Structures Impératives

Alain Camanes

alain.camanes@free.fr

Stanislas

Option Informatique
2021-2022

Plan

Références

```
# let x = ref 0;;
val x : int ref = {contents = 0}
```

↪ Type polymorphe. `'a ref`.

↪ Modification.

```
# x := 2;;
- : unit = ()
# x;;
- : int ref = {contents = 2}
```

Déréférenciation

↪ Déréférenciation. Permet d'accéder au contenu de la référence. !

```
# !x ;;
- : int = 2

# let y = !x * !x;;
val y : int = 4

# x := !x + 1;;
- : unit = ()
# x;;
- : int ref = {contents = 3}
```

Plan

Boucle itérative

```
for indice = int1 to int2 do
  expr
done;;
```

↪ Bornes incluses. `int1` et `int2` sont de type `int` et évalués lors de l'entrée dans la boucle.
Possiblité de remplacer `to` par `downto`.

↪ Expression. `expr` est de type `unit`.

↪ Vocabulaire. `indice` est l'indice de la boucle.

↪ Comportement. Si `int1` $>$ `int2`, la boucle n'est pas effectuée.

Exemple

Calcul de la somme des n premiers entiers non nuls.

```
let somme n =
  let s = ref 0 in
  for i = 1 to n do
  (* Inv : s contient sum_{k=0}^i k *)
    s := !s + i
  done ;
!s;;
```

Boucle conditionnelle

```
while bool1 do
  expr
done;;
```

↪ Types. `bool1` est de type `bool` et `expr` est de type `unit`.

↪ Expressions. Une boucle est une expression de type `unit`.

```
# let cpt = ref 0;;
# while !cpt < 10 do
    print_int !cpt;
    cpt := !cpt + 1;
done;;

0123456789- : unit = ()
```

Exemple

```
let carre_ent n =
  let m = ref 0 in
  while (!m+1) * (!m+1) <= n do
  (* V(m) = n - m^2
     I(m) = m^2 <= n *)
    m := !m + 1
  done;
  !m ;;
```

↪ Variant. `V` est une fonction décroissante.

↪ Invariant. `I` est un invariant de boucle et `I` implique `V` $\geqslant 0$.

↪ Postcondition. Lorsque la condition est fausse, alors,

$$(m+1)^2 > n \text{ et } m^2 \leqslant n$$

Soit $m = \lfloor \sqrt{n} \rfloor$.

Plan

Tableaux

```
# let t = [|2; 3; 5|] ;;
val t : int array = [|2; 3; 5|]
```

↪ Mémoire. Ensemble mémoire de taille fixe.

↪ Type polymorphe. `'a array`.

↪ Structure de données mutable.

↪ Accès en temps constant...

- ...à la longueur `Array.length : 'a array -> int`.
- ...aux éléments `t.(indice)` (indices commencent à 0).

↪ Tableau vide. `[| |]`

Tableaux - Création

↪ Création. `Array.make : int -> 'a -> 'a array`

```
# Array.make 3 1 ;;
- : int array = [|1; 1; 1|]
# let t = [|1;1;1|] ;;
val t : int array = [|1; 1; 1|]
```

↪ Initialisation.

`Array.init : int -> (int -> 'a) -> 'a array`

```
# let carre x = x * x ;;
val carre : int -> int = <fun>
# Array.init 5 carre;;
- : int array = [|0; 1; 4; 9; 16|]
# let rec f n = if n = 0 then "c"
    else let s = f (n-1) in s^s;;
val f : int -> string = <fun>
# Array.init 4 f;;
- : string array = [|"c"; "cc"; "cccc"; "cccccccc"|]
```

Tableaux - Manipulation

↪ Structure de données mutable.

```
# t.(0) <- 3 ;;
- : unit = ()
# t ;;
- : int array = [|3; 1; 1|]
```

↪ Juxtaposition d'effets de bord.

```
# let tab = [|1;3;2|] ;;
val tab : int array = [|1; 3; 2|]

# let tmp = tab.(0) in
  tab.(0) <- tab.(1);
  tab.(1) <- tmp ;;
- : unit = ()

# tab ;;
- : int array = [|3; 1; 2|]
```

