École Normale Supérieure
Département d'informatique
Rapport de stage de L3

Analyse de fréquence d'activation

Quentin Vermande
DI 2018
quentin.vermande@ens.fr

17 Juin 2019 - 31 Juillet 2019

Encadrants :
Pascal Sotin (Tuteur)
Armelle Bonenfant

Équipe TRACES
IRIT Toulouse (Institut de Recherche en Informatique de Toulouse)
118 route de Narbonne
31062 Toulouse cedex 9

Sommaire

1 Introduction

1.1 Équipe d'accueil

L'équipe TRACES fait partie de l'IRIT, l'un des plus grands laboratoires de recherche français avec environ 800 membres. Cette équipe, composée de 7 membres permanents, s'intéresse à l'estimation de bornes supérieures du temps d'exécution de tâches temps-réel critiques. Le temps d'exécution d'un programme dépend des données en entrée. Or le domaine de variation de ces données est souvent trop large pour envisager de tester tous les cas. C'est pourquoi les recherches de ces dernières années se sont orientées vers des méthodes permettant d'évaluer le temps d'exécution pire cas (WCET : Worst-Case Execution Time) à partir de mesures de segments de code (typiquement des blocs de base). Cette évaluation comporte trois volets : une analyse statique du code permet d'identifier les chemins d'exécution possibles ; une étude du matériel cible détermine le temps d'exécution des blocs de base ; enfin, les résultats de ces deux analyses sont combinés pour calculer une borne supérieure du temps d'exécution. Nos travaux s'inscrivent dans le second volet : analyse du comportement temporel du matériel.

1.2 Problématique

L'analyse de fréquence d'activation a pour but de déterminer (une borne supérieure sur) le nombre de fois que l'exécution d'un programme atteint un point de contrôle donné durant l'exécution d'une boucle. Elle s'inscrit dans le cadre de l'analyse statique des programmes, qui consiste à trouver des propriétés d'un programme étant donné son code source.

Cette analyse est plus difficile que le calcul d'une borne sur le nombre maximal d'itérations d'une boucle, puisque ce calcul est exactement l'analyse de fréquence d'activation du corps de la boucle. Elle est aussi plus facile que la détermination d'invariants de boucle, puisqu'il suffit d'incrémenter un compteur au point d'intérêt pour obtenir sa fréquence d'activation. Ces deux problèmes sont indécidables et actuellement des domaines actifs de recherche [1, 2, 3, 4, 5]. Le problème réside dans le fait que les boucles peuvent être itérées un grand nombre de fois, voire indéfiniment, ce qui exclut les simples comptages et qu'il est difficile d'automatiser les raisonnements logiques. En outre, l'idéal

serait d'avoir une expression symbolique du résultat, en fonction des valeurs des variable au début de la boucle.

1.3 Contributions

Le travail présenté ci-après a pour objectif de proposer une analyse la plus générale possible d'un programme et facilement adaptable à n'importe quel outil de calcul d'invariants. Il considère comme donné un outil calculant des invariants de boucle sous forme de relations entre les variables. Pour ce faire a été introduit un langage simple d'arithmétique. Dans ce langage, une formule décrivant la boucle est transformée en contraintes linéaires sur les variables du programme. Puis l'outil *barvinok* calcule la fréquence d'activation du point d'intérêt par décompte des points entiers du polyèdre obtenu.

Ce travail est implémenté dans une bibliothèque, appelée AUBE (pour Activation analysis Using Barvinok's Enumeration). Quand bien même l'outil d'analyse se veut généraliste, il n'est pas possible de traiter des fonctions quelconques. Les traitements implémentés concernent des fonctions provenant de deux thèmes de recherche de l'équipe d'accueil. Les exponentielles et les coefficients binomiaux apparaissent dans la sortie de l'outil *Choregies* qui calcule entre autres des invariants de boucle. Les modulos et les parties entières interviennent dans les analyses de cache.

2 Exemple : analyse d'un programme non-linéaire simple

Cette section décrit l'analyse d'un exemple, donné ci-dessous. L'objectif est de montrer les principes appliqués dans le traitement d'un programme et formalisés par la suite, ainsi que les difficultés rencontrées.

Considérons la boucle :

```
int a = 0;
while(x < N)
{
        if(x%4 == 0)
                ++a;
        x += y;
        ++y;
}
```

À la fin de l'exécution de la boucle, a désigne le nombre de fois que la condition `x%4 == 0` est évaluée à vrai au cours de l'exécution du programme. L'analyse consiste ici à déterminer la valeur de a (ou une borne supérieure).

