

Examen Compilation

Université Paris Diderot, Master Ingénierie Informatique (M1).

Première session 2010-2011. Durée 2h. L'utilisation de notes de cours est autorisée, l'utilisation de tout autre document ou dispositif électronique est interdite. Il est possible de faire l'impasse sur une question et passer à la suivante.

Dans cet exercice on considère des variantes du langage impératif **Imp** et de la machine virtuelle **Vm** discutés dans le cours.

1. On étend le langage des commandes S du langage impératif **Imp** comme suit :

$$S ::= \text{skip} \mid id := e \mid S; S \mid \text{if } b \text{ then } S \text{ else } S \mid \text{while } b \text{ do } S \mid \text{loop}\{S\} \mid \text{block}\{S\} \mid \text{exit } n .$$

On appelle cette extension Imp^+ . Donnez la sémantique à petits pas de Imp^+ en omettant les règles introduites dans le cours pour **Imp**. Les jugements auront la forme $(S, K, s) \rightarrow (S', K', s')$ et les continuations seront définies par : $K ::= \text{halt} \mid \text{endblock}(K) \mid S \cdot K$.

2. Appliquez les règles de réduction à la configuration (S, halt, s) où S est la commande suivante :

$$S = \text{skip}; \text{block}\{\text{block}\{\text{if } 0 < 1 \text{ then exit } 0 \text{ else exit } 1\}; \text{exit } 0\} .$$

3. On étend les instructions de la machine virtuelle **Vm** comme suit :
 - Toute instruction peut être étiquetée. Par exemple on écrira : $\ell : \text{cnst } n$ pour dire que l'instruction **cnst** n est étiquetée par ℓ .
 - On ajoute une instruction **jmp** ℓ de saut non-conditionné à une étiquette avec sémantique (s'il y a plus d'une instruction avec la même étiquette la sémantique est non-déterministe) :

$$C \vdash (i, \sigma, s) \rightarrow (i', \sigma, s) \quad \text{si } C[i] = \text{jmp } \ell \text{ et l'instruction } C[i'] \text{ est étiquetée par } \ell$$

On appelle cette extension Vm^+ . Dans le cours nous avons analysé une compilation de **Imp** vers **Vm** et une autre de **Cminor** vers **RTLabs**. Adaptez les définitions de façon à compiler Imp^+ vers Vm^+ .

Suggestion : La fonction de compilation des commandes prend comme arguments : une commande S , une étiquette ℓ qui correspond à la continuation standard et une liste d'étiquettes ℓ_0, \dots, ℓ_n qui correspondent aux continuations exceptionnelles. Par ailleurs, la fonction rend comme résultat une liste d'instructions du langage Vm^+ et une étiquette qui correspond à la première instruction de la liste. Il peut être utile de compiler **skip** vers une instruction '**nop**' qui ne fait rien.

4. Calculez la compilation de la commande :

$$S = \text{block}\{\text{block}\{\text{loop}\{\text{if } 0 < 1 \text{ then exit } 0 \text{ else exit } 1\}\}; \text{exit } 0\}; \text{skip} .$$

5. On considère maintenant un raffinement de la machine **Vm** (sans étiquettes!) dans laquelle les instructions de branchement **branch**(k) et **bge**(k) prennent deux adresses consécutives si le offset k est plus grand en valeur absolue qu'une certaine valeur $b > 0$. On appelle cela une *expansion* d'une instruction de branchement. Notez que l'expansion d'une instruction de branchement peut en causer d'autres en cascade. Le but est de concevoir et analyser un algorithme qui étant donné un programme C avec n instructions $C[0], \dots, C[n]$: (i) détermine quelles instructions de branchement doivent être expansées et (ii) recalcule les offsets des instructions de branchement. Soit $x_i \in \{1, 2\}$, une variable qui dénote le nombre d'adresses associées à l'instruction $C[i]$, $i \in \{1, \dots, n\}$. Soit of_i l'offset associé à l'instruction i qui est défini comme suit :

$$of_i = \begin{cases} k & \text{si } C[i] = (\text{branch } k) \text{ ou } C[i] = (\text{bge } k) \\ 0 & \text{autrement} \end{cases}$$

Décrivez les *contraintes* (inégalités) que les variables x_i doivent satisfaire en fonction de of_i .

