

Analyse statique-Optimisation :

Quelques problèmes de flots de données

1 Introduction

L'*analyse statique* d'un programme consiste à déterminer des informations sur le comportement à l'exécution de ce programme (sans l'exécuter) et à utiliser ces informations soit pour *vérifier* que le programme satisfait une certaine propriété, soit pour *optimiser* ce programme. Les critères d'optimisation concernent la consommation des ressources (temps d'exécution, consommation mémoire et consommation d'énergie).

Nous présentons les analyses flot de données définies sur une représentation intermédiaire du programme indépendante du langage source et de la machine cible. Cette représentation est structurée en un graphe de flot de contrôle, dont les nœuds sont des blocs de base et les arcs expriment le flot de contrôle.

L'information est déterminée de la manière suivante :

- génération d'un système d'équations,
- résolution des équations,
- transformation de programme.

1.1 Définitions préliminaires

Un bloc de base B est une séquence d'affectations terminée ou non par un branchement. L'algorithme qui partitionne du code 3 adresses en blocs de base détermine les têtes de bloc (première instruction, instruction atteinte par une instruction de branchement ou toute instruction suivant un branchement).

Le graphe de flot de contrôle est un graphe dont l'ensemble des nœuds est constitué de l'ensemble des blocs de base, augmenté d'un bloc initial et d'un bloc final. Un arc relie deux nœuds si le contrôle peut passer d'un nœud à l'autre.

1.2 Génération d'un système d'équations

On veut calculer en chaque point d'exécution du programme des propriétés. Pour ce faire, on détermine pour chaque bloc de base une propriété locale.

A chaque bloc de base B on associe 2 variables $\text{In}(B)$ et $\text{Out}(B)$. Puis, on relie par des équations ces variables de la manière suivante :

- lorsque l'information se propage dans le sens du flot de contrôle, la variable $\text{In}(B)$ s'exprimera en fonction des variables $\text{Out}(B')$, où B' sont des prédécesseurs de B .
- lorsque l'information se propage dans le sens inverse du flot de contrôle, la variable $\text{Out}(B)$ s'exprimera en fonction des variables $\text{In}(B')$, où les B' sont des successeurs de B .
- des fonctions de transfert relient, pour chaque bloc de base les variables $\text{In}(B)$ et $\text{Out}(B)$.

1.3 Résolution d'un système d'équations

Pour chaque bloc de base B , est définie une fonction de transfert F_B . Les propriétés sont exprimées dans un treillis. Rappelons qu'un treillis complet (resp. treillis) $(E, \leq, \sqcup, \sqcap, \perp, \top)$ est un ensemble E muni d'une relation d'ordre \leq , d'un plus petit élément, noté \perp , et d'un plus grand élément, noté \top , tel que tout sous-ensemble (resp sous-ensemble fini) $X \subseteq E$ a une borne supérieure $\sqcup X$, et une borne inférieure $\sqcap X$, qui sont dans ce sous-ensemble. Une solution du système d'équations est un point fixe. On s'intéresse soit à un *plus petit point fixe*, noté PPPF, qui caractérise l'existence d'un point où la propriété est vérifiée, soit à un *plus grand point fixe*, noté PGPF, qui caractérise le fait que la propriété est vérifiée en chaque point.

Analyse en avant, PPPF	$\text{In}(B) = \begin{cases} \perp & \text{si } B \text{ est initial} \\ \sqcup_{B' \in \text{pre}(B)} \text{Out}(B') & \text{sinon} \end{cases}$ $\text{Out}(B) = F_B(\text{In}(B))$
Analyse en avant, PGPF	$\text{In}(B) = \begin{cases} \perp & \text{si } B \text{ est initial} \\ \sqcap_{B' \in \text{pre}(B)} \text{Out}(B') & \text{sinon.} \end{cases}$ $\text{Out}(B) = F_B(\text{In}(B))$
Analyse en arrière, PPPF	$\text{Out}(B) = \begin{cases} \perp & \text{si } B \text{ est final} \\ \sqcup_{B' \in \text{succ}(B')} \text{In}(B') & \text{sinon.} \end{cases}$ $\text{In}(B) = F_B(\text{Out}(B))$
Analyse en arrière, PGPF	$\text{Out}(B) = \begin{cases} \perp & \text{si } B \text{ est final} \\ \sqcap_{B' \in \text{succ}(B)} \text{In}(B') & \text{sinon.} \end{cases}$ $\text{In}(B) = F_B(\text{Out}(B))$

