Notes de lectures: Closed-form Upper Bounds in Static Cost Analysis

Introduction

On cherche à borner le coût en une certaine ressource d'un programme P sur une entrée arbitraire x sans pour autant avoir à déterminer le x qui réalise cette borne ou avoir à calculer $P(x)$. Pour ce faire, on procède avec un programme P et un *modèle de coût* donné. On produit dans un premier temps des *relations de coûts (CR)* pour P. Ces relations décrivent le coût de P selon le modèle du calcul en fonction de la *taille* de son entrée x.

Les CR sont un formalisme pertinent pour l'analyse de coût pour les raisons suivantes:

- Elles sont *agnostiques* au langage et modèle de coût utilisés
- Elles couvrent de nombreuses classes de complexités algorithmiques, et sont donc *polyvalentes*
- On peut y capturer des notions *non-triviales* de coûts, même à posteriori en changeant le modèle.

La dynamique des CR est néanmoins non-triviale: l'abstraction des données du programme à leurs tailles crée une source de non-déterminisme: par exemple, quand on branche selon la valeur d'une variable, et que les deux branches n'ont pas de coût égaux, notre analyse statique se trouve face à une ambiguïté.

Dans la littérature, on utilise souvent des *relations de récurrences* à la place des relations de coûts. Elle y ressemblent fortement, mais sont déterministes. Cela a conduit à essayer de déléguer la recherche de formes fermées des relations de coûts par des systèmes experts de calculs symboliques (Mapple, Mathematica,etc). Mais, nous le verrons, cette approche n'a pas eu le succès escompté.

Notre nouvelle approche, basée sur l'analyse statique d'une sémantique de des CR à la manière de la programmation logique par contrainte, est la première permettant d'obtenir des formes fermées des CR automatiquement pour un grand nombre de programmes impératifs, représentatifs de ceux présent dans le monde réel.

Relations de Coûts

Obtention

On veut pouvoir inférer, depuis le code source d'un programme impératif structuré (avec `if`, `while`,etc) ou non-structuré (avec `goto`, `jump`,etc) les relations de tailles qui existent entre les données qu'il manipule. On procède en trois étapes:

1. On construit les *Control Flow Graph* du programme pour rendre les récurrences explicites.
2. En utilisant l'interprétation abstraite, on obtient des approximations des *relation de tailles* entres les différents appels récursifs. Les relations obtenues sont généralement linéaires (i.e. on travaille dans les domaine abstrait des polyèdres convexes), mais on peut utiliser un autre formalisme à condition d'avoir un solveur adapté plus bas dans la chaîne de traitement.
3. On interprète le programme dans le modèle de coût choisi, afin de générer les relations de coûts en elles-mêmes.

Syntaxe

$$
\begin{aligned}
r \in \mathbb{R}, n \in \mathbb{N}, v \in \mathbb{Z} \\
lin &::= v1 * x1 + ...vn * xn \\
c &::= (lin < lin | lin \leq lin | lin = lin)+ \\
exp &::= r | nat(lin) | exp + exp | exp * exp | exp - r \\
&\quad | exp^r | log_n(exp) | nexp | max(exp+) \\
cost &::= C(x_1...x_n) = exp + \sum(i = 1...n) D_i(y_{i,1}...y_{i,m}) where \varphi \\
CRS &::= cost+
\end{aligned}
$$

On notera que dans *cost*, les variables de *exp* et $y_{i,1}...y_{i,m}$ ne sont pas *définies* grâce à $x_1...x_n$ mais seulement reliées à elles par les contraintes présentes dans φ.

Sémantique

Les *Systèmes de récurrences de coûts (CRS)* sont définis de manière analogues au programmes de programmation logique: les $C(x_1...x_n)$ sont comme des clauses, et la partie droite de l'égalité peut être vue comme une disjonction (somme) de contraintes, testables directement ou par récurrence sur d'autres clauses.