6. Une *solution* au système de contraintes est un vecteur $(x_1, \dots, x_n) \in \{1, 2\}^n$ qui satisfait les contraintes. Une solution est *optimale* si la valeur $\sum_{i=1, \dots, n} x_i$ est la plus petite possible, c'est à dire si la longueur du code est minimum. Montrez que votre système de contraintes a toujours une solution (laquelle?) et qu'il est possible d'ordonner les vecteurs $\{1, 2\}^n$ de façon à que le calcul de la solution optimale se réduise au calcul d'un plus petit point fixe. En particulier, donnez : (i) la valeur initiale de l'itération, (ii) le pas d'itération, (iii) l'argument qui montre que l'itération termine et (iv) l'argument qui montre que quand l'itération termine on a atteint la solution optimale.
7. On suppose maintenant avoir construit un *graphe dirigé* G tel que :
- Les noeuds correspondent aux instructions de branchement.
 - Il y a une arête du noeud i au noeud j si une expansion du noeud j affecte le offset de l'instruction i .

Par ailleurs, on associe à chaque noeud i une valeur nof_i (*new offset*) qui à la terminaison de l'algorithme doit correspondre à la valeur de l'offset pour l'instruction i . On appelle *degré* du graphe le nombre d'arêtes qui peuvent pointer vers un noeud. Donnez une borne au degré de G en fonction de b .

8. Donnez la structure d'un algorithme qui calculerait la solution optimale en s'appuyant sur le graphe G et déterminez sa complexité asymptotique en temps.

Suggestion : on peut s'inspirer de l'algorithme de 'propagation des 1' pour la solution de systèmes d'équations booléennes monotones.

Solution

3 points La sémantique à petits pas :

$$\begin{array}{ll}(\text{block}\{S\}, K, s) & \rightarrow (S, \text{endblock}(K), s) \\(\text{loop}\{S\}, K, s) & \rightarrow (S, \text{loop}\{S\} \cdot K, s) \\(\text{exit } n, S \cdot K, s) & \rightarrow (\text{exit } n, K, s) \\(\text{exit } 0, \text{endblock}(K), s) & \rightarrow (\text{skip}, K, s) \\(\text{exit } n + 1, \text{endblock}(K), s) & \rightarrow (\text{exit } n, K, s) \\(\text{skip}, \text{endblock}(K), s) & \rightarrow (\text{skip}, K, s)\end{array}$$

2 points On utilise les abréviations suivantes :

$$\begin{array}{ll}S & = \text{skip}; B_1 \\B_1 & = \text{block}\{S_1\} \\S_1 & = B_2; \text{exit } 0 \\B_2 & = \text{block}\{S_2\} \\S_2 & = \text{if } 0 < 1 \text{ then exit } 0 \text{ else exit } 1 \\K_1 & = \text{endblock}(\text{halt}) \\K_2 & = \text{endblock}(\text{exit } 0 \cdot K_1)\end{array}$$

Par ailleurs on omet d'écrire la mémoire s qui ne joue aucun rôle dans le calcul.

$$\begin{array}{l} (S, \text{halt}) \\ \rightarrow (\text{skip}, B_1 \cdot \text{halt}) \\ \rightarrow (\text{block}\{S_1\}, \text{halt}) \\ \rightarrow (S_1, \text{endblock}(\text{halt})) \\ \rightarrow (\text{block}\{S_2\}, \text{exit } 0 \cdot K_1) \\ \rightarrow (S_2, K_2) \\ \rightarrow (\text{exit } 0, K_2) \\ \rightarrow (\text{skip}, \text{exit } 0 \cdot K_1) \\ \rightarrow (\text{exit } 0, K_1) \\ \rightarrow (\text{skip}, \text{halt}) \end{array}$$

4 points On prend `nop` comme une abréviation pour `branch 0`.

La compilation des expressions et des conditions booléennes rend maintenant aussi l'étiquette de la première instruction.

La compilation des commandes est la suivante.

