Récursivité & Compléments

Alain Camanes

alain.camanes@free.fr

Stanislas

Option Informatique
2021-2022

Plan

Entiers naturels

$\hookrightarrow$ *Exemple.* `type nat = Zero | S of nat`

```
succ (succ (succ (succ (succ (succ 0)))))
```

$\hookrightarrow$ *Syntaxe.*

- 0,
- `succ n` où `n` est un entier.

Mots sur un alphabet.

$\hookrightarrow$ *Exemple.*

$$a.n.t.i.c.o.n.s.t.i.t.u.t.i.o.\varepsilon$$

$\hookrightarrow$ *Syntaxe.*

- Le mot vide ε,
- $e.m$ où e est l'élément d'un alphabet et m est un mot.

↪ *Exemple.*

$$(2+3)*5+(12*(7+(-1)))$$

↪ *Syntaxe.*

- un entier,
- (e) où e est une expression,
- $(e1+e2)$ où $e1$ et $e2$ sont des expressions,
- $(e1*e2)$ où $e1$ et $e2$ sont des expressions,
- $-e1$ où $e1$ est une expression.

↪ *Type.*

```
type expr = Nat of int
          | Par of expr
          | Plus of expr * expr
          | Mult of expr * expr
          | Opp of expr
```

Formules de la logique propositionnelle

↪ *Exemple.*

$$(\neg x_1 \vee x_2) \wedge (x_3 \vee x_1)$$

↪ *Syntaxe.*

- les éléments d'un ensemble de variables propositionnelles,
- $\perp$,
- (e) si e est une formule,
- $\neg e$ si e est une formule,
- $e1 \wedge e2$ si $e1$ et $e2$ sont des formules,
- $e1 \vee e2$ si $e1$ et $e2$ sont des formules.

↪ *Type.*

```
type form = False
          | Par of form
          | Neg of form
          | And of form * form
          | Or of form * form
```

Listes

↪ *Exemple.*

```
42 :: 2 :: 4 :: 12 :: []
```

↪ *Syntaxe.*

- `[]`,
- `e :: l` si `e` est un élément d'un type fixé et `l` une liste.

↪ *Type.*

```
type maliste = Empty
             | Concat of int * maliste
```

Ensembles inductifs

$\hookrightarrow$ Définition inductive. Soit E un ensemble. Une définition inductive d'une partie X de E consiste en

- un ensemble de base (ensemble des *constantes*) : un sous-ensemble non vide B de E ,
- un ensemble de règles R (ensemble des *constructeurs*) : chaque règle $r_i \in R$ est une fonction $r_i : E^{n_i} \to E$ d'arité n_i.

$\hookrightarrow$ Point fixe. Il existe un plus petit ensemble X tel que

- $B \subset X$,
- X est stable par R, i.e. pour tout $r_i \in R$ et tout $x_1, \ldots, x_{n_i} \in X$, $r_i(x_1, \ldots, x_{n_i}) \in X$.

Listes en OCaml

↪ Définition.

- Objets de base : $\mathscr{B}$ (entiers, flottants, tableaux,...),
- Constante : Liste vide : `[]`,
- Constructeur : `::`.

↪ Type. `'a list`.

```
# 1 :: 2 :: 3 :: 4 :: [] ;;
- : int list = [1; 2; 3; 4]
```

↪ Filtrage. liste vide ou `t::q`.

↪ Vocabulaire.

- *Tête* : $a_n :: a_{n-1} :: \cdots :: a_1 :: [] \mapsto a_n$.
- *Queue* : $a_n :: a_{n-1} :: \cdots :: a_1 :: [] \mapsto a_{n-1} :: \cdots :: a_1 :: []$.

Opérations (I)

↪ Tête. `List.hd : 'a list -> 'a`

```
let tete l =
  match l with
    | [] -> failwith "liste sans tete"
    | t :: q -> t ;;
```

↪ Queue. `List.tl : 'a list -> 'a list`

```
let queue l =
  match l with
    | [] -> failwith "liste sans queue"
    | t :: q -> q ;;
```

↪ Longueur. `List.length : 'a list -> int`

```
let rec longueur l =
  match l with
    | [] -> 0
    | t :: q -> 1 + longueur q ;;
```

Opérations (II)

↪ Concaténation. `@ : list -> list -> list` (infixe).

```
let rec conc l1 l2 =
  match l1 with
    | [] -> l2
    | t::q -> t :: (conc q l2) ;;
```

↪ *Complexité linéaire.*

Accumulateurs

`List.rev : 'a list -> 'a list`

↪ Renversement quadratique.

```
let rec miroir1 l =
  let rec insere_fin a l = match l with
      | [] -> [a]
      | t::q -> t :: (insere_fin a q)
  in match l with
    | [] -> []
    | t::q -> insere_fin t (miroir1 q) ;;
```

↪ Renversement linéaire.

