Université Montpellier-II
UFR des Sciences - Département Informatique - Licence Informatique
UE GLIN302 - Programmation Applicative et Récursive

Cours No 4 : Premières Fonctions Récursives

Notes de cours - 2007-2012

1 Fonctions récursives

1.1 Principe

Une définition inductive d'une partie X d'un ensemble consiste à fournir la donnée explicite de certains éléments de X (base) et le moyen de construire de nouveaux éléments de X à partir d'éléments déjà construits.

Exemple : ensemble des valeurs de la fonction "factorielle" sur les entiers peut être donné par la fonction récursive suivante à partir de la donnée de base "$fact(0) = 1$" et de la règle "$fact(n) = n.fact(n-1)$".

```
1 (define (fact n)
2   (if (= n 0)
3       1
4       (* n (fact (- n 1)))))
```

Fonction récursive : fonction dont la définition inclus (au moins) un appel à elle-même.

Une autre version, en C :

```
1 int fact(int n) {
2   if (n == 0)
3     return 1;
4   else
5     return n * fact(n-1); }
```

Réflexion : considérer le sens du calcul entre les versions itératives et récursives au vu de l'associativité de la multiplication.

Apparté : de l'intérêt du type abstrait "GrandNombre" (factorielle calculée avec la fonction précédente).

```
1  factorielle de  0 = 1
2  factorielle de  1 = 1
3  factorielle de  2 = 2
4  factorielle de  3 = 6
5  factorielle de  4 = 24
6  factorielle de  5 = 120
7  factorielle de  6 = 720
8  factorielle de  7 = 5040
9  factorielle de  8 = 40320
10 factorielle de  9 = 362880
11 factorielle de 10 = 3628800
12 factorielle de 11 = 39916800
13 factorielle de 12 = 479001600
```

```
factorielle de 13 = 1932053504
factorielle de 14 = 1278945280
factorielle de 15 = 2004310016
factorielle de 16 = 2004189184
factorielle de 17 = -288522240
```

1.2 Itération et récursion

Rappel : **Itérer** : répéter n fois un processus en faisant changer la valeur des variables jusqu'a obtention du résultat.

Calcul itératif de factorielle d'un nombre : $n! = \prod_{i=1}^{n} i$

Un calcul itératif se programme par une boucle (*for* ou *while* ou *repeat-until*).

Exemple de fonction itérative pour le calcul de factorielle (en C).

```c
int fact(n) { // n entier
    int i = 0;
    int result = 1;
    while (i < n){
        // result = fact(i) -- invariant de boucle
        i = i + 1;
        result = result * i;
        // result = fact(i) -- invariant de boucle
    }
    // i = n
    return(result); }
```

Inconvénient : nécessité de gérer explicitement l'évolution des variables, l'ordre des affectations et les invariants de boucle.

Autre version plus courte en C :

```c
int factorielle˙iterative(int n) {
    int res = 1;
    for (; n > 1; n--) res *= n;
    return res;
}
```

1.3 Autres Exemples de fonction récursives simples

Multiplication : $an = a + a(n-1)$

```
(define (mult a n)
  (if (= n 0)
      0
      (+ a (mult a (- n 1)))))
```

Puissance : $a^n = a.a^{n-1}$

```
(define (exp a n)
```

```
  (if (= n 0)
      1
      (* a (exp a (- n 1)))))
```

Inverser une chaîne

Idée : $inverse(n) = concatener(dernier(n), inverse(sauf Dernier(n)))$

```
(define (inverse s)
  ;;string-length est une fonction préfé finie
  (let ((l (string-length s)))
    (if (= l 0)
      s
      (string-append (inverse (substring s 1 l)) (substring s 0 1)))))
```

1.4 Exemple : Calcul des termes de suites récurrentes

Toute valeur d'une suite récurrente de la forme :

$u_0 = initial$ et pour $n > 1, u_n = \Phi(u_{n-1}, n)$

peut être calculée par une fonction (de n'importe quel langage de programmation autorisant la définition de fonctions récursives) similaire à la fonction *Scheme* suivante :

```
(define (u n)
  (if (= n 0)
      initial
      (PHI (u (- n 1)) n)))
```

Par exemple calcul de factorielle de 5 :

```
(define initial 1) (define PHI *)
(u 5) -->  120
```

1.4.1 Suites arithmétiques

Tout terme d'une suite arithmétique de raison r de la forme :

$u_0 = initial$ et pour $n > 1, u_n = u_{n-1} + r$

peut être calculée par la fonction

```
(define (ua n r)
  (if (= n 0)
      initial
      (+ (ua (- n 1) r) r)))
```