Notons n le compteur de boucle et x_0 et y_0 les valeurs initiales de x et y. La première étape de l'analyse consiste à calculer une accélération de la boucle, i.e. à déterminer des contraintes sur les variables (y compris le compteur de boucle) valables au début de chaque tour de boucle (avant l'évaluation de la garde). Ces contraintes sont écrites dans un langage d'arithmétique rationnelle simple, présenté dans la section 3.1. On peut calculer l'accélération facilement à la main :

$$\varphi = 0 \leq n \wedge y = y_0 + n \wedge x = x_0 + ny_0 + \binom{n}{2}$$

Ici, φ donne la positivité de n et les expressions de x et y en fonction de n et des paramètres. On ajoute à cette formule les contraintes indiquant que l'exécution a parcouru le chemin de la condition d'entrée de la boucle à l'incrémentation de a et on quantifie sur les variables autres que n :

$$\psi = \exists x, y \in \mathbb{Z}, (\varphi \wedge x < N \wedge x\%4 = 0)$$

Alors, la valeur finale de a est : $\#\{n \in \mathbb{Z}, \psi\}$. Ce cardinal dépend de x_0, y_0 et N qui sont les paramètres de la boucle. Lorsque $\{n \in \mathbb{Z}, \psi\}$ est un polyèdre paramétrique (cf. section 3.1), l'outil *barvinok* permet d'effectuer le décompte. Il faut transformer ψ en une formule ξ telle que :

— ξ est une conjonction de contraintes linéaires en les variables et les paramètres

— $\forall n \in \mathbb{Z}, \psi(n) \rightarrow \xi(n)$, i.e. $\{n \in \mathbb{Z}, \psi(n)\} \subset \{n \in \mathbb{Z}, \xi(n)\}$

Pour ce faire, on commence par supprimer le modulo par division euclidienne (cf. section 3.2). Ici, la condition $x\%4 = 0$ revient à dire que x s'écrit $4q$, avec q entier :

$$\exists x, y, q \in \mathbb{Z}, \varphi \wedge x < N \wedge x = 4q$$

Puis on remplace les occurences de x et y par leur expression ($y = y_0 + n$ et $x = 4q$) :

$$\exists q \in \mathbb{Z}, 0 \leq n \wedge 4q = x_0 + ny_0 + \binom{n}{2} \wedge 4q < N$$

À ce stade, la seule contrainte non-linéaire est $4q = x_0 + ny_0 + \binom{n}{2}$. Cette contrainte pose une importante difficulté car elle fait apparaître plusieurs variables. On ne peut pas traiter correctement le polynôme de degré 2 en n (cf. section 3.2). Il y a deux possibilités : soit on remplace q par $\frac{1}{4}(x_0 + ny_0 + \binom{n}{2})$, ce qui revient à perdre l'information $x\%4 = 0$, soit on écrit $ny_0 = \frac{1}{4}((n+y_0)^2 - (n-y_0)^2)$ et on ajoute une variable quantifiée existentiellement par carré (cf.section 3.2). Dans AUBE, c'est cette deuxième solution qui est retenue. Cela donne :

$$\exists q, a, b, c \in \mathbb{Z}, 0 \leq n \wedge 4q = x_0 + \frac{1}{4}(a-b) + \frac{1}{2}(c-n) \wedge a = (n+y_0)^2 \wedge b = (n-y_0)^2 \wedge c = n^2 \wedge 4q < N$$

Les carrés sont traités par approximation convexe : $\forall x \in \mathbb{R}, 2x - 1 \leq x^2$. On obtient :

$$\exists q, a, b, c \in \mathbb{Z}, 0 \leq n \wedge 4q = x_0 + \frac{1}{4}(a-b) + \frac{1}{2}(c-n) \wedge 2(n+y_0) \leq a \wedge 2(n-y_0) \leq b \wedge 2n \leq c \wedge 4q < N$$

L'outil *barvinok* fournit le résultat final, en l'occurence $+\infty$. L'analyse de la borne sur le nombre d'exécutions de la boucle (i.e. la même analyse sans la contrainte $x\%4 = 0$) donne :

Condition	Résultat
$8x_0 + 4y_0 \leq 1 + 4y_0^2 + 8N$ $2y_0 \leq 1 + 2\sqrt{y_0^2 - y_0 - 2x_0 + 2N + \frac{1}{4}}$	$1 - y_0 + \lfloor \frac{1}{2}(1 + \lfloor 2\sqrt{y_0^2 - y_0 - 2x_0 + 2N + \frac{1}{4}} \rfloor) \rfloor$
Sinon	0

3 Décompte de points entiers dans un ensemble

Cette section présente le formalisme proposé et les résulats théoriques utilisés pour l'analyse.