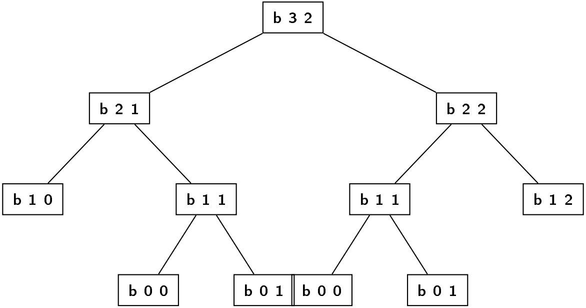
```
let miroir l =
  let rec miroir_aux l accu =  match l with
      | [] -> accu
      | t::q -> miroir_aux q (t::accu)
  in miroir_aux l [] ;;
```

Plan

Coefficients binomiaux : $\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}$

$$C_{0,p} = C_{n,p} = 1,\ C_{n,p} = C_{n-1,p} + C_{n-1,p-1} = \binom{n}{p}$$

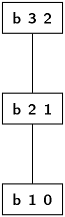
```
let rec bin n p =
  match (n,p) with
    | (_, 0) -> 1
    | (n, p) when p > n -> 0
    | (_, _) -> bin (n-1) p
      + bin (n-1) (p-1) ;;
```



Coefficients binomiaux : $\binom{n}{p} = \frac{n}{p}\binom{n-1}{p-1}$

$C_{n,p} = 2 + C_{n-1,p-1} = 2\min\{n, p\}$.

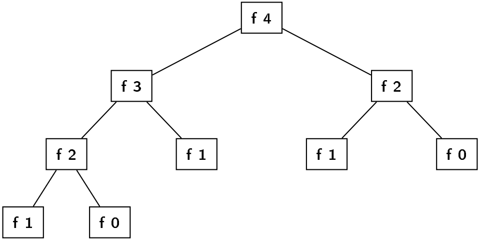
```
let rec bin n p =
  let q = min p (n-p) in match q with
    | 0 -> 1
    | _ ->  n * (bin (n-1) (p-1)) / p ;;
```



Suite de Fibonacci : $F_{n+2} = F_{n+1} + F_n$

$$F_n \sim \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n .$$

```
let rec fibo n = match n with
     |0 -> 0
     |1 -> 1
     |_ -> fibo (n-1) + fibo (n-2);;
```



Suite de Fibonacci : Accumulateur

n appels récursifs

```
let rec fibo n =
    let rec fibo_aux p a b = match p with
    | 0 -> 0
    | 1 -> b
    | _ -> fibo_aux (p-1) b (a + b)
  in fibo_aux n 0 1;;
```

fibo_aux 4 0 1

fibo_aux 3 1 1

fibo_aux 2 1 2

fibo_aux 1 2 3

Moralité sur la Récursivité

$\hookrightarrow$ Principe.

- Casser l'entrée en fonction de son type en un ensemble de cas,
- Traiter les cas de base,
- Supposer que la fonction termine sur les sous-cas,
- Reconstruire la sortie à l'aide des résultats des appels récursifs.

$\hookrightarrow$ Complexité. Récursion sans réflexion peut conduire à l'inondation !

Théorème (Principe de récurrence sur $\mathbb{N}$)

Soit P un prédicat sur l'ensemble des entiers. Alors,

$$[P(0) \text{ et } (\forall\ n \in \mathbb{N},\ P(n)\ \Rightarrow\ P(n+1))]\ \Rightarrow\ \forall\ n \in \mathbb{N},\ P(n).$$

Définition (Relation bien fondée)

Soit $(E, \leqslant)$ un ensemble muni d'une relation binaire. La relation binaire $\leqslant$ est bien fondée s'il n'existe pas de suite d'éléments de E infinie et strictement décroissante.

Exemples

$\hookrightarrow$ $\{n_0, n_0+1, \ldots\}$ pour tout $n_0 \in \mathbb{Z}$.

$\hookrightarrow$ $(\mathbb{N}\backslash\{0,1\}, |)$.

$\hookrightarrow$ $(\mathbb{N}^2, \preccurlyeq)$ pour l'ordre *lexicographique*.

$\hookrightarrow$ *Mots.* $m' \preceq m$ s'il existe a tel que $m = a.m'$.

$\hookrightarrow$ *Ensembles inductifs.* $x \preceq y$ s'il existe $r_i \in R$ et $x_1, x_2, \ldots, x_{n_i} \in E$ tels que

$$r(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_{n_i}) = y.$$

$\hookrightarrow$ Contre-exemple. $(\mathbb{Z}, \leqslant)$, $([0,1[, \leqslant)$.

