SORBONNE UNIVERSITÉ

TD : Exceptions et Continuations — semaine 5

8 mars 2018

Objectif(s)

✯ Étude de l'implémentation des exceptions et continuations.

Exercice 1 – Exceptions

Considérons les deux programmes suivants :

Scheme	OCaml

```
(define (lastn i l)
  (if (pair? l)
    (let ((r (lastn i (cdr l))))
      (cond ((not (integer? r)) r)
        ((= r i) l)
        (#t (+ r 1)) ))
    0))
```

```
type 'a int_or_alpha_list =
  I of int | L of 'a list;;

let lastn i l =
  let rec lastn_aux l =
    match l with
    | [] -> I 0
    | _::t -> let r = lastn_aux t in
     (match r with
      | L al -> r
      | I ir -> if ir = i then (L l)
                else (I (ir+1)))
  in
    match (lastn_aux l) with
    | L al -> al
    | _ -> [];;
```

1. Que font ces deux programmes ? Et quelle est la principale différence entre ces deux programmes ?
2. Écrire une version C utilisant la structure de donnée `liste_entier` vue au td précédent.
3. On veut déclencher une exception dès que la liste cherchée est trouvée ou si la liste est trop courte. En utilisant un mécanisme d'exceptions, modifier les versions OCaml et C de ce programme. Modifier en conséquence les programme précédents. Pour la version C on utilisera l'ensemble de macros et fonctions vu en cours.

Exercice 2 – Programmation fonctionnelle en C

On cherche à écrire en C les fonctions principales d'une bibliothèque sur les listes à la manière de la bibliothèque `List` d'OCaml. Soit le programme suivant :

```
(* on suppose hd, tl et :: predefinies *)

let rec nth l n =
  if l = [] then failwith "nth"
  else let a = hd l and q = tl l in
```

```
        if n= 0 then a else
        if n > 0 then nth q (n-1) else
        invalid_arg "LIST.nth";;

let rec map f l=
  if l = [] then []
  else let a= hd l and q = tl l in
  let r= f a in r :: map f q;;

let rec fold_right f l accu =
  if l = [] then accu
  else let a = hd l and q = tl l in
     f a (fold_right f q accu);;
```

1. Définir en C les fonctions `failwith` et `invalid_arg` en utilisant des exceptions.
2. Traduire les fonctions `nth, map` et `fold_right` en C.
3. A quel moment dans les fonctions que vous venez d'ecrire, un GC pourrait se déclencher ?

Exercice 3 – Implantation de continuations en C

On cherche à définir des continuations en C. Une continuation nécessite de conserver un contexte d'exécution et l'état de la pile courante, autrement dit le code et les données. Lors de son appel, une continuation revient au contexte d'exécution sauvegardé et la pile d'exécution est restaurée—même si celle sauvegardée est plus grande que l'actuelle. On supposera que les registres sont bien traités par les `setjmp` et `longjmp`.

1. Définir un type C pour les continuations qui contiennent au moins un contexte d'exécution (`jmp_buf`) et un champ pour l'état de la pile.
2. Écrire une fonction `save_context` qui sauve l'état de la pile dans une continuation, puis qui retourne une continuation en ayant posé un point de retour `setjmp`. Attention, vérifier le sens de la pile.
3. Écrire une fonction `get_context` qui restaure l'état de la pile tel qu'il est conservé dans la continuation. Attention de bien vérifier que le niveau de la pile actuelle est suffisant pour la pile à restaurer. Effectuer un `longjmp` à la fin de cette fonction pour retourner au point d'exécution.