Récursivité

I Définition. Exemples.

Définition .
On adopera la définition naïve suivante : une fonction récursive est une fonction qui s'appelle elle-même.

Exemples 1 Quelques exemples de fonctions qu'on peut définir de façon récursive :

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

let rec pgcd a b = match b with
| 0 -> a
| _ -> pgcd b (a mod b)

let rec binomial n k = match (n, k) with
| (_, 0) -> 1
| (0, _) -> 0
| _      -> binomial (n-1) k + binomial (n-1) (k-1)

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

let rec valuation p n = match n with
| 0 -> raise Division_by_zero
| _ when n mod p = 0 -> 1 + valuation p (n/p)
| _ -> 0
```

II Récursivité enveloppée, récursivité terminale, itération

J'ai essayé d'insister sur le fait que Caml était optimisé pour gérer efficacement la récursivité, là où Python est surtout efficace pour l'itération. Je vais maintenant essayer de montrer que l'opposition pertinente n'est pas à faire entre itération et récursivité mais plutôt entre récursivité terminale et récursivité enveloppée.

II.1 Itération

L'itération, c'est la répétition d'une suite d'instructions à l'aide d'une boucle. On dispose de deux type de boucles :

1. les boucles **conditionnelles**, qui sont les boucles `while` ;
2. et les boucles **inconditionnelles**, qui sont les boucles `for`

Exemple 2 On a vu avant les vacances de nombreuses fonctions pouvant s'écrire avec des boucles :
recherche dichotomique {dans un tableau trié, de racine de fonctions}, test de primalité, tri de tableau. Notons, et c'est important pour la suite, qu'on pouvait aussi les écrire sans boucle.

Exemple 3 Un autre ?
fibonacci, pgcd, binomiaux, valuations p-adiques

II.2 Récursivité enveloppée

La plupart des fonctions récursives qu'on écrit naturellement sont **récursives enveloppées**.

Définition .
Une fonction est dite **récursive enveloppée** lorsque ses appels récursifs se font *à l'intérieur* d'une fonction :

$$f(x) = \begin{cases} \text{si x est un cas de base alors une valeur de base} \\ \text{sinon } g(x, f(h_1(x)),\dots, f(h_k(x))). \end{cases}$$

La fonction g **enveloppe** les appels récursifs (ou l'unique appel récursif si $k = 1$).

Exemples 4

fibonacci `(+)`

factorielle `fun x -> n * x`

binomiaux `(+)`

valuations `fun x -> 1 + x`

L'utilisation de la récursivité enveloppée implique l'utilisation d'une pile d'exécution.

Exemple 5 Qu'est-ce qui se passe quand on lui demande de calculer 3!, en fait ?

Table 1 – Pile d'appel pour `fact 3`

Calcul	Pile
$3!$	$()$
$3 \times \boxed{2!}$	$(\times 3)$
$3 \times 2 \times \boxed{1!}$	$(\times 2, \times 3)$
$3 \times 2 \times \boxed{1\times 1}$	$(\times 1, \times 2, \times 3)$
$3 \times \boxed{2\times 1}$	$(\times 2, \times 3)$
$\boxed{3\times 2}$	$(\times 2)$
6	$()$

II.3 Récursivité terminale

Attention, toutes les fonctions récursives qu'on écrit ne sont pas récursives enveloppées !

Exemple 6 L'exemple typique :

Table 2 – Pile d'appel pour `pgcd 123 45`

Calcul	Pile
`pgcd 123 45`	$()$
`pgcd 45 33`	$()$
`pgcd 33 12`	$()$
`pgcd 12 9`	$()$
`pgcd 9 3`	$()$
`pgcd 3 0`	$()$
`0`	$()$

Définition .

Une fonction $f : E \to X$ est dite **récursive terminale** lorsqu'elle est de la forme :

$$f(x) = \begin{cases} \text{si x est un cas de base alors une valeur de base} \\ \text{sinon } f(g(x)) \end{cases}$$

Exemples 7 Autres exemples déjà rencontrés : `itere f (n-1) (f x)` *mais pas* `f (itere f (n-1) x)`.

Exemple 8 Écrivons maintenant une fonction **récursive terminale** qui calcule la factorielle.