Exemple : Multiplication de 4 par 6 : (ua 6 4) avec initial = 0

A noter que le code suivant ne fonctionne pas (voir cours No 3, liaison lexicale) :

```
(let ((initial 0)) (ua 3 4))
```

Pour **éviter de passer par une variable globale ou de passer un paramètre à chaque appel récursif**, on peut utiliser une version de la forme spéciale **let** permettant de définir des fonctions internes, temporaires et récursives.

```
(define (ua n r initial)
  (let f ((n n))
    (if (= n 0)
        initial
        (+ r (f (- n 1))))))

(ua 6 4 0)
= 24
```

1.4.2 Suites géométriques

Tout terme d'une suite géométrique de raison q de la forme :

$u_0 = initial$ et pour $n > 1, u_n = q.u_{n-1}$ peut être calculée par la fonction ug suivante :

```
(define (ug q n initial)
  (let f ((n n))
    (if (= n 0)
        initial
        (* q (f (- n 1))))))
```

Exemple : 4 puissance 3,

```
(ug 4 3 1)
= 64
```

1.5 Calcul de la somme des termes d'une suite

1.5.1 Exemple historique

La flèche de Zénon (philosophe paradoxal) n'arrive jamais à sa cible (ou Achille ne rattrape jamais la tortue) si on décrit le mouvement comme une suite d'étapes : parcourir la moitié de la distance puis la moitié de ce qui reste, puis la moitié de ce qui reste, etc.

la flèche n'arrive jamais car : $\lim_{n\to+\infty} \sum_{i=1}^{n} 1/2^i = 1$.

Ce que l'on peut vérifier avec :

```
(define fzenon (lambda (n)
    ;; la part de distance parcourue par la flèche à l'étape n
    (if (= n 1)
        (/ 1 (expt 2 n))
        (+ (sz (- n 1)) (/ 1 (expt 2 n))))))
```

Plus lisible, utiliser une fonction annexe pour calculer "(/1 (expt 2 n))"

```
(define (f n) (/ 1 (expt 2 n)))

```

```
(define (fzenon1 n)
  (if (= n 1)
      (f 1)
      (+ (f n) (fzenon1 (- n 1)))))
```

Mais la fonction f ainsi isolée est polluante.

La solution suivante est plus élégante.

```
(define fzenon2
  (let ((f (lambda (n) (/ 1 (expt 2 n)))))
    (let fzenon ((n n))
      (if (= n 1)
          (f 1)
          (+ (f n) (fzenon (- n 1)))))))
```

Elle suppose un bonne compréhension de la notion de portée et durée de vie des identificateurs.

1.5.2 Généralisation au calcul de la somme des termes de toute suite

```
(define (sommeSuite n)
  (if (= n 0)
      (u 0)
      (+ (u n) (sommeSuite (- n 1)))))
```

A essayer avec : (define (u n) (fact n))

Optionnel : même fonctionnalité en n'écrivant qu'une seule fonction récursive, à condition de passer la fonction du calcul d'un terme en argument. La fonction somme devient une fonctionnelle ou fonction d'ordre supérieur.

```
(define (sommeSuite n u)
  (if (= n 1)
      (u 1)
      (+ (sommeSuite (- n 1) u) (u n))))
```

On peut par exemple écrire :

```
(somme 10 (lambda (n) (/ 1 (exp 2 n))))
```

1.6 Interprétation d'un appel récursif

Appel récursif : appel réalisé alors que l'interprétation d'un appel précédent de la même fonction n'est pas achevé.

L'interprétation du code d'une fonction récursive passe par une phase d'expansion dans laquelle les appels récursifs sont "empilés" jusqu'à arriver à un appel de la fonction pour lequel une condition d'arrêt est vérifiée, alors suivie par une phase de contraction dans laquelle les résultats partiels précédemments empilés sont utilisés.

1.7 Découverte d'une solution récursive à des problèmes

Disposer d'une solution récursive à un problème permet d'écrire simplement un programme résolvant (calculant quelque chose de relatif à) ce problème. La découverte de telles solutions est parfois complexe mais rentable en terme de simplicité d'expression des programmes.

Exemple : Algorithme récursif de calcul du pgcd de deux nombres non nuls :

Require: $b \neq 0$
  if b *divise* a **then**
    $pgcd(a, b) = b$
  else
    $pgcd(a, b) = pgcd(b, modulo(a, b)))$
  end if

Implantation en Scheme :

```
1 (define (pgcd a b)
2   (if (= b 0)
3     (error "b doit être non nul")
4     (let ((m (modulo a b)))
5       (if (= m 0)
6         b
7         (pgcd b m)))))
```

1.8 Récursivité terminale et non terminale

Appel récursif non terminal : appel récursif argument d'un calcul englobant.