Caractérisation des relations bien fondées

Définition (Élément minimal)

m est un élément minimal d'une partie A de $(E, \leqslant)$ si

$$\forall\ a \in A,\ a \leqslant m \ \Rightarrow\ m = a.$$

$\hookrightarrow$ *Exemples.* Les cas de base des ensembles inductifs.

Théorème

Il y a équivalence entre

1. *$(E, \leqslant)$ est bien fondée.*
2. *toute partie non vide de E possède un élément minimal.*

Caractérisation - Preuve

Théorème

$(E, \leqslant)$ est bien fondée ssi toute partie non vide de E possède un élément minimal.

$(\Leftarrow)$ Absurde : il existe $(u_n) \in \mathscr{S}(E)$ strictement décroissante.

$$S = \{u_n,\ n \in \mathbb{N}\}.$$

$S \neq \emptyset$, donc S admet un élément minimal $m = u_{n_0}$. Alors,

$$u_{n_0+1} < u_{n_0}.$$

Caractérisation - Preuve

Théorème

$(E, \leqslant)$ est bien fondée ssi toute partie non vide de E possède un élément minimal.

$(\Rightarrow)$ Contraposée : A une partie non vide sans élément minimal.

$$\forall\ m \in A, \exists\ a \in A\ ;\ \ a < m.$$

On construit ainsi une suite strictement décroissante.

Théorème (Principe d'induction)

Soit P un prédicat sur $(E, \leqslant)$ bien fondé. Si

$$\forall\, a \in E,\ (a\ \textit{minimal} \quad \Rightarrow \quad P(a))$$

et

$$\forall\, x, y \in E\ ;\ y \leqslant x,\ (P(y) \ \Rightarrow \ P(x))\,,$$

alors

$$\forall\, x \in E,\ P(x).$$

Démonstration.

$F = \{x \in E\ ;\ \text{non}\ P(x)\}$. *Absurde :* $F \neq \emptyset$.
Il existe un élément minimal $m_0 \in F$ et $P(m_0)$ est faux.
Par hypothèse d'initialisation, m_0 non minimal dans E.
Pour tout $y < m_0$, $y \notin F$, soit $P(y)$ est vrai et $P(m_0)$ est vrai.

Théorème (Terminaison)

Soient $(E, \leqslant)$ *un ensemble bien fondé,* $f : A \to X$ *une fonction récursive et* $\varphi : A \to E$*. On note* $\mathscr{M} = \{x \in A \ ; \ \varphi(x) \text{ est minimal dans } E\}$*. Si*

- *Le calcul de* f *termine sur tous les éléments de* $\mathscr{M}$*.*
- *Pour tout* $x \in A$*, le calcul de* $f(x)$ *ne nécessite qu'un nombre fini de calculs* $f(x_1), \ldots, f(x_n)$ *tels que* $\varphi(x_i) < \varphi(x)$*.*

Alors, la fonction f *termine.*

$$X_0 = \{a \in A \ ; \ f(a) \text{ ne termine pas}\},$$
$$F = \varphi(X_0).$$

Supposons par l'absurde $F \neq \emptyset$.
F possède un élément minimal $m_0 = \varphi(x_0)$.
f termine sur $\mathscr{M}$, donc $x_0 \notin \mathscr{M}$ et

$$M = \{m \in \varphi(A) \ ; \ m < m_0\} \neq \emptyset.$$

f termine sur M car m_0 est minimal.
$f(x_0)$ fait appel à un nombre fini de calculs de x_i tels que $\varphi(x_0) > \varphi(x_i) \in M$, donc $f(x_0)$ termine.

Terminaison - Exemple (I)

```
let rec bin n p = match (n,p) with
    | (_, 0) -> 1
    | (_, _) -> if p > n then 0
      else bin (n-1) p + bin (n-1) (p-1);;
```

$\mathbb{N}^2$ muni de l'ordre lexicographique.

$\hookrightarrow$ Le cas de base $(0, 0)$ est traité et la fonction termine sur $1 = \binom{0}{0}$.

$\hookrightarrow$ Soit $(n, p) \in \mathbb{N}^2$. On suppose que `bin` termine sur tout couple (n', p') strictement inférieur à (n, p) et qu'elle vaut $\binom{n'}{p'}$.
Si $p > n$ la fonction termine sur $0 = \binom{n}{p}$ (en particulier si $n = 0$).
Sinon, comme $(n-1, p) \prec (n, p)$ et $(n-1, p-1) \prec (n, p)$, `bin (n-1) p` et `bin (n-1) (p-1)` terminent et, d'après la formule du triangle de Pascal, le programme est valide.