La sémantique des *CRS* suit donc celle d'un programme en progammation logique, définie par des arbres d'évaluations. Pour évaluer un $C(v_1...v_n)$, on procède par étapes:

1. On choisie une des clauses du CRS qui a C en tête.

2. on instancie les variables libres de manière à satisfaire φ.
3. On calcule et accumule exp sous l'instanciation précédente
4. On itère sur tout les D_i

Quand on ne peut pas satisfaire φ, on a atteint un échec.

On définit alors la sémantique d'un CRS comme l'ensemble de arbres d'évaluations qu'il engendre. Le coût d'un arbre est défini par induction, et le coût du CRS est l'ensemble des coût de ses arbres d'évaluation.

On note $Trees(C(v_1...v_n), S)$ l'ensemble des arbres de l'évaluation de $C(v_1...v_n)$ dans le CRS S, et $Answer(C(v_1...v_n), S)$ l'ensemble des coûts engendrés.

Différences avec les relations de récurrences

Non-déterminisme L'abstraction des valeurs par leurs tailles rends les CR *très* non-déterministes, même dans les cas ou le programme ne l'est pas. Voir par exemple la fonction `filter`, laissée en exercice au lecteur.

Contraintes inexactes Les arguments des appels récursifs des CR sont liées aux arguments à la racine par des relations imprécises. On même peut avoir à les tirer dans les ensembles infinis. Au contraire, dans les RR, les arguments des appels récursifs sont connus si on connaît les arguments de l'appel précédent.

Arguments multiples Dans les RR, l'argument décroît selon un ordre bien fondé, ce qui garantie la terminaison: on parle de récurrence structurelle. Dans les CR, l'ordre bien-fondé peut exister sur une *relation de taille* arbitraire.

Les deux premiers cas sont une source importante de non-déterminisme du résultat. Les CR ne définissent pas des fonctions mais des relations entre les tailles des arguments et la "taille" du calcul associé. De plus, les systèmes experts de résolutions de récurrence ne peuvent généralement par inférer seuls l'argument décroissant de manière bien-fondé nécessaire à la résolution de la récurrence. Ces systèmes ne sont pas adaptés à la mise en forme close de CR.

On pourrait essayer de simplifier les CR pour en retirer le non-déterminisme. Hélas, des exemples réels simples montrent sans ambiguïté qu'il existe des CR non-déterministes dont la borne supérieure du coût n'est pas modélisable par par une CR déterministe.

Bornes Supérieures de Forme Fermées des Relations de Coûts

Dans une CRS S, on définit les /Bornes Supérieures de formes fermées d'une CR C comme une fonction $f : \mathbb{Z}^n \to \mathbb{R}^+$ telle que:

- $f(x_1...x_n) = exp$
- $\forall v_1...v_n, \forall r \in Answers(C(v_1...v_n), S), f(v_1...v_n) \geq r$

La méthode décrite dans l'article construit les bornes sup f en approchant par le haut le nombre et le coût individuel des noeuds internes et des feuilles des arbres d'évaluation de C. Formellement:

$f(x) = internes(x) * \text{cout-interne}(x) + feuilles(x) * \text{cout-feuille}(x)$

La méthode décrite dans l'article construit les bornes sup f en approchant par le haut le nombre et le coût individuel des noeuds internes et des feuilles des arbres d'évaluation de C. Formellement:

$f(x) = internes(x) * \text{cout-interne}(x) + feuilles(x) * \text{cout-feuille}(x)$

Bornage du Nombre de Noeuds

On borne les valeurs de $internes(x)$ et $feuilles(x)$ en donnant une borne supérieure de la hauteur $h(x)$ et du *facteur de branchement* b des arbres d'évaluation. Le facteur de branchement est facilement borné par le nombre maximal d'appels récursifs dans une équation pour C. Il est donc immédiatement calculable.

