



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

Albert Cohen

► To cite this version:

Albert Cohen. Analyse de flot de données de programmes récursifs à l'aide de grammaires hors-contexte. Rencontres francophones du parallélisme (RenPar9), May 1997, Lausanne, Suisse. hal-01257321

HAL Id: hal-01257321

<https://hal.science/hal-01257321>

Submitted on 20 Jan 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Albert Cohen^a

^aPRiSM, Université de Versailles, 45 Avenue des tats-Unis, 78035 Versailles, France.

Résumé

Cet article présente une application originale des langages algébriques l'analyse de flot de données de programmes récursifs. La technique présentée autorise une description globale et précise du flot, en montrant d'importantes propriétés sémantiques.

Mots clés Analyse de flot, parallélisation, compilation, récursivité, langages algébriques.

1. Introduction

Les techniques de parallélisation automatique pour programmes impératifs reposent presque toujours sur des analyses statiques d'accès à la mémoire. En effet, l'exécution dans un ordre quelconque de deux accès à la mémoire nécessite que ceux-ci ne soient pas en conflit.

Ainsi les *analyses d'alias* recherchent des couples d'accès à la même case mémoire, et les *analyses de dépendance* expriment les accès conflictuels comme relations entre instances d'instructions [1]. L'*analyse de flot de données* raffine ces relations en déterminant, pour une lecture d'une cellule mémoire — le *puits* — la *dernière* écriture dans cette même cellule — la *source*.

Sur les nœuds de boucles, on distingue les programmes *contrôle statique* dont l'exécution fixe dès la compilation est propice à une analyse *exacte* [2], des programmes *contrôle dynamique* dont l'ensemble des opérations est imprédictible statiquement, on parle d'analyse *floue* [3].

La distinction s'étend à la récursion et dans un travail précédent [4] des restrictions garantissent le contrôle statique. On étudie ici l'analyse *floue* de programmes récursifs à contrôle dynamique. Les analyses abstraites classiques calculent un flot de données trop imprécis pour valider les transformations de la parallélisation automatique. L'approche de ce papier est différente : on présente une modélisation par langages algébriques puis une discussion des résultats.

2. Exemples

On utilisera une syntaxe Pascal mais la méthode présentée ici est indépendante du langage. Considérons la procédure F fig. 1, initiée par l'appel $F(0, 0)$. L'aide d'un étiquetage l, s, r , on identifie l'instance d'une instruction au *mot de contrôle* formé de la concaténation des étiquettes des instructions menant à cette instance dans l'*arbre d'appel* : e.g. l'instance $\circ$ de s fig. 2.a est exécutée après l'appel r, l, r et l ; on la note par $rlrls$. $\triangleleft$ note l'ordre textuel des instructions : $l \triangleleft s \triangleleft r$ (infixe). L'ordre d'exécution est alors l'ordre *lexicographique* $\ll$ des mots de contrôle. On ne fait *aucune hypothèse sur les instructions conditionnelles* (P_l et P_r).

<pre> procedure $F(l, r)$ begin l if (P_l) then $F(l+1, r)$; s $A[l+r] := \dots A[l+r-1] \dots$; r if (P_r) then $F(l, r+1)$ end </pre>	<pre> procedure $G(l, r)$ begin l if (P_l) then $G(l+1, r)$; s $A[l-r] := \dots A[l-r]+1 \dots$; r if (P_r) then $G(l, r+1)$ end </pre>
---	---

FIG. 1 – Procédures F et G .

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

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

$rlrs$ lit aussi $A[3]$, et les instances prcdentes assignant $A[3]$ sont les mmes que pour $rlrls$. Mais dsormais, rls (prcdant le puits $rlrs$) est excute, en tant qu'ancetre¹ de $rlrs$:

Lemme 1 Soit s une instruction d'une procudre F n'appartenant pas une conditionnelle. Chaque appel F excutant s , l'excuton d'une instance implique celle de tous ses ancres¹.

Les critiques prcdant rls sont donc «crases» (notes □, fig. 2.b). La source de $rlrs$ est rls .

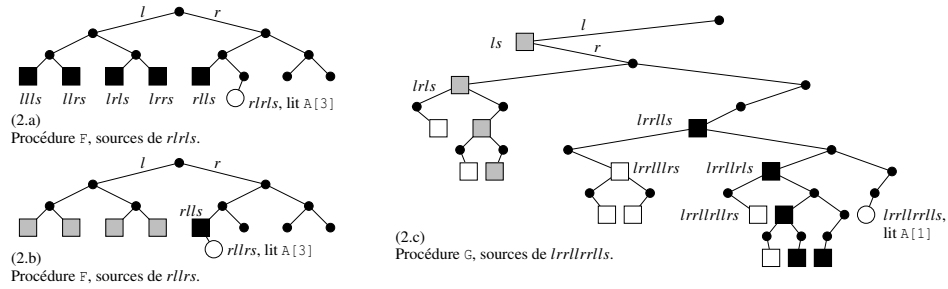


FIG. 2 – Arbres d'appel partiels de F et G .

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

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

3. Caractrisation des sources l'aide de langages algébriques

Lorsque l'instruction d'criture appartient une conditionnelle on ne peut en garantir l'exécution, 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 critiques en conflit dont un ancetre postérieur est source potentielle ; ■ celles qui sont crases par un ancetre du *puits* mais n'ont pas d'ancetre postérieur source potentielle.