1.3.1 Résolution d'équations dans un treillis

Soit $(E, \leq, \sqcup, \sqcap, \perp, \top)$ un treillis complet. Etant donné un système d'équations

$$X_j = F_j(X_1, \dots, X_n) \text{ pour } j \in [1, n]$$

il existe une plus petite (resp. plus grande) solution si les F_j sont \sqcup (resp. \sqcap) continues. Dans ce cas, la plus petite solution est la limite de la suite :

$$\begin{aligned} X_j^0 &= (\perp, \dots, \perp) \\ X_j^{i+1} &= F_j(X_1^i, \dots, X_n^i) \end{aligned}$$

(resp. la plus grande solution est la limite de la suite :

$$\begin{aligned} X_j^0 &= (\top, \dots, \top) \\ X_j^{i+1} &= F_j(X_1^i, \dots, X_n^i). \end{aligned}$$

1.4 Transformation de programme

La transformation de programme consiste à utiliser l'information calculée précédemment pour modifier le programme. Ceci est fait de telle sorte que le programme initial et le programme transformé sont sémantiquement équivalents.

2 Expressions disponibles

Une expression $x+y$ est disponible en un point p du graphe de flot de contrôle si tous les chemins allant du bloc initial à p contiennent $x+y$ et les opérandes x et y ne sont pas modifiés après la dernière occurrence de l'expression.

2.1 Equations

Le treillis est l'ensemble des parties de l'ensemble des expressions apparaissant dans le programme, ordonné par inclusion, tel que l'opération borne supérieure est l'union, l'opération borne inférieure est l'intersection. Au niveau d'un bloc de base, la propriété de disponibilité s'exprime en utilisant les fonctions **Gen** et **Kill**. La fonction de transfert s'exprime en fonction de **Gen** et **Kill**.

$$\begin{aligned} \text{In}(B) &= \bigcap_{B' \in \text{pre}(B)} \text{Out}(B') \\ \text{Out}(B) &= (\text{In}(B) \setminus \text{Kill}(B)) \cup \text{Gen}(B) \end{aligned}$$

Les propriétés locales sont définies comme suit :

- **Kill**(B) est l'ensemble des expressions supprimées dans le bloc de base. Une expression $x+y$ apparaissant dans un bloc de base est supprimée si sa dernière occurrence d'apparition est suivie d'une définition d'un de ses deux opérandes. Une expression n'apparaissant pas dans un bloc de base est supprimée si l'un ou l'autre de ses opérandes est modifié.
- **Gen**(B) est l'ensemble des expressions générées dans le bloc de base. Une expression est générée si elle apparaît dans un bloc de base et si sa dernière occurrence d'apparition n'est suivie d'aucune définition de ses opérandes.

2.2 Calcul des propriétés locales

On va déterminer, pour chaque bloc de base, quelle est l'information créée et supprimée. Pour définir **Gen** et **Kill**, on introduit les fonctions locales **Gen_l** et **Kill_l** :

Gen (B)	=	Gen_l (B, \emptyset)
Kill (B)	=	Kill_l (B, \emptyset)
Gen_l ($x := a; B, X$)	=	Gen_l ($B, X \setminus \{e' \mid x \in \text{VarLib}(e')\} \cup \{a \mid x \notin \text{VarLib}(a)\}$)
Gen_l (if b goto l, X)	=	$X \cup \{b\}$
Gen_l (goto l, X)	=	X
Gen_l (ϵ, X)	=	X
Kill_l ($x := a; B, X$)	=	Kill_l ($B, X \cup \{e' \mid x \in \text{VarLib}(e')\} \cup \{a \mid x \in \text{VarLib}(a)\}$)
Kill_l (if b goto l, X)	=	X
Kill_l (goto l, X)	=	X
Kill_l (ϵ, X)	=	X