On cherche maintenant à borner la hauteur d'un arbre d'évaluation $T \in Trees(C(v_1...v_n), S)$. On applique un pré-traitement à S pour garantir un invariant sur T: C'est la *Mise en forme de récurrence directe*: Dans une descente dans T, les noeuds successifs représentent des appels imbriqués aux relations de coût idoines, étiquetés par les têtes des relations (C, D, ...); La forme de récurrence directe impose que si C apparaît comme un descendant de C, alors c'est un descendant direct. Ce résultat permet de réduire le bornage de $h(x)$ au bornage du nombre d'appels successifs à une relation de coût C. Une technique de passage en forme de récurrence directe est présentée plus loin.

Le bornage du nombre d'appel successifs à une relation de coût ou de récurrence à été étudié dans le cadre de l'analyse de terminaison. Il en résulte des algorithmes d'inférence de *fonctions de classement*, qui associe aux arguments des relations un *rang* dans un ordre bien-fondé. Si une fonction de classement f_C existe pour une relation de coût C, alors la hauteur de l'arbre engendré par $C(v_1...v_n)$ est bornée $f_C(v_1...v_n)$. Les fonctions de classement que nous utilisons sont linéaires, et inférées par l'algorithme décrit par Podelski & Rybalchenko dans *A complete method for the for synthesis of linear ranking functions (VMCAI04)*.

Une fois les bornes de la hauteur est du facteur de branchement établie, il suffit d'approximer l'arbre d'évaluation par un arbre complet de hauteur $h(v_1...v_n)$ et de facteur b.

Bornage du Coût par Noeud

Il reste maintenant à obtenir les fonctions cout-interne et cout-feuille. Premièrement, on utilise l'interprétation abstraite sur la forme de récurrence directe de S pour obtenir des approximations sûres des invariants entre les appels successifs

aux relations de coûts. On remarque alors que les coûts sont monotones en leurs composants *nat*. Il suffit alors de borner ces composants en utilisant les invariants calculés plus haut et les arguments à la racine de l'arbre d'évaluation. (Voir la functions `ub_exp` de l'article original pour l'algorithme exact). Les relations de coûts récursives de S engendrent les noeuds internes, et les non-récursives engendrent les feuilles. Les bornes supérieures calculées ici forment les fonctions cout-interne et cout-feuille.

Cas des Algorithmes *Diviser-pour-régner*

Dans les cas des algorithmes diviser-pour-régner, l'approximation noeud-par-noeud n'est pas assez précise. On borne alors le coût d'une relation de coût C par $C^+(x) = levels(x) * cout - level(x)$. Comme précédemment, on borne de nombre de niveau avec la hauteur de l'arbre.

Pour borner le coût par niveau, nous développons une caractérisation des CR *diviser-pour-régner* compatible avec notre analyse, un test automatique d'appartenance à cette classe de CR, et une méthode pour calculer une borne $cout - level$ pour ces CR.

Caractérisation Une CR est *diviser-pour-régner* si, pour tout ses arbres d'évaluation, le coût propre d'un niveau de son arbre d'évaluation des supérieur ou égal au coût propre du niveau directement en dessous

Calcul On calcule, pour une CR C, les paires (exp, exp') des coûts abstraits propres exp de chaque équation la définissant, et de la somme exp' des coût abstraits propres de leurs enfants respectifs. Pour prouver que C est diviser-pour-régner, il suffit de prouver que pour tout assignement satisfaisant les contraintes locales de ces noeuds, le coût de C est au moins égal au coût total de ces enfants.

Bornage On borne alors le coût d'un niveau par le coût de la racine, qui est le niveau de coût maximal. L'algorithme de calcul est le même que pour le calcul de cout-interne.

Mise en Forme de Récurrence Directe des Relations de Coûts

On effectue une analyse de composantes fortement connexes dans les CRS, . Il y a un lien entre C et D si D intervient à droite dans une équation définissant C. Ensuite, le cas échéant, on inline selon ces composantes pour rendre toutes les définitions récursives *directement* récursives.