3.1 Polyèdres

Les ensembles considérés par *barvinok* sont des polyèdres. La définition proposée est basée sur les travaux de Verdoolaege et al. [6] :

Définition 1 (polyèdre paramétrique) *Soient $d \in \mathbb{N}^*$ et $p \in \mathbb{N}$. Un polyèdre paramétrique de dimension d à p paramètres est une application P de $\mathbb{Q}^p$ dans l'ensemble des polyèdres de dimension d telle qu'il existe $n \in \mathbb{N}, A \in \mathcal{M}_{n,d}(\mathbb{Q}), B \in \mathcal{M}_{n,p}(\mathbb{Q})$ et $C \in \mathcal{M}_{n,1}(\mathbb{Q})$ tels que :*

$$\forall (x_1, ..., x_p) \in \mathbb{Q}^n, P(x_1, ..., x_p) = \{(y_1, ..., y_d) \in \mathbb{Q}^d, A \begin{pmatrix} y_1 \\ \vdots \\ y_d \end{pmatrix} \leq B \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} + C\}$$

L'accélération de la boucle donne un ensemble de contraintes sur les variables et les paramètres, dans le langage suivant :

Définition 2 (arithmétique rationnelle) *On définit le langage de l'arithmétique (non-linéaire) rationnelle :*

- *les symboles de constante sont les rationnels*
- *les symboles de fonction sont $+$ (binaire, infixe), $*$ (binaire, infixe), $/$ (binaire, infixe), $\%$ (modulo informatique, binaire, infixe), $\hat{}$ (exposant entier, binaire, infixe), log (binaire), floor (unaire) et $\binom{\cdot}{\cdot}$ (coefficient binomial)*
- *les symboles de relation sont $=$ (binaire, infixe), $\leq$ (binaire, infixe) et $<$ (binaire, infixe)*

On se place dans la théorie classique de l'arithmétique (i.e. Th$(\mathbb{Q})$ avec les interprétations classiques des symboles de constante, de fonction et de relation).

L'objectif est de transformer les contraintes données en entrée en contraintes linéaires, c'est-à-dire ne faisant apparaître que des termes linéaires :

Définition 3 (terme linéaire) *Les termes linéaires sont définis par induction par :*

- *toute variable et tout symbole de constante sont des termes linéaires*
- *si s et t sont des termes linéaires, alors $s + t$ aussi*
- *si c est un symbole de constante et t est un terme linéaire, alors $c * t$ est un terme linéaire*

Définition 4 (terme entier) *Un terme est dit entier si le remplacement de toutes les variables par des entiers donne un entier. En particulier,*

- *toute constante entière et toute variable est un terme entier*
- *toute combinaison linéaire à coefficients entiers de termes entiers est un terme entier*
- *toute partie entière et tout coeficient binomial est un terme entier*
- *tout modulo et toute puissance entre deux termes entiers est un terme entier*

Proposition 1 (représentation des polyèdres paramétriques par des formules) *Soient $d \in \mathbb{N}^*, p \in \mathbb{N}$ et P un polyèdre paramétrique de dimension d à p paramètres. Alors il existe $k, n \in \mathbb{N}$, $t_1, ..., t_n$ des termes linéaires sur $v_1, ..., v_{p+d+k}$ et $r_1, ..., r_n \in \{=, \leq\}$ tels que :*

$$\forall (v_1, ..., v_p) \in \mathbb{Q}^p, P(v_1, ..., v_p) = \{(v_{p+1}, ..., v_{p+d}) \in \mathbb{Q}^d, \exists v_{p+d+1}, ..., v_{p+d+k}, \bigwedge_{i=1}^{n} t_i \ R_i \ 0$$

Démonstration 1 *On écrit P comme la projection d'un polyèdre de dimension $d+k$ sur ses d premières dimensions.*

Exemple 1 *Considérons le programme :*

```
int a = 0;
int N = rand();
for(int i = 0; i < N; ++i)
        if(i%4 == 0) ++a;
```

L'accélération de la boucle donne qu'après n itérations, on a $i = n$.
Alors la valeur finale de a est :
$\#\{n \in \mathbb{N}, \exists i, i = n \wedge i < N \wedge i\%4 = 0\} = \#\{n \in \mathbb{N}, n \leq N - 1 \wedge n \in 4\mathbb{Q}\}$
$= \#\{n \in \mathbb{N}, \exists a \in \mathbb{N}, n \leq N - 1 \wedge n = 4a\}$
Ce cardinal peut être calculé automatiquement et vaut $\#\{a \in \mathbb{N}, 4a \leq N-1\} = max(0, \lfloor \frac{N-1}{4} \rfloor + 1)$.