Autrement dit, si le bloc B est une séquence de n affectations $x_i := a_i$ (suivie ou non d'un branchement) :

\dots
 $1 : x_1 := a_1$
 \dots
 $i : x_i := a_i$
 \dots
 $n : x_n := a_n$

- pour la fonction **Gen**, on construit la suite $X_0 = \emptyset, X_1, \dots, X_{n+1}$:

$$\begin{aligned}
& X_0 = \emptyset \\
1 & : x_1 := a_1 \\
& X_1 = \begin{cases} \emptyset & \text{si } x_1 \in \text{VarLib}(a_1) \\ \{a_1\} & \text{sinon} \end{cases} \\
& X_i \\
i & : x_i := a_i \\
& X_{i+1} = \begin{cases} X_i \setminus \{e' \mid x_i \in \text{VarLib}(e')\} & \text{si } x_i \in \text{VarLib}(a_i) \\ X_i \setminus \{e' \mid x_i \in \text{VarLib}(e')\} \cup \{a_i\} & \text{sinon} \end{cases} \\
n & : x_n := a_n
\end{aligned}$$

Gen est alors X_{n+1} si cette suite d'affectations n'est pas suivie d'un branchement conditionnel ou $X_{n+1} \cup \{b\}$ si elle est suivie d'un branchement portant sur la condition b .

- $\text{Kill} = \bigcup_{i=1}^n \{e \mid x_i \in \text{VarLib}(e)\}$

2.3 Résolution des équations

On veut déterminer, pour chaque bloc de base, la valeur de $\text{In}(B)$ et de $\text{Out}(B)$. Pour cela,

- on détermine les propriétés locales **Gen**(B) et **Kill**(B) pour chaque bloc de base B ,
- on initialise $\text{In}(B)$ à l'ensemble de toutes les expressions pour chaque bloc B , différent du bloc initial ; pour le bloc initial B_0 , on initialise à l'ensemble vide,
- on itère les calculs suivants jusqu'à stabilisation :
 1. calculer $\text{Out}(B)$,
 2. calculer $\text{In}(B)$.

On peut aussi résoudre le système d'équations suivant:

$$\text{In}(B) = \bigcap_{B' \in \text{pre}(B)} \{(\text{In}(B') \setminus \text{Kill}(B')) \cup \text{Gen}(B')\}$$

ou celui-ci :

$$\text{Out}(B) = \left(\bigcap_{B' \in \text{pre}(B)} \text{Out}(B') \setminus \text{Kill}(B) \right) \cup \text{Gen}(B)$$

2.4 Application

Suppression des calculs redondants. Pour chaque expression disponible e à l'entrée d'un bloc de base, si cette expression existe dans ce bloc de base et si les opérandes ne sont pas modifiés entre le début du bloc et la position de l'expression, alors celle-ci est redondante. Pour la supprimer, on crée une nouvelle variable u , et par une recherche en arrière, on localise les instructions de la forme $x := e$, que l'on remplace par

$\begin{cases} u := e \\ x := u \end{cases}$ et on remplace par u le calcul de l'expression, dans le bloc où ce calcul est redondant.

3 Variables actives

Le treillis est l'ensemble des parties de l'ensemble des variables apparaissant dans le programme, ordonné par inclusion, tel que l'opération borne supérieure est l'union, l'opération borne inférieure est l'intersection.

- Une variable x est *inactive* en un point i du programme si tout chemin allant de i à un point j ne contient aucune utilisation de cette variable ; j étant soit le bloc de sortie soit un bloc contenant une affectation à x , qui ne soit pas précédée d'une utilisation de x .
- Une instruction $d:x := e$ est *inutile* si x est inactive en fin de bloc et n'est pas utilisée dans le bloc après l'instruction.

