

Examen Compilation

Université Paris Diderot, Master Ingénierie Informatique (M1).

Première session 2012-2013. Durée 2h. L'utilisation de notes de cours est autorisée, l'utilisation de tout autre document ou dispositif électronique est interdite.

On rappelle la sémantique de la machine **Vm** étudiée dans le cours.

Règle	$C[i] =$
$C \vdash (i, \sigma, s) \vdash (i+1, n \mid \sigma, s)$	<code>cnst(n)</code>
$C \vdash (i, \sigma, s) \vdash (i+1, s(x) \mid \sigma, s)$	<code>var(x)</code>
$C \vdash (i, n \mid \sigma, s) \vdash (i+1, \sigma, s[n/x])$	<code>setvar(x)</code>
$C \vdash (i, n \mid n' \mid \sigma, s) \vdash (i+1, (n +_{\mathbb{Z}} n') \mid \sigma, s)$	<code>add</code>
$C \vdash (i, \sigma, s) \vdash (i+k+1, \sigma, s)$	<code>branch(k)</code>
$C \vdash (i, n \mid n' \mid \sigma, s) \vdash (i+1, \sigma, s)$	<code>bge(k)</code> et $n <_{\mathbb{Z}} n'$
$C \vdash (i, n \mid n' \mid \sigma, s) \vdash (i+k+1, \sigma, s)$	<code>bge(k)</code> et $n \geq_{\mathbb{Z}} n'$

Aux instructions ci-dessus il faut ajouter l'instruction **halt** qui arrête le calcul. L'objectif de cet exercice est de concevoir une *analyse statique* des programmes de la **Vm** qui pour chaque instruction cherche à prédire la *taille de la pile* (le nombre d'entiers sur la pile). On fait l'hypothèse que dans un code **Vm** les instructions sont numérotées de 0 à $N-1$, et que tous les sauts (explicites ou implicites) sont dans l'intervalle $I = [0, N]$.

Question 1 Donner un exemple de code **Vm** qui contient : (i) au plus 5 instructions, (ii) au moins une instruction qui n'est pas accessible. (iii) au moins une instruction qui est exécutée au moins une fois avec une pile de taille n et au moins une autre fois avec une pile de taille n' avec $n \neq n'$.

En **ocaml** on représente le code **Vm** avec les types suivants :

```
type instruction = Cnst of int | Var of string | Setvar of string |
                  Add | Branch of int | Bge of int | Halt ;;
```

```
type code = instruction list;;
```

Question 2 Dans cette question comme dans la suivante on peut utiliser des fonctions de la bibliothèque **ocaml** sur les listes. Dans ce cas, dire brièvement ce que la fonction est censée faire. Écrire une fonction **ocaml**

```
val dj_cence : code -> (int * int) list list = <fun>
```

qui appliquée à un code c produit une liste dont le i -ème élément est une liste de couples d'entiers ℓ avec la propriété suivante :

le couple (j, ℓ) paraît dans la liste ℓ si et seulement si la i -ème instruction du code peut sauter à l'instruction j en faisant varier la taille de la pile de ℓ .

Par exemple, si la i -ème instruction est **Bge(k)** et $k \neq 0$ alors la liste ℓ associée pourrait être $[(i+1, -2); (i+1+k, -2)]$.

Question 3 Écrire une fonction **ocaml**

```
val predecesseur: code -> int -> (int * int) list = <fun>
```

qui appliquée à un code c et à un entier $j \in I$ produit une liste de couples d'entiers ℓ qui satisfait la propriété suivante :

le couple (i, j) paraît dans la liste ℓ si et seulement si la i -ème instruction du code peut sauter à l'instruction j en faisant varier la taille de la pile de $j - i$.

On écrit aussi :

$$\begin{aligned} i \in \text{pred}(j) & \iff \text{ssi } (i, j) \text{ st dans la list } (\text{predecesseur } c \ j) \\ \text{diff}(i, j) &= \text{ssi } (i, j) \text{ st dans la list } (\text{predecesseur } c \ j) \end{aligned}$$