Exemple 2 *Considérons le programme :*

```
int a = 0;
int s, N = rand();
for(int i = 0; i < N; i += s)
        ++a;
```

L'accélération de la boucle donne qu'après n itérations, on a $i = s * n$.
Alors la valeur finale de a est :
$a = \#\{n \in \mathbb{N}, \exists i \in \mathbb{N}, i = s * n \wedge i < N\} = \#\{n \in \mathbb{N}, s * n \leq N - 1\}$
$= \#\{n \in \mathbb{N}, (s \leq -1 \wedge Q \leq n) \vee (s = 0 \wedge 0 \leq N - 1) \vee (1 \leq s \wedge n \leq P)\}$ (où l'on a posé les paramètres $P = \lfloor \frac{N-1}{s} \rfloor$ et $Q = \lceil \frac{N-1}{s} \rceil$)

$$= \begin{cases} \#\{n \in \mathbb{N}, Q \leq n\} & \text{si} \quad s \leq -1 \\ \#\{n \in \mathbb{N}, 0 \leq N - 1\} & \text{si} \quad s = 0 \\ \#\{n \in \mathbb{N}, n \leq P\} & \text{si} \quad 1 \leq s \end{cases}$$

Les trois cas peuvent être résolus automatiquement et on obtient :

s	N	résultat
≤ -1		$+\infty$
$= 0$	≥ 1	$+\infty$
$= 0$	≤ 0	0
≥ 1		$P + 1$

3.2 Transformation d'ensembles en polyèdres

Cette sous-section décrit les opérations effectués lors de la transformation d'une formule.

Soit φ la formule donnée par l'accélération de la boucle. φ s'écrit sous la forme $\bigwedge_{i=1}^{n} t_i \; R_i \; s_i$ où $n \in \mathbb{N}$, $t_1, ..., t_n, s_1, ..., s_n$ sont des termes et $R_1, ..., R_n \in \{=, \leq, <\}$. Supposons donnés $d + 1$ la dimension de l'espace représenté par la formule, q le nombre de variables implicitement quantifiées existentiellement et p le nombre de paramètres. En pratique, pour l'analyse de fréquence d'activation, on ne considère que v_0 le compteur de boucle, donc $d = 0$. On considère dorénavant que $v_0, ..., v_d$ sont les coordonnées d'un point de l'espace considéré, que $v_{d+1}, ...v_{d+q}$ sont les variables quantifiées existentiellemet et $v_{d+q+1}, ...v_{d+q+p}$ sont les paramètres, $v_0, ..., v_{d+p+q}$ étant les seules variables apparaissant dans la formule. On cherche à transformer cette formule en une formule (plus faible) représentant un polyèdre. Dorénavant, on oublie le contexte de l'analyse de fréquence d'activation pour ne travailler que sur des formules logiques et des polyèdres. On dispose de plusieurs stratégies pour transformer φ en une formule représentant un polyèdre.

Considérons φ de la forme ci-dessus. Dans ce qui suit, une variable (resp. variable quantifiée existentiellement, resp. paramètre) désigne une variable (resp. variable quantifiée existentiellement, resp. paramètre) de φ. Une nouvelle variable (resp. variable quantifiée existentiellement, resp. paramètre) est une variable (resp. variable quantifiée existentiellement, resp. paramètre) n'apparaissant pas dans φ. On autorise les contraintes ne faisant apparaître que des paramètres à ne pas être linéaires. On autorise aussi les disjonctions.

La transformation la plus simple consiste à remplacer un terme non-linéaire ne contenant que des paramètres par un nouveau paramètre, selon les deux lemmes suivants :

Lemme 1 (extraction de termes) *Soit t un terme entier. Soient v une nouvelle variable et φ' obtenue à partir de φ en remplaçant toutes les occurences de t par v. Alors on a l'équivalence :*

$$\forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, \varphi \Leftrightarrow \exists v, (\varphi' \wedge v = t)$$

Lemme 2 (gestion des contraintes non-linéaires ne faisant apparaître que des paramètres) *Soient t un terme et v et w deux nouvelles variables. Soit φ' la formule obtenue à partir de φ en remplaçant toutes les contraintes de la forme $s\ R\ t$ où s est un terme entier par :*

- *si $R \in \{=\}, s = v$*
- *si $R \in \{\leq, <\}, s \leq w$.*

Alors on a l'équivalence

$$\forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, \varphi \Leftrightarrow \exists v, w, (\varphi' \wedge v = t \wedge w = \lfloor t \rfloor)$$

Remarque 1 *Ces deux lemmes sont utilisés lorsque t est non-linéaire et que toutes les variables apparaissant dans t sont des paramètres. v est alors un nouveau paramètre et la contrainte $v = t$ ou $v = \lfloor t \rfloor$ est retenue en dehors de la formule.*

De manière générale, on peut éliminer toute fonction injective dans une égalité et toute fonction strictement monotone dans une inégalité de manière exacte :