Une méthode simple pour déterminer si une variable est inactive est de réaliser une analyse des variable actives, dont nous donnons les équations :

$$\begin{aligned} \text{Out}(B) &= \bigcup_{B' \in \text{succ}(B)} \text{In}(B') \\ \text{In}(B) &= (\text{Out}(B) \setminus \text{Kill}(B)) \cup \text{Gen}(B) \end{aligned}$$

$\text{Gen}(B)$ est l'ensemble des variables x telles que : x est utilisée dans le bloc B , et dans ce bloc B , toute éventuelle affectation à x est placée après la première utilisation de x .

$\text{Kill}(B)$ est l'ensemble des variables x affectées dans le bloc B .

3.1 Calcul des propriétés locales

Comme pour les expressions disponibles, on va déterminer les fonctions locales Gen_l et Kill_l :

$\text{Gen}(B)$	$= \text{Gen}_l(B, \emptyset)$
$\text{Kill}(B)$	$= \text{Kill}_l(B, \emptyset)$
$\text{Gen}_l(B; x := a, X)$	$= \text{Gen}_l(B, X \setminus \{x\} \cup \text{VarLib}(a))$
$\text{Gen}_l(B; \text{if } b \text{ goto } l, X)$	$= \text{Gen}_l(B, X \cup \text{VarLib}(b))$
$\text{Gen}_l(b; \text{goto } l, X)$	$= \text{Gen}_l(B, X)$
$\text{Gen}_l(\epsilon, X)$	$= X$
$\text{Kill}_l(B; x := a, X)$	$= \text{Kill}_l(B, X \cup \{x\})$
$\text{Kill}_l(B; \text{if } b \text{ goto } l, X)$	$= \text{Kill}_l(B, X)$
$\text{Kill}_l(b; \text{goto } l, X)$	$= \text{Kill}_l(B, X)$
$\text{Kill}_l(\epsilon, X)$	$= X$

Comme précédemment, si le bloc B est une suite de n affectations, suivie ou non d'un branchement,

- pour la fonction Gen , on construit la suite $X_0 = \text{Gen}, X_1, X_{n+1}$, avec

$$\begin{aligned} X_{n+1} &= \begin{cases} \emptyset & \text{si la séquence n'est pas suivie d'un branchement conditionnel} \\ \{b\} & \text{si la séquence est suivie d'un branchement conditionnel portant sur } b \end{cases} \\ X_i &= (X_{i+1} \setminus \{x_i\}) \cup \text{VarLib}(a_i) \end{aligned}$$

- $\text{Kill}(B) = \{x_1, \dots, x_n\}$

3.2 Résolution des équations

Comme pour les expressions disponible, on veut déterminer, pour chaque bloc de base, la valeur de $\text{In}(B)$ et de $\text{Out}(B)$. Pour cela,

- on détermine les propriétés locales $\text{Gen}(B)$ et $\text{Kill}(B)$ pour chaque bloc de base B ,
- on initialise $\text{Out}(B)$ à l'ensemble vide \emptyset ,
- on itère les calculs suivants jusqu'à stabilisation :
 1. calculer $\text{In}(B)$.
 2. calculer $\text{Out}(B)$,

3.3 Application

Suppression des instructions inutiles. Les instructions inutiles sont les instructions d'affectation dont la partie gauche est inactive. On propose donc la transformation suivante :

- calcul des variables actives pour chaque bloc de base,
- pour chaque bloc de base B , on effectue la transformation locale suivante :

On initialise $X = \text{Out}(B)$	$F(B; \text{if } b \text{ goto } l, X)$	$= F(B, X \cup \text{VarLib}(b)); \text{if } b \text{ goto } l$
	$F(B; \text{goto } l, X)$	$= F(B, X); \text{goto } l$
	$F(B; x := a, X)$	$= \begin{cases} F(B, X) & \text{si } x \notin X \\ F(B, X \cup \text{VarLib}(a)); x := a & \text{si } x \in X \end{cases}$
	$F(\epsilon, X)$	$= \epsilon$

4 Propagation des constantes

Chaque variable prend ses valeurs dans un domaine, étendu par deux valeurs : \top et \perp . La valeur \top indique une information non-constante, alors que la valeur \perp indique l'absence d'information. On veut définir un treillis sur le domaine étendu $D \cup \{\top, \perp\}$. La relation d'ordre est définie comme suit : si $v \in D$ alors $\perp < v$ et $v < \top$.

