullet \alpha v \alpha' v'$ o $\bullet$ est un symbole de sparation, tels que $u\alpha' v'$ soit source potentielle de $u\alpha v$. Pour w et w' donns, il existe un unique mot $u \bullet \alpha v \alpha' v'$ vrifiant $\alpha' \lhd \alpha$. Rciproquement, la dtermination de w et w' est rendue possible par $\bullet$ et le s qui termine αv. La transformation est donc bijective, et on notera $\langle w \bullet w' \rangle$ le mot $u \bullet \alpha v \alpha' v'$. L'quation de conflit devient [2] $\varphi(\alpha) + \varphi(v) = \varphi(\alpha') + \varphi(v')$.

Le fonctionnement de l'automate propos la figure 3 s'nonce relativement simplement :

1. la pile contenant initialement le symbole Z, lire le prfixe u ($\mathbf{q_0}$) et le symbole $\bullet$ [3] ($\mathbf{q_1}$) ;
2. lire α et v en «mmorisant» $\varphi(\alpha) + \varphi(v)$ sur la pile ($\mathbf{q_2}$), jusqu' ce que s soit lu ($\mathbf{q_3}$ et $\mathbf{q_4}$) ;
3. lire α' avec $\alpha' \lhd \alpha$ ($\mathbf{q_3}$ et $\mathbf{q_4}$) et v' ($\mathbf{q_5}$) en mmorisant $\varphi(\alpha) + \varphi(v) - \varphi(\alpha') + \varphi(v')$; en s'assurant de surcrot que l'on *ne lit pas* l lorsque la fonction de conflit est nulle, jusqu' ce que s soit lu dans une configuration o la fonction de conflit est nulle ($\mathbf{q_6}$).

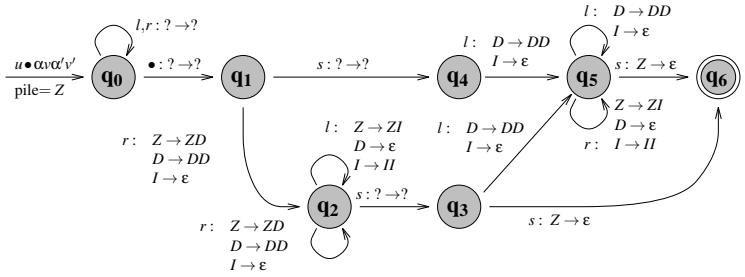

FIG. 3 – *Automate pile dcrivant le flot de la procdure* G.

L'automate reconnat toutes les sources potentielles. On montre notamment que les critures de type □ (dont un anctre *postrieur* est source potentielle) sont limines, car une transition lisant l n'est pas valide lorsque la pile est Z ($\mathbf{q_3} \to \mathbf{q_5}$, $\mathbf{q_4} \to \mathbf{q_5}$ et $\mathbf{q_5} \to \mathbf{q_5}$). Pourtant, les critures de type ■ prcdant un anctre de w ne sont pas limines. Les langages algbriques ne permettent pas de dcrire ces instances ; cela fera l'objet de travaux ultrieurs.

3.3. Grammaire hors-contexte

Donner l'expression complte de la grammaire prsente peu d'intrt, son calcul est simple [5]. On note tout de mme que les sources du sous-arbre d'une source potentielle w' (les ■ issus de w') constituent le langage de Lukasiewicz [6] : $\mathrm{S} \vdash w' \cdot s \,\big|\, w' \cdot r\text{Ł}s \quad \text{Ł} \vdash l \,\big|\, r\text{ŁŁ}$.

4. Gnralisation

On montre qu'il est possible d'tendre la *construction* de l'automate de la procdure G aux programmes respectant les hypothses nonces prcdemment, et l'algorithme reste simple : le nombre d'tats de l'automate crot linairement avec le nombre d'instructions. Formellement :

Thorme 1 *Pour* deux *instructions de lecture et d'criture donnes, on* construit *un automate pile (ou une grammaire) acceptant* $\langle w \bullet w' \rangle$ *lorsque l'une des conditions suivantes est vrifie :*
(i) w' *est une source potentielle de* w *;*
(ii) w' *crit dans la mme cellule que* w*, et une source potentielle anctre de* w*, mais pas de* w'*,*
est effectivement excute.