Lemme 3 (gestion des termes non-linéaires) *Soient s, t et u trois termes et ψ une formule. Pour $x_0, .., x_{d+q+p} \in \mathbb{Z}$, notons $\overline{x} = (x_0, ...x_{d+q+p})$. Supposons qu'il existe $f : \mathbb{R} \to \mathbb{R}$ telle que, pour tout $\overline{x} \in \mathbb{Z}^{d+q+p+1}$ tel que $\psi(x_i/v_i)$, on ait $t(x_i/v_i) = f(u(x_i/v_i))$ (informellement, $t = f(u)$). Soit v une nouvelle variable. Supposons qu'il existe un terme inv sur $v, v_0, ..., v_{d+q+p}$ tel que pour tout $\overline{x} \in \mathbb{Z}^{d+q+p+1}$ tels que $\psi(x_i/v_i)$, on ait $inv(t/v)(x_i/v_i) = u(x_i/v_i)$ (informellement $inv(t) = u$ ou encore $inv = f^{-1}$). Alors :*

- *on a l'équivalence :*

$$\forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, (\psi \wedge s = t) \Leftrightarrow (\psi \wedge v_0 = inv(s/v_0))$$

- *si f est strictement croissante, alors :*
  - *s'il existe un terme inf sur les variables $v_0, ..., v_{d+q+p}$ tel que pour tout $\overline{x} \in \mathbb{Z}^{d+q+p+1}$, on ait $inf(s/v_0)(x_i/v_i) = \inf(\{x \in f(\mathbb{R}), s \leq x\})$, alors on a l'équivalence :*

$$\forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, (\psi \wedge s \leq t) \Leftrightarrow (\psi \wedge inv(inf(s/v_0)/v_0) \leq v_0)$$

  - *s'il existe un terme sup sur les variables $v_0, ..., v_{d+q+p}$ tel que pour tout $\overline{x} \in \mathbb{Z}^{d+q+p+1}$, on ait $sup(s/v_0)(x_i/v_i) = \sup(\{x \in f(\mathbb{R}), x \leq s\})$, alors on a l'équivalence :*

$$\forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, (\psi \wedge s < t) \Leftrightarrow (\psi \wedge inv(sup(s/v_0)/v_0) < v_0)$$

- *si f est strictement décroissante, alors :*
  - *s'il existe un terme inf sur les variables $v_0, ..., v_{d+q+p}$ tel que pour tout $\overline{x} \in \mathbb{Z}^{d+q+p+1}$, on ait $inf(s/v_0)(x_i/v_i) = \inf(\{x \in f(\mathbb{R}), s \leq x\})$, alors on a l'équivalence :*

$$\forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, (\psi \wedge s \leq t) \Leftrightarrow (\psi \wedge v_0 \leq inv(inf(s/v_0)/v_0))$$

  - *s'il existe un terme sup sur les variables $v_0, ..., v_{d+q+p}$ tel que pour tout $\overline{x} \in \mathbb{Z}^{d+q+p+1}$, on ait $sup(s/v_0)(x_i/v_i) = \sup(\{x \in f(\mathbb{R}), x \leq s\})$, alors on a l'équivalence :*

$$\forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, (\psi \wedge s < t) \Leftrightarrow (\psi \wedge v_0 < inv(sup(s/v_0)/v_0))$$

Remarque 2 *Ce lemme n'est utilisé que lorsque s ne fait intervenir que des paramètres.*

Ce lemme sera illustré par quelques corollaires.

Exemple 3 *Soit le programme :*

```
int a = 0;
int N = rand();
for(int i = 1; i < N; i *= 2)
        ++a;
```

L'accélération de la boucle donne $i = 2^n$. Alors on a la valeur finale de a :
$a = \#\{n \in \mathbb{N}, \exists i \in \mathbb{N}, i = 2^n \wedge i < N\} = \#\{n \in \mathbb{N}, 2^n \leq N-1\}$

Or $x \mapsto 2^x$ est bijective de $\mathbb{R}$ dans $\mathbb{R}^{+*}$ de réciproque $\log_2$. Donc, pour $n, N \in \mathbb{Z}$:

$$\begin{aligned} 2^n \leq N-1 &\Leftrightarrow 0 < N-1 \wedge 2^n \leq N-1 \text{ (car } 0 < 2^n) \\ &\Leftrightarrow 1 < N \wedge n \leq \log_2(N-1) \end{aligned}.$$

Donc on a la disjonction de cas :
- si $N \leq 1$, alors $a = 0$
- si $1 < N$, alors $a = \#\{n \in \mathbb{N}, n \leq \log_2(N-1)\} = \lfloor \log_2(N-1) \rfloor + 1$.

Les produits peuvent être éliminés de deux manières différentes selon la présence de paramètres ou non. La première est exacte tandis que la seconde introduit une (importante) approximation.