3.1. Modle

On considre un programme compos d'une unique procudre rcursive, les variables inductives indexant des tableaux sont traduites 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 caractrisées 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 procudre 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 $\phi(w) = |l|_w - |r|_w$. On crit que w et w' accdent au mme lment : $\phi(w) = \phi(w')$ ², c'est l'*quation de conflit*.

¹. La notion d'ancetre diffre de celle de *prfixe* : si u est *prfixe* d'un mot de contrle w , u est *ancetre* 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'excuton avec un automate reconnaissant ww' , on doit dcomposer l'ordre lexicographique :

$$w' \ll w \iff w = u\alpha v, \quad w' = u\alpha'v', \quad \alpha' \triangleleft \alpha, \quad u \in \Sigma^*, \quad \alpha v, \alpha'v' \in \Sigma^*s. \quad (1)$$

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' donnns, il existe un unique mot $u \bullet \alpha v \alpha' v'$ vrifiant $\alpha' \triangleleft \alpha$. 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 (q_0) et le symbole $\bullet$ ³ (q_1) ;
2. lire α et v en «mmorisant» $\varphi(\alpha) + \varphi(v)$ sur la pile (q_2), jusqu' ce que s soit lu (q_3 et q_4) ;
3. lire α' avec $\alpha' \triangleleft \alpha$ (q_3 et q_4) et v' (q_5) en mmorisant $\varphi(\alpha) + \varphi(v) - \varphi(\alpha') + \varphi(v')$; en s'assurant de surcroit 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 (q_6).

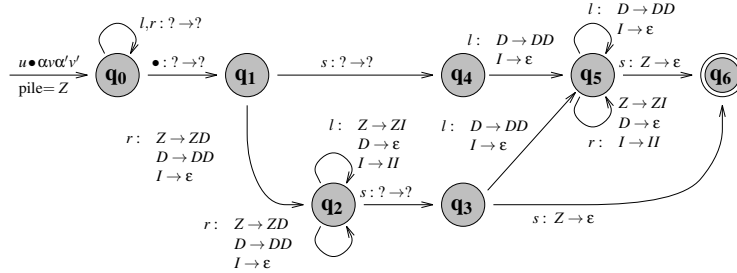


FIG. 3 – Automate pile dcrivant le flot de la procdure G .

L'automate reconnat toutes les sources potentielles. On montre notamment que les critures de type $\square$ (dont un ancre *postrieur* est source potentielle) sont limines, car une transition lisant l n'est pas valide lorsque la pile est Z ($q_3 \rightarrow q_5$, $q_4 \rightarrow q_5$ et $q_5 \rightarrow q_5$). Pourtant, les critures de type $\blacksquare$ prcdant un ancre 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 $\blacksquare$ issus de w') constituent le langage de Lukasiewicz [6] : $S \vdash w' \cdot s \mid w' \cdot r \mathbb{L} s \quad \mathbb{L} \vdash l \mid r \mathbb{L} \mathbb{L}$.

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 linaiement 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 ancre 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 consequence.

Remarque : lorsque l’instruction source considérée est englobée par une instruction conditionnelle il est impossible de garantir qu’un ancre du puits est effectivement exécuté, la condition (ii) n’a donc plus lieu d’être. Dans ce cas, la caractérisation à l’aide de langages algébriques assure une *précision optimale* (sans hypothèse sur les prédicats des instructions conditionnelles).

Programmes réels : les restrictions (sec. 3) sont trop strictes. Les mots de contrôle ainsi que l’équivalence des ordres lexicographique et d’exécution se conservent avec des procédures mutuellement récursives. En contraignant les variables d’induction, la méthode s’étend ainsi à des programmes plus gros, comprenant notamment les *nids de boucles*. Enfin les tableaux multi-dimensionnels posent problème car la classe des langages algébriques n’est pas close pour l’intersection ; une solution peu satisfaisante consiste «linéariser» les tableaux.

5. valuation et développements

De nombreuses propriétés sémantiques se traduisent en problèmes décidables sur des grammaires hors-contexte. On peut décider en temps linéaire si w' est une source potentielle de w , vérifier la validité des accès, rechercher les lectures de non-initialisés, montrer que l’ensemble des sources potentielles est fini, etc. Ces propriétés peuvent être montrées pour un puits donné ou pour toutes les instances d’une instruction d’assignation. L’intersection d’un langage algébrique avec un langage rationnel tant algébrique, on montre même que la décidabilité d’une propriété pour un puits implique la décidabilité pour tout ensemble rationnel de puits.

Il semble ainsi que cette technique gagne en précision sur les analyses classiques, bien qu’aucune expérimentation ne permette de l’affirmer présent (l’implémentation actuelle se limite à l’analyse exacte [4]) ; bien sûr, des restrictions sur les programmes analysés sont inévitables.

De tels résultats s’avèrent très utiles pour l’optimisation et la vérification de code, et feront l’objet de travaux futurs. En revanche, l’expansion de tableaux et la déduction automatique d’un ordonnancement parallèle à partir de cette méthode restent ouvertes.

6. Conclusion

Cet article présente une analyse de flot de données des accès aux tableaux dans les programmes récursifs. Le contrôle tant dynamique, on ne peut calculer que des ensembles de sources potentielles. Afin de les décrire on introduit un modèle fondé sur les langages algébriques. La précision obtenue est élevée et les propriétés du flot sont découvertes automatiquement.

Les applications potentielles sont nombreuses, notamment en parallélisation automatique, sachant que l’adaptation des techniques de vérification et de transformation de programme à ce modèle de description du flot requiert sans doute le développement 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, Université 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.