On définit la borne supérieure \sqcup : Pour $x \in D \cup \{\top, \perp\}$ et $v_1, v_2 \in D$

$x \sqcup \top = \top$	$x \sqcup \perp = x$	$v_1 \sqcup v_2 = \top$ si $v_1 \neq v_2$	$v_1 \sqcup v_1 = v_1$
------------------------	----------------------	---	------------------------

Soit \mathcal{V} , l'ensemble des variables apparaissant dans un programme. On suppose, pour simplifier, qu'elles prennent toutes leur valeur dans le même domaine D . On considère le treillis $[\mathcal{V} \mapsto D \cup \{\top, \perp\}]$ dont l'ordre est induit par celui de $D \cup \{\top, \perp\}$: $f \leq g$ ssi $\forall x \cdot f(x) \leq g(x)$. La borne supérieure est définie comme suit : $(f \sqcup g)(x) = f(x) \sqcup g(x)$. Le plus petit élément est la fonction nulle part définie, notée f_\perp . Par ailleurs, les fonctions de $[\mathcal{V} \mapsto D \cup \{\top, \perp\}]$ sont étendues de manière naturelle aux expressions du programme utilisant les variables de \mathcal{V} .

4.0.1 Equations

$$\begin{aligned} \text{In}(B) &= \begin{cases} f_\perp & \text{si } B \text{ est initial,} \\ \bigsqcup_{B' \in \text{pre}(B)} \text{Out}(B') & \text{sinon} \end{cases} \\ \text{Out}(B) &= F_b(\text{In}(B)) \end{aligned}$$

La fonction de transfert est définie comme suit, par induction sur la séquence des affectations d'un bloc de base.

$$\begin{aligned} F_{\mathbf{x}:=\mathbf{a}} ; \mathbf{B}(f) &= F_{\mathbf{B}}(f[x \mapsto f(a)]) \\ F_{\epsilon}(f) &= f \end{aligned}$$

Remarque : Pour que cette analyse ait du sens, il faut supposer que les variables sont correctement initialisées.

4.1 Transformation de programmes

Dans chaque bloc de base B , soit $f \in \text{In}(B)$, on remplace $\mathbf{x} := \mathbf{a}$ par $\mathbf{x} := v$ si $v = f(a) \neq \top$

4.2 Détection des constantes sur le langage while

On modifie d'abord les règles de sémantiques, en introduisant une nouvelle relation de transition, notée $\longrightarrow_{\#}$ qui sera une abstraction de la sémantique standard. On intègre la modification de programmes et on modifie les configurations en conséquence. On modifie le domaine des expressions booléennes en rajoutant la valeur \top . Pour les commandes, on a les règles suivantes :

$$\begin{array}{c} (x := a, \sigma) \longrightarrow_{\#} \sigma[x \mapsto \mathcal{A}[a]\sigma] \quad (\text{skip}, \sigma) \longrightarrow_{\#} \sigma \\ \frac{(S_1, \sigma) \longrightarrow_{\#} \sigma_1 \quad (S_2, \sigma_1) \longrightarrow_{\#} \sigma_2}{(S_1; S_2, \sigma) \longrightarrow_{\#} \sigma_2} \end{array}$$

4.2.1 Cas du If

$$\begin{array}{c} \frac{(S_1, \sigma) \longrightarrow_{\#} \sigma_1}{(\text{if } b \text{ then } S_1 \text{ else } S_2, \sigma) \longrightarrow_{\#} \sigma_1} \mathcal{B}[b]\sigma = \text{tt} \quad \frac{(S_2, \sigma) \longrightarrow_{\#} \sigma_2}{(\text{if } b \text{ then } S_1 \text{ else } S_2, \sigma) \longrightarrow_{\#} \sigma_2} \mathcal{B}[b]\sigma = \text{ff} \\ \frac{(S_1, \sigma) \longrightarrow_{\#} \sigma_1 \quad (S_2, \sigma) \longrightarrow_{\#} \sigma_2}{(\text{if } b \text{ then } S_1 \text{ else } S_2, \sigma) \longrightarrow_{\#} \sigma_1 \sqcup \sigma_2} \mathcal{B}[b]\sigma = \top \end{array}$$