Corollaire 1 (gestion optimisée des produits) *Soient $R \in \{=, \leq, <\}$ et s, t et u trois termes. Alors on a l'équivalence :*

$$\forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, s \ R \ t * u \Leftrightarrow (t < 0 \wedge u \ R \ \frac{s}{t}) \vee (t = 0 \wedge s \ R \ 0) \vee (0 < t \wedge \frac{s}{t} \ R \ u)$$

Lemme 4 (gestion approchée des produits) *Soient s et t deux termes. Soient v et w deux nouvelles variables. Soit φ' la formule obtenue à partir de φ en remplaçant toutes les occurences de $s * t$ par $\frac{1}{4}(v - w)$. Alors $\varphi \Leftrightarrow \varphi' \wedge v = (s+t)^2 \wedge w = (s-t)^2$.*

Certains polynômes de degré deux peuvent même être traités de manière exacte :

Corollaire 2 (gestion des polynômes de degré deux) *Soient v une variable et a, b et c trois termes dans lesquels v n'apparaît pas. Alors on a les équivalents :*

$$\begin{aligned} \forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, av^2 + bv + c = 0 \ &\Leftrightarrow \ (a = 0 \wedge bv + c = 0) \\ &\vee \ (4ac \leq b^2 \wedge (v = \tfrac{-b+\sqrt{b^2-4ac}}{2a} \\ &\vee \ v = -\tfrac{b+\sqrt{b^2-4ac}}{2a})) \end{aligned}$$

Et, pour $R \in \{\leq, <\}$:

$$\begin{aligned} \forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, 0 \ R \ av^2 + bv + c \ &\Leftrightarrow \ (a = 0 \wedge 0 \ R \ bv + c) \\ &\vee \ (0 < a \wedge (b^2 \ R \ 4ac \vee v \ R \ -\tfrac{b+\sqrt{b^2-4ac}}{2a} \vee \tfrac{-b+\sqrt{b^2-4ac}}{2a} \ R \ v)) \\ &\vee \ (a < 0 \wedge 4ac \ R \ b^2 \wedge -\tfrac{b+\sqrt{b^2-4ac}}{2a} \ R \ v \wedge v \ R \ \tfrac{-b+\sqrt{b^2-4ac}}{2a}) \end{aligned}$$

Les modulos et les parties entières de fractions peuvent être éliminés de manière exacte par division euclidienne :

Lemme 5 (gestion des modulos) *Soient s et t deux termes. Soit q une nouvelle variable. Soit φ' la formule obtenue en remplaçant toutes les occurences de $s\%t$ par $(s - q * t)$. Alors on a l'équivalence :*

$$\begin{aligned} \forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, \varphi \Leftrightarrow \varphi' \wedge \ &(\ 0 \leq s \wedge 0 < t \wedge 0 \leq s - q * t \wedge s - q * t < t \\ &\vee \ 0 \leq s \wedge t < 0 \wedge 0 \leq s - q * t \wedge s - q * t < -t \\ &\vee \ s < 0 \wedge 0 < t \wedge -t < s - q * t \wedge s - q * t \leq 0 \\ &\vee \ s < 0 \wedge t < 0 \wedge t < s - q * t \wedge s - q * t \leq 0) \end{aligned}$$

Lemme 6 (gestion de la partie entière) *Soient s et t deux termes et q une nouvelle variable. Soit φ' la formule obtenue à partir de φ en remplaçant toutes les occurences de $\lfloor \frac{s}{t} \rfloor$ par q. Alors on a l'équivalence :*

$$\forall v_0, ..., v_{d+q+p} \in \mathbb{Z}, \varphi \Leftrightarrow \varphi' \wedge 0 \leq s - q * t \wedge ((0 < t \wedge s - q * t < t) \vee (t < 0 \wedge s - q * t < -t))$$

4 AUBE (Activation analysis Using Barvinok's Enumeration)

AUBE est une nouvelle bibliothèque qui implémente les résultats ci-dessus.

4.1 Entrée et sortie de la bibliothèque

Le module principal `Analysis` contient une fonction `of_formule` qui prend en entrée une formule sur le langage de l'arithmétique rationnelle décrit section 3.1 et retourne un ensemble de couples `contrainte, résultat`, où `résultat` est une borne sur le nombre de points entiers dans l'ensemble décrit par la formule, lorsque les paramètres vérifient la contrainte `contrainte`.

Les variables apparaissant dans la formule sont réparties en trois catégories : les variables correspondant aux dimensions du polytopes, les variables quantifiées existentiellement et les paramètres. Ces deux premières catégories sont implémentées sous un seul objet. Les variables sont codées par un entier. Les premières variables sont considérées comme non quantifiées, toutes les suivantes sont quantifiées. Par conséquent, la formule donnée en entrée est une formule sans quantificateurs, où les premières variables correspondent aux dimensions du polyèdre et les autres sont implicitement quantifiées.

Dans le cas particulier de l'analyse de fréquence d'activation, la variable de boucle est la seule variable non quantifiée, les variables du programme sont toutes quantifiées existentiellement et les valeurs des variables avant la première exécution de la boucle sont les paramètres.