Terminaison - Exemple (I)

```
let rec bin n p = match (n,p) with
    | (_, 0) -> 1
    | (_, _) -> if p > n then 0
      else bin (n-1) p + bin (n-1) (p-1);;
```

$\mathbb{N}$ muni de l'ordre usuel. Preuve par récurrence sur $n + p$

$\hookrightarrow$ Le cas de base $0 = 0 + 0$ est traité et la fonction termine sur $1 = \binom{0}{0}$.

$\hookrightarrow$ Soit $(n, p) \in \mathbb{N}^2$. On suppose que `bin` termine sur tout entier $n' + p'$ strictement inférieur à $n + p$ et qu'elle vaut $\binom{n'}{p'}$.
Si $p > n$ la fonction termine sur $0 = \binom{n}{p}$ (en particulier si $n = 0$).
Sinon, comme $n - 1 + p < n + p$ et $n - 1 + p - 1 < n + p$, `bin (n-1) p` et `bin (n-1) (p-1)` terminent et, d'après la formule du triangle de Pascal, le programme est valide.

Terminaison - Exemple (II)

```
let rec rev l = match l with
    | [] -> []
    | t::q -> (rev q) @ [t];;
```

L'ensemble des listes muni de l'ordre induit $\preceq$ par la structure inductive.

$\hookrightarrow$ Le cas de base `[]` est traité et la fonction renvoie la liste vide.

$\hookrightarrow$ Soit $\ell =$ `t::q` une liste. On suppose que `rev` termine pour toute liste $\ell' \preceq \ell$ et renvoie la liste retournée.
Alors, l'appel `rev q` termine et renvoie la liste renversée. Ainsi, `(rev q) @ [t]` termine et renvoie la liste ℓ renversée.

Plan

Récursivité terminale

Définition

Une fonction récursive est terminale si

- *elle effectue un seul appel récursif,*
- *il se trouve en dernière position.*

↪ *Formellement.*

```
let rec est_terminale n = match n with
| 0 -> 1
| _ -> est_terminale (n-1);;
```

↪ Dernier calcul effectué : l'*opération* ∗.

```
let rec fact n = match n with
  | 0 -> 1
  | n -> n * fact (n-1);;
```

↪ Aucun calcul en suspens.

```
let rec f_terminale n =
  match n with
    | base -> f_b n
    | _ -> bord n;
      f_terminale (f n);;
```

```
let f_iter x =
  let arg = ref x in
  while (base <> !arg) do
    bord !arg;
    arg := f !arg
  done;
  f_b !arg;;
```

Non Terminal → Terminal : Accumulateurs

```
let rec fact_nt n =
  match n with
    | 0 -> 1
    | _ -> n * fact_nt (n-1);;
```

```
let fact n =
  let rec fact_t n ac =
    match n with
      | 0 -> ac
      | _ -> fact_t (n-1) (ac*n)
  in
  fact_t n 1;;
```

Non Terminal → Terminal : Second exemple

↪ Non terminal.

```
let rec range a b =
  if a > b then []
  else a :: (range (a+1) b);;

# List.length (range 1 1000000);;
Stack overflow during evaluation(looping recursion?)
```

↪ Terminal.

```
let range1 a b =
  let rec aux a b acc =
    if a > b then acc
    else aux (a+1) b (a::acc)
  in aux a b [];;

# List.length (range1 1 1000000);;
- : int = 1000000
```

Plan

Que se passe-t-il ?

↪ *Exemple 1.* Quand évaluer ?

```ocaml
let f () = while true do () done;;

let g x y = x + 1;;

g 3 (f ());;
```

↪ *Exemple 2.* Que passe-t-on dans `f` ?

```ocaml
let f x = x + x in f (Random.int 5);;
```

Vocabulaire

↪ Paramètres formels : déclaration d'une fonction.

```
let f x1 ... xn = ...
```

↪ Paramètres effectifs : appel d'une fonction.

```
f e1 ... en
```

↪ Valeur :

- *type primitif* (booléen, entier, flottant,. . .)
- *pointeur* vers bloc mémoire (tableau, enregistrement, constructeur constant,. . .)

↪ Évaluation stricte. ✓
Opérandes / paramètres effectifs évalués *avant* l'opration / l'appel.

↪ Évaluation paresseuse. ×
Opérandes / paramètres effectifs évalués *si nécessaire*.

↪ *Exemple.*

```
let f () = while true do () done;;

let g x y = x + 1;;

g 3 (f ());;
```

Passage des paramètres

↪ Appel par valeur. ✓

Valeurs des paramètres effectifs affectées à de nouvelles variables.

```
let incr x = x := !x + 1;;

# let r = ref 41 in
    incr r; !r;;
- : int = 42
```

↪ Appel par nom. ×

Paramètres effectifs *substitués* aux paramètres formels, évalués si nécessaire.

Simulation en OCaml.

```
let f x =
  x () + x ();;

let v = f (fun () -> Random.int 4);;
val v : int = 5
```