On considère un treillis D qui contient les éléments suivants : (i) l'élément nul \perp , (ii) les éléments singuliers fng pour $n \in \mathbb{N}$, (iii) l'élément nul \mathbf{N} et les nombres naturels $f0, 1, 2, \dots, g$. Les éléments du treillis D sont ordonnés par la relation d'inclusion. On dénote par $_$ l'opération de sup. Notons que $fng _ fmg = \mathbf{N}$ si $n \neq m$. Ensuite on construit le treillis $[I \rightarrow D]$ des fonctions de I dans D ordonnées par points, c'est à dire si $f, g \in [I \rightarrow D]$ alors $f \leq g$ ssi $\forall i \in I \ f(i) \leq g(i)$. On définit les fonctions $iszero : I \rightarrow D$ et $update : D \times \mathbb{Z} \rightarrow D$ par :

$$iszero(i) = \begin{cases} f0g & \text{si } i = 0 \\ \perp & \text{autrement.} \end{cases} \quad update(d, n) = \begin{cases} \mathbf{N} & \text{si } d = \mathbf{N} \\ fmg + n & \text{si } d = fmg, m + n \leq g \\ \perp & \text{autrement.} \end{cases}$$

Ensuite, on définit une fonction $Inv : [I \rightarrow D] \rightarrow [I \rightarrow D]$ par :

$$Inv(f)(j) = \left(\bigvee_{i \in \text{pred}(j)} up(f(i), \text{diff}(i, j)) \right) _ iszero(j).$$

Question 4 Montrer que la fonction Inv est monotone sur le treillis $[I \rightarrow D]$.

Question 5 Décrire une méthode algorithmique pour calculer le plus petit point fixe de Inv qu'on dénote par SS (stack size). Il faut notamment argumenter pour montrer la terminaison de la méthode.

Question 6 Appliquer la méthode de la question 5 au programme :

```
0: cnst(0)
1: cnst(1)
2: bge(-3)
3: halt
```

Expliciter le calcul et le résultat final pour chaque instruction.

Question 7 Montrer que si $C \vdash (0, \epsilon, s) \xrightarrow{*} (i, \sigma, s')$ alors $j \in SS(i)$. Dans l'analyse des instructions on pourra se limiter à examiner un ou deux cas.

Question 8 Dériver comme corollaire que si $SS(i) = \emptyset$ alors l'instruction i n'est jamais exécutée (elle n'est pas accessible).

Question 9 Donner un exemple de programme avec une instruction i telle que $SS(i) = \emptyset$ et avec une instruction j tel que $SS(j) \neq \emptyset$ mais l'instruction j n'est pas accessible à partir d'une configuration initiale $(0, \epsilon, s)$.

Question 10 On fait référence à la fonction de compilation du langage **Imp** vers la **Vm** définie dans le cours. Peut-on prédire la taille de la pile de la i -ème instruction d'un code de la forme $C(\text{prog } S)$ en fonction du type de l'instruction (7 cas) ? Que peut-on dire sur la valeur de $SS(i)$? On ne vous demande pas de faire des preuves mais de formuler une conjecture confortée par une brève argumentation informelle et/ou par des petits exemples.

Solution

Réponse question 1 [1 point]

```
0: cnst(0)
1: br nch(-2)
2: h lt
```

L'instruction 1 est exécuté avec un pil d taill 1,2,3,... alors qu l'instruction 2 n'est jamais exécuté .

Réponse question 2 [2 points]

```
let next i inst = m tch inst with
  Cnst(n) -> [(i+1,1)]
| V r(x)  -> [(i+1,1)]
| Setv r(x) -> [(i+1,-1)]
| Add      -> [(i+1,-1)]
| Br nch(k) -> [(i+1,0)]
| Bge(k)    -> if k=0 then [(i+1,-2)] else [(i+1,-2);(i+1+k,-2)]
| H lt      -> [] ;;

let rec dj c i = m tch c with
  [] -> []
| inst::c1 -> (next i inst)::(dj c1 (i+1)) ;;

let dj cence c = dj c 0;;
```

Réponse question 3 [2 points]

```
let rec conc t l1 l2 = m tch l1 with
  [] -> l2
| x::l3 -> x::(conc t l3 l2);;

let rec genlist l j i = m tch l with
  [] -> []
| (k,delt)::l1 -> if (k=j) then (i,delt)::(genlist l1 j i)
                  else (genlist l1 j i) ;;

let rec pre djlist j i = m tch djlist with
  [] -> []
| l::djlist1 -> conc t (genlist l j i) (pre djlist1 j (i+1)) ;;

let predecesseur c j =
let djlist = dj cence c in
pre djlist j 0;;
```

Réponse question 4 [2 points]

On commence par observer que la fonction *update* est monotone sur D :

$$d \leq_D d' \text{ implique } \text{update}(d, _) \leq \text{update}(d', _)$$

Ensuite on suppose $f \leq_{[I \rightarrow D]} g$ et on montre que pour tout j :

$$\begin{aligned} & \text{Inv}(f)(j) \\ &= (\bigvee_{i \in \text{pred}(j)} \text{up}(f(i), \text{diff}(i, j))) _ \text{isero}(j) \\ & \quad (\bigvee_{i \in \text{pred}(j)} \text{up}(g(i), \text{diff}(i, j))) _ \text{isero}(j) \\ &= \text{Inv}(g)(j) \end{aligned}$$