4.2 Calcul de cardinal

Les transformations de la formule sont codées dans le module `Transformation`. Chaque fonction du module a une fonction précise, qui peut servir à éliminer les termes non-linéaires de la formule ou à mettre la formule sous une forme où il est plus simple de faire certaines transformations :

- `elimQuantifiees` : pour chaque variable v, s'il existe une contrainte de la forme $v = t$ où t est un terme entier, remplace chaque occurence de v par t et supprime la contrainte $v = t$.
- `elimModulos` : pour chaque terme de la forme $t\%s$, effectue la division euclidienne de t par s en ajoutant le quotient en variable quantifiée
- `elimFloor` : pour chaque terme de la forme $\lfloor \frac{t}{s} \rfloor$ où s et t sont des termes entiers, effectue la division euclidienne de t par s en ajoutant le quotient en variable quantifiée
- `isolateParams` : sépare les paramètres des autres variables en utilisant la commutativité et l'associativité de la somme et du produit pour avoir les paramètres à gauche et les autres variables à droite
- `elimPdt` : pour chaque terme de la forme st, divise par s si s ne contient que des paramètres, remplace st par $a - b$ en ajoutant les contraintes $2s - 1 \leq a$ et $2t - 1 \leq b$ sinon si s et t sont entiers
- `elimRat` : multiplie chaque terme par une constante, de sorte qu'il n'y ait plus que des coefficients entiers
- `introduceParameters` : remplace chaque terme non-linéaire ne contenant que des paramètres par un nouveau paramètre (ou sa partie entière)
- `extractOr` : met la formule sous forme normale disjontive
- `elimPolynomeDeg2` : pour chaque terme t représentant un polynôme de degré 2 dont les coefficients des termes et l'indéterminée est une variable v, si toutes les variables dans les coefficients sont des paramètres, met le polynôme sous forme canonique et prend la racine carrée du carré et de la constante restante (cf. section 3.2). Sinon, remplace un carré par une nouvelle variable et une contrainte (cf. section 3.2)
- `elimStrictIneq` : remplace toutes les contraintes de la forme $0 < t$ par une contrainte de la forme $0 \leq s$.

Ces transformations sont combinées dans la fonction `of_formule` du module `Polyhedron` qui prend en entrée la formule et le nombre de variables de chaque type et renvoie une disjonction de sous-formules, chaque sous-formule représentant un polyèdre. Cette formule est donnée à la fonction `of_formule` du module `Analysis` qui appelle *barvinok* sur chaque polyèdre et récupère chaque sortie.

5 Travaux connexes

Accélération. Dans [4], Kincaid et al. calculent l'accélération d'une boucle dans le domaine abstrait *wedge* des conjonctions de contraintes dans un langage d'arithmétique similaire à celui proposé section 3.1. Le corps de la boucle est transformé en formule, elle-même approximée par un *wedge*, à savoir une conjonction de contraintes non-linéaires en les variables. Ce *wedge* est renforcé par des techniques de déduction utilisant des polyèdres et des bases de Gröbner. Puis, de ce *wedge* sont extraites des relations de récurrence qui sont résolues par l'algorithme OCRS, basé sur le *discrete operational calculus* de Berg (1967).

Dans [3], Kincaid et al. calculent cette accélération par une formule logique dans le langage décidable *exponential-polynomial rational arithmetic* (EPRA). Les boucles linéaires (voire polynomiales solvables) sont approximées par des boucles linéaires à valeurs propres périodiques rationnelles, qui sont ensuite transformées en formules logiques par le calcul de valeurs propres généralisées.

Complexité et borne sur le nombre d'itérations d'une boucle. Dans [1], Brockschmidt et al. représentent le programme sous forme de graphe de flot de contrôle. L'algorithme alterne entre calcul de bornes sur le nombre d'exécutions des transitions et de bornes sur les variables. La complexité est calculée à l'aide de *polynomial ranking functions* : à chaque point du programme est associé un polynôme en les variables de sorte que la valeur observée en le point du programme exécuté soit décroissante au cours de l'exécution. Pour les bornes sur les variables, l'analyse calcule un *result variable graph* qui décrit les dépendances entre les variables et des bornes locales.

Sinn et al. poursuivent le même objectif dans [5] avec la même représentation du programme. Chaque arête est étiquettée avec un ensemble de contraintes de la forme $v \leq t + c$ où v est une variable, t est une variable ou un symbole de constante et $c \in \mathbb{N}$. L'algorithme calcule ensuite des bornes locales sur le nombre d'exécutions d'une transition en se basant sur les variables décrémentées. Puis le résultat est calculé immédiatement à partir de ces bornes locales.