---

2. Les accs en criture et en lecture sont les mmes sur cet exemple, mais l'analyse ne fait pas cette hypothse.
3. Si les accs en lecture et en criture n'taient pas les mmes, il faudrait mettre jour la pile en consquence.

Remarque : lorsque l'instruction source considre est englobe par une instruction conditionnelle il est impossible de garantir qu'un anctre du puits est effectivement excut, la condition (*ii*) n'a donc plus lieu d'tre. Dans ce cas, la caractrisation l'aide de langages algbriques assure une *prcision optimale* (sans hypothse sur les prdicats des instructions conditionnelles).

Programmes rels : les restrictions (sec. 3) sont trop svres. Les mots de contrle ainsi que l'quivalence des ordres lexicographique et d'excution se conservent avec des procdures mutuellement rcursives. En contraignant les variables d'induction, la mthode s'tend ainsi de plus gros programmes, comprenant notamment les *nids de boucles*. Enfin les tableaux multi-dimensionnels posent problme car la classe des langages algbriques *n'est pas* close pour l'intersection ; une solution peu satisfaisante consiste «linariser» les tableaux.

5. valuation et dveloppements

De nombreuses proprits smantiques se traduisent en problmes dcidables sur des grammaires hors-contexte. On peut dcider en temps linaire si w' est une source potentielle de w, vrifier la validit des accs, rechercher les lectures de non-initialises, montrer que l'ensemble des sources potentielles est fini, etc. Ces proprits peuvent tre montres pour un puits donn ou pour toutes les instances d'une instruction d'assignation. L'intersection d'un langage algbrique avec un langage rationnel tant algbrique, on montre mme que la dcidabilit d'une proprit pour un puits implique la dcidabilit pour tout ensemble rationnel de puits.

Il semble ainsi que cette technique gagne en prcision sur les analyses classiques, bien qu'aucune exprimentation ne permette de l'affirmer prsent (l'implmentation actuelle se limite l'analyse exacte [4]) ; bien sr, des restrictions sur les programmes analyss sont invitables.

De tels rsultats s'avrent trs utiles pour l'optimisation et la vrification de code, et feront l'objet de travaux futurs. En revanche, l'expansion de tableaux et la dduction automatique d'un ordonnancement parallle partir de cette mthode restent ouvertes.

6. Conclusion

Cet article prsente une analyse de flot de donnes des accs aux tableaux dans les programmes rcursifs. Le contrle tant dynamique, on ne peut calculer que des ensembles de sources potentielles. Afin de les dcrire on introduit un modle fond sur les langages algbriques. La prcision obtenue est leve et les proprits du flot sont dcouvertes automatiquement.

Les applications potentielles sont nombreuses, notamment en paralllisation automatique, sachant que l'adaptation des techniques de vrification et de transformation de programme ce modle de description du flot requiert sans doute le dveloppement de nouveaux algorithmes.

Bibliographie

1. Pugh W. – A practical algorithm for exact array dependence analysis. *Communications of the ACM*, vol. 35, n° 8, août 1992, pp. 27–47.
2. Feautrier P. – Dataflow analysis of scalar and array references. *Int. Journal of Parallel Programming*, vol. 20, n° 1, février 1991, pp. 23–53.
3. Collard J.-F., Barthou D. et Feautrier P. – Fuzzy array dataflow analysis. *In : Proc. of 5th ACM SIGPLAN, Symp. PPoPP*, pp. 92–102. – Santa Barbara, CA, juillet 1995.
4. Cohen A., Collard J.-F. et Griebl M. – *Array Data-flow Analysis for Imperative Recursive Programs*. – Rapport technique n° 96/035, PRiSM, Universit de Versailles, septembre 1996.
5. Hopcroft J. E. et Ullman J. D. – *Introduction to Automata Theory, Languages, and Computation*. – Addison-Wesley, 1979.
6. Berstel J. – *Transductions and Context-Free Languages*. – Teubner, 1979.