Réponse question 5 [4 points]

On sait que *Inv* a un plus petit point fixe SS . On sait aussi que la séquence de fonctions dans $[I \rightarrow D]$:

$$\text{SS}_0 = \lambda i \geq I.; \quad \text{SS}_{n+1} = \text{Inv}(\text{SS}_n)$$

a la propriété que :

$$\text{SS}_i \leq_{[I \rightarrow D]} \text{SS}_{i+1} \leq_{[I \rightarrow D]} \text{SS}$$

Chaque fonction f dans $[I \rightarrow D]$ peut être représentée par un tableau fini à partir duquel on peut calculer le tableau qui représente $\text{Inv}(f)$. Pour montrer qu'on peut calculer SS il reste à observer que chaque séquence strictement croissante dans $[I \rightarrow D]$ doit être finie. En effet la séquence la plus longue est bornée par $3 \cdot (N + 1)$: la taille de la pile de chaque instruction peut passer de 0 à un singleton et ensuite à \mathbf{N} . Il existe donc un k tel que

$$\text{SS}_k = \text{SS}_{k+1} = \text{SS}.$$

Le calcul s'arrête au premier k qui satisfait cette propriété.

Réponse question 6 [2 points]

Le calcul est résumé par le tableau suivant :

	SS_0	SS_1	SS_2	SS_3	$\text{SS}_4 = \text{SS}_5$
0	;	$f0g$	$f0g$	$f0g$	$f0g$
1	;	;	$f1g$	$f1g$	$f1g$
2	;	;	;	$f2g$	$f2g$
3	;	;	;	;	$f0g$

Réponse question 7 [4 points]

Par récurrence sur la longueur de la réduction.

– Comme cas de base on a $(0, \epsilon, s)$. On remarque :

$$0 = j\epsilon j \geq \text{SS}(0) \quad f0g$$

- Pour l pas d récurrence , on suppose :

$$C \setminus (0, \epsilon, s) \not\models^* (i, \sigma, s) \not\models (j, \sigma', s')$$

Par hyp. réc. on sait qu $j\sigma j \geq SS(i)$. Donc soit $SS(i) = fj\sigma jg$ soit $SS(i) = \mathbf{N}$. On procède par analyse de la i -ème instruction du code C . Par exemple, supposons $C[i] = \text{bge}(k)$ et $j = i + 1 + k$. Alors $i \geq \text{pred}(j)$, $\text{diff}(i, j) = -2$, $j\sigma j - 2$ et $j\sigma'j = j\sigma j - 2$. Par définition de point fixe on a :

$$SS(j) = \left(\bigvee_{i \in \text{pred}(j)} up(SS(i), \text{diff}(i, j)) \right) _ isero(j)$$

Si $SS(i) = fj\sigma jg$ alors $up(SS(i), \text{diff}(i, j)) = fj\sigma'jg$. Si $SS(i) = \mathbf{N}$ alors $up(SS(i), \text{diff}(i, j)) = \mathbf{N}$. Dans les deux cas, $j\sigma'j \geq SS(j)$.

Réponse question 8 [1 point]

Si l'instruction i est exécutée alors $(0, \epsilon, s) \not\models^* (i, \sigma, s')$ et donc $j\sigma j \geq SS(i) \notin ;$.

Réponse question 9 [2 points]

On reprend le programme de la question 6 et on y ajoute une instruction :

```
0: cnst(0)
1: cnst(1)
2: bge(-3)
3: halt
4: halt
```

Dans ce cas, on a $SS(4) = ;$ et $SS(3) \notin ;$ alors que l'instruction 4 est inaccessibles.

Réponse question 10 [2 points]

- Si e est une expression alors le code $\mathcal{C}(e)$ augmente la taille de la pile de 1. La taille de la pile en fonction de l'instruction exécutée est la suivante :

Instruction	Taille pile
var	0
cnst	0
add	2
setvar	1
branch	0
bge	2
halt	0

Pour les instructions `var`, `cnst` et `add` la valeur exacte de la taille de la pile est aussi fonction de l'expression dont ils sont la compilation.

- Au niveau du graph de flot de contrôle, toutes les instructions sont accessibles et l'analyse statique calcul pour chaque instruction une valeur singleton.