```
let fact n =
  let rec aux n acc = match n with
  | 0 -> acc
  | _ -> aux (n-1) (n * acc)
in fact n 1
```

II.4 En guise de conclusion

Quelle est la différence entre itération et récursivité terminale ?

Il n'y en a pas.

III Terminaison des fonctions récursives

III.1 Terminaison des fonctions itératives

Ce **n'est pas** l'objet de ce chapitre. Mais un bref rappel (un appel ?) : pour montrer qu'une boucle `while` termine, on utilise un **variant de boucle**, c'est-à-dire une expression entière dépendant des variables de la fonction qui décroît strictement à chaque itération de la boucle.

Exemples 9 Variant de boucle pour la dichotomie : largeur de la plage de recherche.

III.2 Un problème à résoudre

Une fonction récursive n'a aucune raison de toujours terminer. Pourquoi ?

III.3 Cas des fonctions $\mathbb{N} \to X$

Lemme .

Si $f : \mathbb{N} \to X$ est une fonction récursive telle que :

- le cas $n = 0$ est un cas de base (la fonction ne s'appelle pas elle-même sur cet argument) ;
- les autres cas se traitent par (un nombre borné d') appels récursifs sur des entiers strictement plus petits ;

Alors f termine.

Démonstration. par l'absurde : si f strictement décroissante à valeurs dans $\mathbb{N}$, alors $|f^{\to}(\mathbb{N})| = +\infty$, alors on a une suite strictement décroissante d'entiers, impossible. □

Remarque 1

Même si on n'a pas $f : \mathbb{N} \to X$, on peut parfois s'y ramener !
Par exemple, pour montrer qu'une fonction définie sur les *'a listes* termine, il suffit de montrer :

- le cas sur la liste vide est un cas de base
- et les appels récursifs sur la liste se font sur des listes de longueur strictement décroissante.

Le cas de $\mathbb{N} \to X$ est en fait un cas particulier du cas général.
On verra comment retrouver le cas des *'a listes* comme un autre cas particulier de ce cas général plus bas.

III.4 Bon ordre sur un type ou un ensemble

Définition .

On dit que $(E, \leqslant_E)$ est **bien ordonné**, ou que $\leqslant_E$ **est un bon ordre sur** E lorsque toute partie non vide de E a un plus petit élément.

Exemples 10

— $(\mathbb{N}, \leq)$

— $(\mathbb{N}^2, \leq_{\text{lex}})$ avec $(a,b) \leq_{\text{lex}} (c,d) \overset{\text{def}}{\Leftrightarrow} a < c$ ou $(a = c$ et $b \leq d)$

Théorème .

Dans un ensemble bien ordonné, toute suite strictement décroissante d'éléments est **finie**.

III.5 Terminaison de fonctions récursives

Corollaire .

Si E est bien ordonné et $f : E \to F$ est une fonction récursive telle que :

- les éléments minimaux de E sont des cas de base (f ne s'appelle pas elle-même sur ces argument) ;
- les autres cas se traitent par appels récursifs sur des éléments de E strictement plus petits ;

Alors f termine.

Exemple 11 Un exemple : le calcul des binomiaux via la formule de Pascal (`binom`)

— Définie sur $(\mathbb{N}^2, \leq_{\text{lex}})$

— cas de base contient $\min_{(\mathbb{N}^2, \leq_{\text{lex}})} \mathbb{N}^2 = (0,0)$ (c'est `| (0, n) -> 1` avec $\texttt{n} = 0$)

— appels sur des entiers strictement décroissants $\binom{n-1}{k-1}$ et $\binom{n-1}{k}$ or $(k-1, n-1) <_{\text{lex}} (k,n)$ et $(k, n-1) <_{\text{lex}} (k,n)$.

D'après le théorème de terminaison, `binom` termine.

Remarque 2

Des fois, ça se passe mal (on ne peut pas utiliser le théorème précédent, mais les fonctions terminent quand même... ou ne terminent pas... ou on ne sait pas...). C'est la vie.

Exemples 12

Fonction 91 de McCarthy

```
let rec f n = if n > 100
  then n - 10
  else f (f (n + 11))
```

Syracuse

$$f := n \mapsto \begin{cases} 0 & \text{si } n \in \{0,1\} \\ f(3n+1) & \text{si } n \in 2\mathbb{N}+1 \\ f(\frac{n}{2}) & \text{si } n \in 2\mathbb{N} \end{cases}.$$

IV Correction des fonctions récursives

IV.1 Correction des fonctions itératives

Ce **n'est pas** l'objet de ce chapitre. Mais un bref rappel (un appel ?) : pour montrer qu'une fonction itérative est correcte, on utilise un **invariant de boucle**, c'est-à-dire un énoncé dépendant des variables de la fonction qui reste inchangé à chaque itération de la boucle, qui est vrai avant la première itération de la boucle, et qui donne le résultat demandé à l'issue de la dernière itération de la boucle.

Exemple 13 Prenons le calcul itératif d'une factorielle.

```
let fact n =
  let result = ref 1 in
  for i = n downto 1 do
    result := !result * i
  done;
  !result
```

Invariant :

```
!result * fact i
```

(à la sortie, `!result` $=$ `!result * fact 1` et $\forall i$, `!result * fact $i$ = fact n`

```
$\forall i,$ !result * fact $i$ = fact n
```

IV.2 Correction des fonctions récursives

Théorème .

Si E est bien ordonné alors on a $\Big(\forall x \in E,\ \big(\forall y < x,\ P(y)\big) \Rightarrow P(x)\Big) \Longrightarrow \big(\forall x \in E,\ P(x)\big)$.

Remarquons que pour $E = \mathbb{N}$, on retrouve le théorème de la récurrence forte.

Corollaire .

Si E est bien ordonné et $f : E \to F$ est une fonction récursive qui termine, telle que :

- f est correcte sur ses cas de base ;
- sur les autres cas, f est correcte en supposant qu'elle l'est sur ses appels récursifs ;

alors f est correcte.

Exemple 14 Reprenons l'exemple du calcul des binomiaux via la formule de Pascal.

— `binom` est correct sur les cas de base par définition

— Supposons les appels récursifs corrects. On a

$$\begin{aligned} \texttt{binom}(n)(k) &= \texttt{binom}(n)(k-1) + \texttt{binom}(n-1)(k-1) && \text{par définition} \\ &= \binom{n-1}{k-1} + \binom{n-1}{k} && \text{par hypothèse} \\ &= \binom{n}{k} && \text{d'après Pascal} \end{aligned}$$

.

Un algorithme du programme (on le reprend en TP) : l'exponentiation rapide.

On considère la fonction suivante.

```
let rec f x n = match n with
        | 0 -> 1
        | n when n mod 2 = 0 -> let a = f x (n/2) in a*a
        | n -> let a = f x (n/2) in x*a*a
```

1. Que permet-elle de calculer ?
2. Le démontrer.
3. Montrer qu'elle termine.
4. (Introduction au prochain chapitre :) quel est son intérêt par rapport à une version naïve ?

Comme on le voit avec cet exemple, il est parfois difficile de passer un algorithme en récursivité terminale.