IFIPS – Cycle Ingénieur 2e année Informatique
Compilation 2006–2007

Travaux Dirigés n^{o}7
Génération de code

1 Allocation de variables locales

On étudie un langage pour des expressions qui contiennent des définitions locales de variables. Ce langage est décrit par la grammaire suivante dans laquelle `cte` est un token correspondant à une valeur entière $(1, 2, \ldots)$, `ident` est un token correspondant à un identificateur qui peut représenter une variable locale ou bien une fonction prédéfinie et finalement `let` et `in` sont des mots clés et `=`, `(`, `,` et `)` sont des symboles. E est un non-terminal permettant de dériver les expressions du langage.

$$E ::= \texttt{cte} \mid \texttt{ident} \mid \texttt{let ident = } E \texttt{ in } E \mid \texttt{ident(}E\texttt{,}E\texttt{)} \mid \texttt{(}E\texttt{)}$$

Dans une expression `let` x `=` e_1 `in` e_2, x est une nouvelle variable qui peut être utilisée uniquement dans l'expression e_2. L'évaluation de l'expression complète consiste à évaluer e_1, à stocker la valeur obtenue dans une case mémoire m puis à évaluer e_2 en accédant à la case mémoire m à chaque utilisation de x.

On cherche à produire, pour chaque expression E, du code de la machine à pile introduite en cours dont l'exécution a pour effet de calculer la valeur de l'expression E au sommet de la pile. On suppose dans la suite que dans l'expression E, toutes les variables locales ont un nom différent. On choisit de représenter chaque variable locale à un emplacement fixé à partir de la base de la pile.

1. On suppose donnée une fonction `index` qui à chaque variable locale x de E, associe un entier n qui représente le décalage par rapport à la base de la pile à laquelle la variable x est stockée. On suppose que le nombre de cases mémoire différentes à utiliser pour stocker les variables locales de l'expression E est N. On suppose également que pour chaque fonction prédéfinie f, il y a un label `label`(f) auquel sont associées les instructions machines correspondant au calcul de f. Ces fonctions prennent en argument deux entiers et renvoient un entier, cet entier est stocké à la place du premier argument.

   Proposer une fonction `code` qui produit pour chaque expression E, les instructions de la machine à pile nécéssaires à l'évaluation de cette expression.

2. Soit l'expression

   ```
   let x = 1 in
   let t = let u = 2 in plus(u,3) in
   f(let y = h(2,t) in g(y,x), let z=4 in g(z,z))
   ```

On suppose que x est stocké au décalage 0, t et u en 1 et y et z en 2. Donner le code produit par votre programme pour cette expression.

3. On se propose maintenant de calculer une fonction `index` correcte. On adopte la stratégie suivante : si en un point de programme, les variables visibles sont $x_0, x_1, \ldots, x_n$ alors chaque x_i aura pour décalage i, c'est-à-dire que l'on aura $\texttt{index}(x_i) = i$.
   Proposer une fonction `analyse` qui étant donnée une expression E, attribue à chaque variable locale de E un décalage. On pourra supposer donnée une fonction `set_index` telle que (`set_index` n x) remplit une table utilisée par la fonction `index` en associant le décalage n à la variable x.

4. Proposer une fonction `maxvar` qui calcule pour chaque expression E le nombre de cases mémoire à réserver pour les variables locales en utilisant la stratégie précédente.
   Combien de cases mémoire doit-on utiliser pour l'expression de la question 2 ?

5. Proposer un décalage pour les variables de l'expression de la question 2 qui n'utilise que deux cases mémoire en supposant que dans l'expression $f(e_1, e_2)$, l'expression e_1 est évaluée avant e_2.

6. On se propose de mettre en œuvre une analyse plus fine pour le calcul de `index` qui prenne en compte l'utilisation effective des variables. Pour réaliser cette analyse, on transforme l'expression en une suite d'affectations élémentaires de la forme $x := t$ avec x une variable, et t un terme simple qui peut être une constante c, une variable y ou bien $f(y, z)$ l'application d'une fonction f à deux variables y et z.

   Pour compiler une expression e on se donne une variable r dans laquelle on souhaite stocker la valeur de e. Si e est une constante ou une variable, on utilise l'affectation correspondante. Si e est de la forme `let` $x = e_1$ `in` e_2 on engendre les affectations élémentaires pour calculer la valeur de l'expression e_1 dans la variable x puis les affectations pour calculer e_2 dans la variable r. Si e est de la forme $f(e_1, e_2)$, on crée de nouvelles variables r_1 et r_2, on engendre les affectations pour calculer e_1 dans r_1 puis e_2 dans r_2 et on ajoute l'affectation $r := f(r_1, r_2)$, si e_1 (resp. e_2) est déjà une variable, on peut l'utiliser directement.

   (a) Donner la suite d'affectations correspondant à l'expression de la question 2.

   (b) Fournir pour chaque point du programme ainsi obtenu l'ensemble des variables vivantes. Construire le graphe d'interférence (en ne tenant compte que des variables locales du programme, i.e. en omettant les variables introduites par la transformation ci-dessus) et en déduire une manière d'allouer chacune des variables.

   (c) Proposer une fonction `transforme` qui étant données une variable r et une expression e, calcule la suite d'affectations équivalente pour le calcul de la valeur de e dans la variable r.