Exemple : l'appel récursif dans la définition de factorielle est non terminal car sa valeur est ensuite multipliée par n.

Appel récursif terminal appel récursif dont le résultat est celui rendu par la fonction contenant cet appel.

Exemple : appel récursif à pgcd dans la fonction précédente.

Propriété : l'interprétation d'un appel récursif terminal peut être réalisée sans consommer de pile.

Il est possible, en terme de mémoire, d'interpréter une fonction récursive terminale comme une fonction itérative car la gestion de la mémoire se déduit trivialement des transformations sur les paramètres.

1.9 Récursivité croisée

Exemple d'école "pair-impair" sur les entiers naturels

```
1 (define (pair n)
2   (or (= n 0) (impair (- n 1))))
3
4 (define (impair n)
5   (and (not (= n 0)) (pair (- n 1))))
```

Intéressant de découvrir "letrec" pour définir des récursions croisées.

```
(letrec <bindings> <body>)

Syntax: <Bindings> should have the form ((<variable1> <init1>) ...),
```

and <body> should be a sequence of one or more expressions. It is an error for a <variable> to appear more than once in the list of variables being bound.

Semantics: The <variable>s are bound to fresh locations holding undefined values, the <init>s are evaluated in the resulting environment (in some unspecified order), each <variable> is assigned to the result of the corresponding <init>, the <body> is evaluated in the resulting environment, and the value(s) of the last expression in <body> is(are) returned. Each binding of a <variable> has the entire letrec expression as its region, making it possible to define mutually recursive procedures.

```
(define (pair? n)
  (letrec ((est-pair? (lambda (n)
                        (or (= n 0) (est-impair?(- n 1)))))
           (est-impair? (lambda (n)
                        (and (not (= n 0)) (est-pair? (- n 1))))))
    (est-pair? n)))
```

1.10 Fonctions de dessin de figures fractales

Voir, http://classes.yale.edu/fractals/.

Vidéo : “Fractales à la recherche de la dimension cachée”, Michel Schwarz et Bill Jersey, 2010.

Autre cours : “Les images fractales en Scheme, Une exploration des algorithmes récursifs” - Tom Mens - University de Mons-Hainaut (U.M.H.).

Programmation avec effets de bord (impressions à l’écran) :

```
;; choisir langage PLT
(require (lib "graphics.ss" "graphics"))
(open-graphics)
(define mywin (open-viewport "Triangle de Sierpinsky" 600 600))
(define uneCouleur (make-rgb .5 0 .5))
```

Exememple des triangles de *Sierpinski* :

```
(define (s-carré n x y cote)
  ;; n nombre d'itérations
  ;; x, y : coordonées du coin supérieur gauche de la figure
  ;; cote : longueur du carré
  (if (= 0 n)
      ((draw-solid-rectangle mywin) (make-posn x y) cote cote uneCouleur)
      (let ((moitié (/ cote 2)) (quart (/ cote 4)))
        (s-carré (- n 1) (+ x quart) y moitié)
        (s-carré (- n 1) x (+ y moitié) moitié)
        (s-carré (- n 1) (+ x moitié) (+ y moitié) moitié))))
```

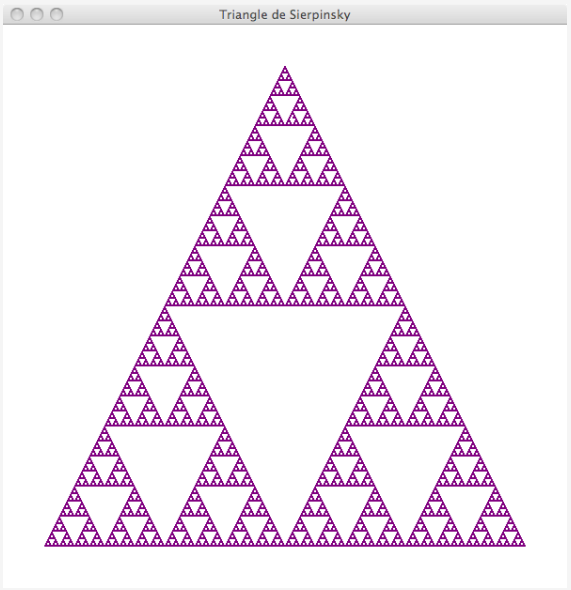

FIG. 1 – (s-carré 8 44 44 600) : sont tracés $3^{n_{initial}}$ carrés de coté "$cote_{initial}/2^{n_{initial}}$", soit pour $n_{initial} = 8$ et $cote_{initial} = 512$, $3^8 = 6561$ carrés, de côté $512/2^8 = 2$.