Incomplétude de l'Analyse de Coût

Le problème de l'arrêt se réduit à un calcul de coût dans un modèle de temps. Donc notre analyse sera forcément incomplète. Mais même quand un pro-

gramme admet un coût fini dans le modèle considéré, notre analyse comporte des sources de pertes de précision:

- La transformation du programme en système de relations de coût utilise des techniques d'interprétation abstraites qui donnent des résultats approximatifs sur l'aliasing ou les relations de tailles. On notera que certain de ces problèmes sont aussi indécidables.
- L'obtention de bornes supérieures pour les CRS est aussi indécidable, même pour des version appauvris des CR. On distingue les pertes de précisions suivantes dans l'analyse:
  - La mise en forme de récurrence directe n'est pas systématiquement faisable, notamment dans le cas de définition mutuellement récursives.
  - Certaines CR n'ont pas de fonction de classement linéaires. On pourrait étendre nos fonctions de classement à des classes plus expressives, mais le besoin ne s'est pas fait ressentir en pratique.
  - Les invariants recherchés lors du bornage du coût par noeud peuvent ne pas être linéaires. Pour intégrer des classes d'invariants plus intéressantes, il conviendrait d'adapter la procédure de maximisation.

Travaux Connexes & Conclusion

Dès le système *METRIC* de Wegbriet, on cherche à obtenir des CR à partir de programme. Mais *METRIC* se limite aux RR, qui sont déterministes, et déjà, les solutions exactes sous formes fermées sont hors de portée. On se contente alors de bornes supérieures. On retrouve ensuite les travaux de Métayer avec *ACE*, de Rosenthal avec l'interprétation abstraite, de Walder avec l'analyse de rigueur au service de l'analyse de ressources, et Sands et ses théories d'équivalence de coûts. Notre contribution consiste à considérer des relations (1) non-déterministes, (2) augmentées de contraintes de tailles sur leurs arguments plus large que la simple égalité définitionnelle.

On distingue deux classes de système d'inférence de coût par CR. Les systèmes **algébriques** sont basés sur les nombreux travaux sur la résolution de relations de récurrences, soit en implémentant un solveur limité en tant que composant de l'analyseur, soit en déléguant la résolution à un *Computer Algebra System*. On citera *Mathematica, Mapple, Maxima*…

Les systèmes **transformationels**, eux, représentent les CR comme des programmes fonctionnels, et la mise en forme close correspond à une réécriture du programme de base augmenté d'une notion de consommation de ressource. *ACE* fût le premier de ces systèmes, basé sur une grande quantité de règles de réécriture manuelles. Depuis, des avancées ont permis de trouver des bornes sous forme close pour certaines classes de programmes (cf. Rosenthal FPCA'89).

Les deux approches souffrent de problèmes en cas de non-déterminisme des CR, ce qui est assez courant dans le monde réel, et les résultats obtenus sont rarement

utiles s'il sont trop coûteux à obtenir ou trop algébriquement complexes: que faire d'un coût de 5*sqrt(5)*(1-sqrt(5))$^{(x+1)}$ - 5*sqrt(5)*(1+sqrt(5))$^{(x+1)}$ + … ? Au contraire, l'approche décrite dans cette article produit des résultats corrects et simples même en cas de non-déterminisme, mais donc plus approximatifs.

On note l'existence de *PURRS*, qui ressemble à notre travail, mais avec uniquement des CR déterministes, et des travaux de Marion&al. sur les bornes polynomiales des tailles de piles pour les programmes fonctionnels.

Notre travail mets en lumière la pertinence des CR comme *langage cible* des analyses de coûts pour une analyse agnostique du langage source. Nous avons étendu notre travail à des CR prenant en compte les phénomènes de libération mémoire par un ramasse-miette, et donc le bornage de tas.