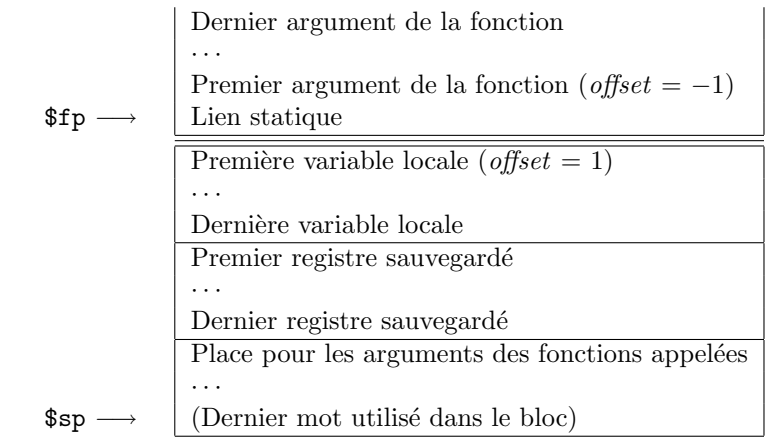
Compilation — TD 4

J. Chroboczek et P. Letouzey

3 Novembre 2006

Convention (rappel) La structure du bloc d'activation (*frame*) de chacune des fonctions d'un programme CTigre (y compris de la fonction principale) est la suivante :



L'*offset* d'une variable est sa position dans le bloc, compté en nombre de mots vers le bas à partir du début du bloc. Les *offsets* des variables du bloc commencent à partir de 1, le mot d'offset 0 étant occupé par le lien statique.

Dans la mesure où les paramètres de la fonction, déposés sur la pile par l'appelant *avant* l'appel de la fonction, sont situés juste au-dessus du bloc d'activation, il est naturel de leur associer des offsets négatifs (calculés de la même manière que précédemment).

Exercice 1 On considère le programme CTigre suivant :

```
let i := 10 in
let g(x : int) : int =
    let a := 3 in
        let f(y : int) : int = y * i in
            let h(z : int) : int = z / a in
    f(h(x)) + a
in g(12)
```

1. Indiquez pour chaque fonction la valeur de l'attribut *level* correspondant, et, pour chaque variable, les valeurs des attributs *level* et *offset* correspondants (le niveau du programme principal étant 1).
2. Décrivez la structure du bloc d'activation du programme principal et de chacune des fonctions.

3. Décrivez l'évolution de la pile lors de l'exécution du programme, en supposant que `$sp` = 2000 au début de l'exécution du programme.
   (On rappelle qu'à chaque appel de fonction, `$sp` pointe sur le lien statique fourni par l'appelant à la suite des paramètres de la fonction.)
4. Comment est compilé (en pseudo-code C) l'appel à la variable `a` dans le corps de la fonction `h` ?

   Vérifiez sur l'état de la pile au moment ou la fonction `h` est en cours d'exécution (question 3), que la suite d'accès mémoire effectuées par ce morceau de code est correcte.

Exercice 2 On considère le programme CTigre suivant :

```
let a := 3 in
let f(x : int, n : int) : int =
    let g(n : int) =
        if n = 0 then 1
                 else x * g(n - 1)
    in g(n)
in f(a, 2)
```

1. Décrivez la structure du bloc d'activation du programme principal et de la fonction f (après avoir déterminé les *levels* et les *offsets*).
2. Indiquez l'évolution de la pile au cours de l'exécution de ce programme, en supposant `$sp` initialisé à 2000.
3. Comment est compilé (en pseudo code C) l'appel à la variable x dans le corps de la fonction g ? (Vérifiez que ça marche lors de l'exécution.)

Exercice 3 – Initialisation des attributs *level* et *offset*

Le code fourni pour votre projet contient les définitions de constructeurs suivantes :

```
type exp_desc =
     VarExp of var
   ...
   | ForExp of forexp
   | LetExp of dec list * exp
   ...
and var = SimpleVar of attribvar
          ...
   ...
and attribvar = { vid: symbol;
                  v_level: int option;
                  v_offset: int option}
   ...
and dec = FunDec of fundec list
        | VarDec of vardec list
...
and forexp = { var: symbol;
               for_level: int option;
               for_offset: int option;
                ...}
```

```
and fundec = { fun_name: symbol;
               params: field list;
               result: symbol option;
               f_level: int option;
               ... }
and vardec = { v_name: symbol;
               var_typ: symbol option;
               var_level: int option;
               var_offset: int option;
               ... }
...
```

1.

1. Pourquoi le type `dec` est-il défini comme une liste ?
2. À quoi sert chacun des champs dont le nom se termine par `level` ou `offset` ? Pourquoi est-ce une option ?
3. Pourquoi des champs `for_level` et `for_offset` sont-ils inclus ? Seraient-ils nécessaires en C ou en Pascal ?

2. La semaine prochaine, nous implémenterons une fonction

$$\texttt{set\_ast\_attributes} \quad : \quad \texttt{exp} \rightarrow \texttt{exp}$$

qui remplit les champs `*_level` et `*_offset` d'un arbre de syntaxe. Pour cela, nous utiliserons une fonction auxiliaire

```
set_ast_attributes_helper
    int Symbol.table →
    (int * int) Symbol.table →
    int → int ref →
        exp → exp
```

qui prendra comme arguments respectifs :

- une table `ftbl` associant un *level* à chaque symbole de fonction visible ;
- une table `vtbl` associant une paire (*level*, *offset*) à chaque symbole de variable visible ;
- un entier `level` indiquant le *level* de la fonction parcourue ;
- une référence à un entier `offset` décrivant le prochain offset disponible.

1. Quel type de parcours fera la fonction `set_attributes_helper` ?
2. Pourquoi `offset` a-t-elle besoin d'être une référence ? Comment pourrait-on s'en passer dans un langage fonctionnel pur ?