Dans [2], Flores-Montoya prend un objectif un peu plus large qui consiste à calculer une fonction de coût sur un programme. L'information extraite du programme est représentée sous forme d'expressions du coût associées à des contraintes sur les variables, et ce pour chaque partie du programme (i.e. boucle). Ces relations sont transformées en une *structure de coût*, une expression et des contraintes sur les variables qui décrivent le comportement global du programme. De cette structure est déduite une borne sur la fonction de coût. Les termes utilisés font apparaître des polynômes en les variables, des minimums et des maximums.

Décompte des points entiers d'un polytope. Verdoolaege et al. cherche à calculer dans [6] le nombre de points à coordonnées entières dans un polytope paramétrique (cf. section 3.1). Cet objectif est atteint par un algorithme en temps polynomial travaillant sur des quasi-polynômes, i.e. des polynômes dont les coefficients sont des fonctions périodiques de $\mathbb{Z}^n$ dans $\mathbb{Q}$. En effet, le résultat est lui-même un quasi-polynôme. L'algorithme de Barvinok consiste à calculer la série génératrice du polytope P, i.e. $\sum_{a \in \mathbb{Z}^n \cap P} X^a$ et à l'évaluer en 1. Le résultat est découpé en *chambres*, i.e. sous-espaces de l'espace des paramètres dans lesquels le résultat admet une forme close en fonction des paramètres.

6 Conclusion

Durant ce stage, mon travail a été constitué de deux parties. D'une part, j'ai cherché un moyen de produire une analyse aussi précise que possible avec l'aide ponctuelle de l'équipe puis formalisé mes résultats. D'autre part j'ai implémenté ces résultats théoriques dans AUBE (excepté le parseur de la sortie de *barvinok* écrit par M. Sotin).

Par manque de temps, ce travail n'a pu être testé que sur des exemples écrits à la main et le lien avec l'outil *Choregies* n'a pas été fait. Il serait intéressant de voir si la bibliothèque se comporte aussi bien sur de gros programmes analysés automatiquement que sur les petits exemples traités à la main. Sans doute peut-on optimiser le traitement des formules, par exemple en évitant de faire des transformations inutiles, comme l'élimination des modulos lorsqu'il n'y en a pas. En outre, comme le montre l'exemple introductif (cf. section 2), le point critique de l'analyse est l'élimination approchée des produits (cf. section 3.2).

Ce stage a constitué pour moi un premier pas dans le domaine de la recherche. Il s'est agi de comprendre les besoins de la partie de l'équipe qui travaillait sur l'analyse de cache et résoudre au mieux le problème posé. L'écriture du rapport avec les conseils de mes encadrants a été une expérience très instructive, autant sous l'aspect de la forme que de la présentation formelle de mes résultats. En un peu moins de deux mois, j'ai pu découvrir deux domaines de recherche, à savoir l'analyse de programmes et le décompte de points entiers dans un polyèdre, et ainsi améliorer ma culture scientifique. Mais j'ai surtout pu grandement valoriser l'expérience de l'informatique que j'acquiers depuis six ans, ainsi que deux cours de mathématiques en particulier de l'année passée. Le cours de logique m'a permis de trouver facilement un formalisme agréable et adapté au problème et, sans le cours d'algèbre, certains articles auraient été très difficiles à lire. Cela me conforte dans le choix de la double licence info-maths et dans l'idée de ne pas abandonner les mathématiques en master.

Pour finir, j'aimerais remercier l'équipe d'accueil et en particulier mes deux encadrants pour m'avoir suivi et épaulé tout au long du stage, ainsi que tous les stagiaires auprès desquels j'ai travaillé et qui m'ont réservé un accueil très chaleureux.

Références

[1] M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, and J. Giesl. Analyzing runtime and size complexity of integer programs. *ACM Trans. Program. Lang. Syst.*, 38(4) :13 :1–13 :50, 2016.

[2] Antonio Flores-Montoya. Upper and lower amortized cost bounds of programs expressed as cost relations. pages 254–273, 2016.

[3] Zachary Kincaid, Jason Breck, John Cyphert, and Thomas W. Reps. Closed forms for numerical loops. *PACMPL*, 3(POPL) :55 :1–55 :29, 2019.

[4] Zachary Kincaid, John Cyphert, Jason Breck, and Thomas W. Reps. Non-linear reasoning for invariant synthesis. *PACMPL*, 2(POPL) :54 :1–54 :33, 2018.

[5] Moritz Sinn, Florian Zuleger, and Helmut Veith. Complexity and resource bound analysis of imperative programs using difference constraints. *J. Autom. Reasoning*, 59(1) :3–45, 2017.

[6] Sven Verdoolaege, Rachid Seghir, Kristof Beyls, Vincent Loechner, and Maurice Bruynooghe. Counting integer points in parametric polytopes using barvinok's rational functions. *Algorithmica*, 48(1) :37–66, 2007.