Séquences & Priorités

```
let test1 tab =
 if tab.(0) <= tab.(1)
 then tab.(0) <- 1
 else tab.(1) <- 0;
 if tab.(1) <= tab.(2)
 then tab.(2) <- 1
 else tab.(1) <- 2 ;;

let t = [|0;2;3|]
 in test1 t; t ;;

-:int array = [|1; 2; 1|]
```

```
let test2 tab =
 if tab.(0) <= tab.(1)
 then tab.(0) <- 1
 else
   begin
   tab.(1) <- 0;
   if tab.(1) <= tab.(2)
   then tab.(2) <- 1
   else tab.(1) <- 2
   end;;

let t = [|0;2;3|]
 in test2 t; t ;;

-:int array = [|1; 2; 3|]
```

Attention !

↪ Alias. Utiliser `Array.make_matrix` au lieu de `Array.make`.

```
# let t1 = [|1; 3; 5|] ;;
val t1 : int array = [|1; 3; 5|]
# let t2 = t1 ;;
val t2 : int array = [|1; 3; 5|]
# t2.(0) <- 3 ;;
- : unit = ()
# t1 ;;
- : int array = [|3; 3; 5|]
```

↪ Égalités structurelle & physique.

```
# let a = [|1;2;3|] and b = [|1;2;3|] ;;
val a : int array = [|1; 2; 3|]
val b : int array = [|1; 2; 3|]
```

```
# a = b ;;
- : bool = true
```

```
# a == b ;;
- : bool = false
```

= vs. == (Hors Programme)

↪ ==. Égalité physique des représentants en mémoire.

↪ =. Égalité sémantique : comparaison des valeurs jusqu'à la fin ou jusqu'à ce que deux valeurs distinctes soient représentées.

```
# [1] == [1];;
- : bool = false
# 1.0 == 1.0;;
- : bool = false
```

```
# let v = [|1;2;3|];;
val v:int array=[|1;2;3|]
# let u = v;;
val u:int array=[|1;2;3|]
```

```
# u == v;;
- : bool = true
# u == [|1;2;3|];;
- : bool = false
# u.(0) <- 0;;
- : unit = ()
# u;;
- : int array = [|0;2;3|]
# v;;
- : int array = [|0;2;3|]
```

Minimum d'un tableau

↪ *Récursivement*, à l'aide d'une fonction auxiliaire.

```
let min tab =
  let rec min_partiel m i =
    if i = Array.length tab then m
    else if m < tab.(i)
         then min_partiel m (i+1)
    else min_partiel tab.(i) (i+1)
  in min_partiel tab.(0) 0 ;;
```

↪ *Itérativement*, à l'aide d'une référence.

```
let min tab =
  let m = ref tab.(0) in
  for i = 1 to Array.length tab-1 do
    (if tab.(i) < !m then m := tab.(i))
  done; !m;;
```

Algorithme de Hörner

```
let horner p x =
  let n = Array.length p-1 in
  let px = ref p.(n) in
    (* I : Px = \sum_{k=0}^i a_{n-k} x^{i-k} *)
    for i = 1 to n do
      px := !px *. x +. p.(n-i)
    done;
    !px;;

val horner : float array -> float -> float = <fun>
```

n additions et n multiplications.

Plan

Fonctions pures

↪ Effets de bord (*side effects*). Modification d'un état en dehors de l'environnement local : modification d'une référence, d'une donnée mutable, opérations d'entrée/sortie (print,...),...

↪ Fonction pure.

- L'évaluation de la fonction n'a pas d'effets de bord.
- La valeur de retour est toujours la même si les arguments sont identiques.

```
# let pi = ref 3.14;

# let f () =
    pi := !pi +. 1.;
    !pi;;

val f : unit -> float
```

```
# let f x =
    x + 1;;

val f : int -> int
```

Expressions impures

↪ Expressions qui effectuent des *effets de bord*.

↪ Type de ces expressions : `unit`.

↪ *Exemples.*

```
# let a = ref 3.14;;
val a : float ref = {contents = 3.14}
# a := !a +. 10.;;
- : unit = ()
# for i = 1 to 4 do a := !a +. float_of_int i done;;
- : unit = ()

# print_float !a;;
23.14- : unit = ()

# let t = Array.make 10 1;;
val t : int array = [|1; 1; 1; 1; 1; 1; 1; 1; 1; 1|]
# t.(0) <- 3;;
- : unit = ()
```