Ici, $(\sigma_1 \sqcup \sigma_2)(x) = \sigma_1(x) \sqcup \sigma_2(x)$

4.2.2 Cas du While

$$\begin{array}{c} (\text{while } b \text{ do } S \text{ od}, \sigma) \longrightarrow_{\#} \sigma \quad (\mathcal{B}[b]\sigma = \text{ff}) \\ \frac{(S, \sigma) \longrightarrow_{\#} \sigma_1}{(\text{while } b \text{ do } S \text{ od}, \sigma) \longrightarrow_{\#} \sigma_1} (\sigma = \sigma_1) \\ \frac{(S, \sigma) \longrightarrow_{\#} \sigma_1 \quad (\text{while } b \text{ do } S \text{ od}, \sigma \sqcup \sigma_1) \longrightarrow_{\#} \sigma_2}{(\text{while } b \text{ do } S \text{ od}, \sigma) \longrightarrow_{\#} \sigma_2} (\mathcal{B}[b]\sigma \neq \text{ff} \text{ et } \sigma \neq \sigma_1) \end{array}$$

4.3 Transformation de programmes

On supprime les calculs inutiles. Pour cela, on définit la règle de transformation de programme

$$\sigma \vdash S \triangleright S'$$

qui signifie, la commande S a le même effet que la commande S' dans la mémoire σ . Lorsque S' est une séquence de **skip**, on remplacera S' par **skip**. Cette dernière transformation n'est pas présentée ci-dessous.

$$\sigma \vdash x := a \triangleright x := v \quad \text{si } \mathcal{A}[a]\sigma = v \neq \top$$

$$\sigma \vdash x := a \triangleright x := a \quad \text{si } \mathcal{A}[a]\sigma = \top$$

$$\sigma \vdash \text{skip} \triangleright \text{skip}$$

$$\frac{\sigma \vdash S_1 \triangleright S'_1 \quad (S_1, \sigma) \longrightarrow_{\#} \sigma_1 \quad \sigma_1 \vdash S_2 \triangleright S'_2}{\sigma \vdash S_1; S_2 \triangleright S'_1; S'_2}$$

4.3.1 Cas du If

$$\frac{\sigma \vdash S_1 \triangleright S'_1}{\sigma \vdash \text{if } b \text{ then } S_1 \text{ else } S_2 \triangleright S'_1} \mathcal{B}[b]\sigma = \text{tt}$$

$$\frac{\sigma \vdash S_2 \triangleright S'_2}{\sigma \vdash \text{if } b \text{ then } S_1 \text{ else } S_2 \triangleright S'_2} \mathcal{B}[b]\sigma = \text{ff}$$

$$\frac{\sigma \vdash S_1 \triangleright S'_1 \quad \sigma \vdash S_2 \triangleright S'_2}{\sigma \vdash \text{if } b \text{ then } S_1 \text{ else } S_2 \triangleright \text{if } b \text{ then } S'_1 \text{ else } S'_2} \mathcal{B}[b]\sigma = \top$$

4.3.2 Cas du while

$$\sigma \vdash \text{while } b \text{ do } S \text{ od} \triangleright \text{skip} \mathcal{B}[b]\sigma = \text{ff}$$

$$\frac{\sigma \vdash S \triangleright S'}{\sigma \vdash \text{while } b \text{ do } S \text{ od} \triangleright \sigma \vdash \text{while } b \text{ do } S' \text{ od}} \text{ sinon}$$

Propriétés

- Si $(S, \sigma) \longrightarrow \sigma'$ et $\sigma \vdash S \triangleright S'$ alors $(S', \sigma) \longrightarrow \sigma'$
- Si $(S, \sigma) \longrightarrow \sigma_1$ et $(S, \sigma) \longrightarrow_{\#} \sigma_2$ alors $\sigma_1 \leq \sigma_2$