$$\begin{aligned}
\mathcal{C}(\text{skip}, \ell, \vec{\ell}) &= \text{let } \ell' = \text{new in} \\
&\quad (\ell' : \text{nop}, \ell') \\
\mathcal{C}(x := e, \ell, \vec{\ell}) &= \text{let } (C_e, \ell') = \mathcal{C}(e) \\
&\quad \text{in } ((C_e) \cdot (\text{setvar}(x)), \ell') \\
\mathcal{C}(S_1; S_2, \ell, \vec{\ell}) &= \text{let } (C_2, \ell_2) = \mathcal{C}(S_2, \ell, \vec{\ell}), (C_1, \ell_1) = \mathcal{C}(S_1, \ell_2, \vec{\ell}) \\
&\quad \text{in } (C_1 \cdot C_2, \ell_1) \\
\mathcal{C}(\text{exit } i, \ell, \ell_0 \cdots \ell_n) &= \text{let } \ell' = \text{new} \\
&\quad \text{in } (\ell' : \text{jmp } \ell_i, \ell') \\
\mathcal{C}(\text{block}\{S\}, \ell, \vec{\ell}) &= \mathcal{C}(S, \ell, \ell \cdot \vec{\ell}) \\
\mathcal{C}(\text{loop}\{S\}, \ell, \vec{\ell}) &= \text{let } (C, \ell') = \mathcal{C}(S, \ell, \vec{\ell}) \\
&\quad \text{in } (C \cdot (\text{jmp } \ell'), \ell') \\
\mathcal{C}(\text{if } b \text{ then } S_1 \text{ else } S_2, \ell, \ell, \vec{\ell}) &= \text{let } \ell' = \text{new}, (C_1, \ell_1) = \mathcal{C}(S_1, \ell, \vec{\ell}), \\
&\quad (C_2, \ell_2) = \mathcal{C}(S_2, \ell, \vec{\ell}), (C_b, \ell_0) = \mathcal{C}(b, k) \\
&\quad \text{in } (C_b \cdot C_1 \cdot (\text{branch } k') \cdot C_2, \ell_0) \\
\text{où } k = |C_1| + 1, k' = |C_2| \\
\mathcal{C}(\text{while } b \text{ do } S, \ell, \vec{\ell}) &= \mathcal{C}(\text{block}\{\text{loop}\{\text{if } b \text{ then } S \text{ else exit } 0\}\}, \ell, \vec{\ell})
\end{aligned}$$

2 points La compilation de la commande S :

```

ℓ3 :  cnst(1)
      cnst(0)
      bge(1)
      jmp(ℓ2)
      jmp(ℓ1)
      jmp(ℓ3)
ℓ2 :  jmp(ℓ1)
ℓ1 :  nop

```

2 points On souhaite minimiser la valeur $\sum_{i=1,\dots,n} x_i$ en satisfaisant les contraintes suivantes :

$$x_i = \begin{cases} 1 & \text{si } of_i = 0 \\ 1 & \text{si } of_i > 0 \text{ et } x_{i+1} + \dots + x_{i+of_i} \leq b \\ 1 & \text{si } of_i < 0 \text{ et } x_{i+1+of_i} + \dots + x_i \leq b \\ 2 & \text{autrement} \end{cases}$$

3 points Ce problème admet toujours une solution qui consiste à prendre $x_i = 2$ pour toutes les instructions telle que $of_i \neq 0$. Pour ordonner les vecteurs, on pose :

$$(x_1, \dots, x_n) \leq (y_1, \dots, y_n)$$

si $x_i \leq y_i$ pour $i = 1, \dots, n$.

De plus on peut toujours trouver une solution optimale par la méthode itérative suivante :

- (i) On pose comme valeur initiale $x_i = 1$ pour $i = 1, \dots, n$.
- (ii) Tant qu'il y a une variable x_i telle que $x_i = 1$ et x_i ne satisfait pas les contraintes on affecte la valeur 2 à x_i et on itère.
- (iii) Cette méthode termine car chaque variable peut passer de 1 à 2 au plus une fois.
- (iv) De plus elle maintient l'invariant que la valeur (x_1, \dots, x_n) est une borne inférieure à la solution optimale.

1 point Si i est une instruction de branchement et of_i est le offset associé alors soit le offset est supérieur à b et dans ce cas on sait que $x_i = 2$ soit il est inférieur et dans ce cas il y a au plus b instructions qui peuvent affecter la valeur de x_i .

4 points On maintient un vecteur nof et x tel que nof_i est la valeur présumée de l'offset de l'instruction i . Au début $nof_i = of_i$ et $x_i = 1$.

Pour toutes les instructions i telle que $of_i > b$ on insère i dans un ensemble W

Tant que W n'est pas vide on itère les opérations suivantes :

- On extrait un élément i de W .
- On pose $x_i = 2$. Pour toute instruction j dont l'offset dépend de i on incrémente nof_j de 1 (ceci utilise le graphe G). De plus si $x_j = 1$ et $nof_j > b$ alors on insère j dans W .

Si m est le nombre d'instructions de branchement alors l'algorithme termine en $O(mb)$. Chaque instruction est insérée au plus une fois dans W et chaque fois qu'on extrait un élément de W on fait un travail borné par b .