

# Programmation Fonctionnelle I, Printemps 2017 – TD10

<http://deptinfo.unice.fr/~roy>

L'univers arborescent des récurrences doubles...

**Exercice 1** Le cours page 11 a programmé le **parcours en profondeur préfixe** :

```
(define (arbre->prefixe A)
  (if (feuille? A)
      (list A)
      (cons (racine A) (append (arbre->prefixe (fg A)) (arbre->prefixe (fd A))))))
```

- a) Sur le même modèle, programmez le **parcours en profondeur infixe** d'un arbre : on visite le fils gauche, la racine, et enfin le fils droit :  $(\text{arbre} \rightarrow \text{infixe } '(+ (* x 2) y)) \rightsquigarrow (x * 2 + y)$
- b) L'inconvénient du parcours infixe précédent est qu'on ne peut pas reconstituer facilement l'arbre à partir de son parcours, à cause de la priorité des opérateurs. Par exemple le parcours  $(x * 2 + y)$  peut provenir de deux arbres distincts. Lesquels ?
- c) Modifiez-le pour lever l'inconvénient de b) :  $(\text{arbre} \rightarrow \text{infixe-par } '(+ (* x 2) y)) \rightsquigarrow ((x * 2) + y)$

**Exercice 2** Le **parcours en profondeur préfixe est inversible** : si une liste plate  $L$  est le parcours d'un arbre  $A$ , alors on peut reconstruire l'arbre  $A$ . Ce n'est pas difficile, mais la récurrence est un peu délicate.

- a) Essayons de prendre comme hypothèse de récurrence : *supposons le problème résolu sur le reste de la liste  $L$* . Montrez que ce choix est mauvais.
- b) Dans ce cas, faisons comme dans le cours 5 pages 18-19 : **généralisons les données du problème !** Travaillons sur une liste plate  $L$  débutant par un parcours préfixe et se terminant par n'importe quoi (que nous nommerons le *bruit*). Par exemple  $L = (+ * - x y z + u v w + y z)$ . Que vaut le *bruit* ici ?
- c) Montrez qu'il est maintenant légal de poser l'hypothèse de récurrence qui n'aboutissait pas en b).
- d) Il nous faut en plus gérer ce *bruit* : **généralisons le but du problème !** Qui devient : programmer une fonction (*arboriser*  $L$ ) prenant une liste plate  $L$  débutant par un parcours préfixe, et retournant une liste à deux éléments  $(A B)$  constituée de l'arbre  $A$  de ce parcours préfixe et de la liste plate  $B$  représentant le bruit qui suit ce parcours préfixe. Exemple :

```
(arboriser '(+ * - x y z + u v w + y z))  $\rightsquigarrow$  ((+ (* (- x y) z) (+ u v)) (w + y z))
                                     arbre          bruit
```

- e) En déduire la fonction (*préfixe* $\rightarrow$ *arbre*  $L$ ) prenant un parcours préfixe plat  $L$  et retournant l'unique arbre  $A$  dont  $L$  est le parcours. En d'autres termes, on veut la réciproque de la fonction (*arbre* $\rightarrow$ *prefixe*  $A$ ) et on déclenchera l'erreur *Parcours trop long* au besoin. Exemple :

```
(prefixe->arbre '(+ * - x y z + u v))  $\rightsquigarrow$  (+ (* (- x y) z) (+ u v))
(prefixe->arbre '(+ * x 2 3 4 5))  $\rightsquigarrow$  ERROR : Parcours trop long (4 5)
```

L'exercice ci-dessous traite des listes mais d'un point de vue arborescent. En effet, une liste  $L$  s'écrit sous la forme  $(\text{cons } (\text{first } L) (\text{rest } L))$  et peut être vue comme un arbre binaire de racine *cons*, de fils gauche le *first* et de fils droit le *rest* ! Parcourir une liste en plongeant d'abord à l'intérieur du *first* avant de visiter le *rest* revient à effectuer un **PARCOURS EN PROFONDEUR** ou **ARBORESCENT** de la liste.

**Exercice 3** Une liste est dite « plate » lorsqu'aucun de ses éléments n'est une liste. Par exemple la liste  $(2 \text{ foo } 5)$  est plate tandis que  $(2 (\text{foo}) 5)$  ne l'est pas. Programmez la fonction (*aplatir*  $L$ ) prenant une liste  $L$  et retournant une copie « plate » de  $L$  (*on chasse les parenthèses internes !*) :

```
(aplatir '(H (So 3) () (K (Cl 2))))  $\rightarrow$  (H So 3 K Cl 2)
```

## Exercices Complémentaires

**Exercice 4** [Examen] Programmez une fonction (arbre->anglais A) retournant une liste contenant une description en anglais des calculs effectués par l'arbre A. *Supposez par récurrence que vous savez traduire les sous-arbres !*

```
> (arbre->anglais '(+ (* x 2) (* y (- z 3))))  
(the sum of the product of x and 2 and the product of y and the difference of z and 3)
```

**Exercice 5** [Examen]. Etudiez le problème solve du cours page 19 :

```
> (solve '(+ (* (* 2 x) y) z) '(+ u 1)) ; résoudre  $(2x)y + z = u + 1$  comme équation en x  
(/ (- (+ u 1) z) (* 2 y))
```


Indication : pour programmer (solve A B) où il est garanti que x ne figure que dans A et il n'y figure qu'une seule fois, il s'agit de *faire passer de l'autre côté* tous les éléments de A jusqu'à ce que A soit réduit à x. C'est moins difficile qu'il n'y paraît...

---

# Programmation Fonctionnelle I, Printemps 2017 – TP10

<http://deptinfo.unice.fr/~roy>


**Exercice 1** a) Programmez une fonction (`compter A op`) retournant le nombre d'apparitions de l'opérateur `op` dans l'arbre `A` :

(`compter '(+ (/ (+ 2 3) 4) (+ x y)) '+`)  3


b) Programmez une fonction (`transformer A op1 op2`) prenant un arbre binaire d'expression `A`, et retournant une copie de l'arbre `A` dans laquelle chaque apparition de l'opérateur `op1` aura été remplacée par l'opérateur `op2` :

> (`transformer '(+ (/ (+ 2 3) (* (- 4 x) y)) (+ x y)) '+ '*`)  
(`* (/ (* 2 3) (* (- 4 x) y)) (* x y)`)

**Exercice 2** a) Programmez la fonction (`valeur A AL`) citée dans le cours page 20, qui retourne la valeur d'un arbre `A` en présence d'une liste `AL` d'associations (*variable valeur*). Une telle liste se nomme une **A-liste**. Il s'agit donc d'étendre aux arbres *algébriques* la fonction (`valeur A`) de la page 10 du cours :


(`valeur '(+ (* x 2) y) '((x 3) (y 1))`)  7

b) Programmez une fonction (`remplacer A AL`) prenant un arbre algébrique `A` et remplaçant dans `A` chaque variable par la valeur indiquée dans la `A-liste` `AL`. On déclenche l'erreur *Variable inconnue* si une variable de `A` ne figure pas dans `AL`. Exemple :

(`remplacer '(+ (* (- x 8) y) x) '((x 3) (y 2))`)  (+ (\* (- 3 8) 2) 3)


c) En déduire une autre solution possible de la question a). Laquelle est la plus efficace ?

**Exercice 3** Terminez le simplificateur (`simplif A`) du cours page 15, en programmant les spécialistes `simplif-`, `simplif*` et `simplif/`. Tâchez d'avoir de plus de règles de simplification possibles, mais vous ne parviendrez bien entendu pas au niveau du simplificateur de Mathematica ! Pas grave...

(`simplif '(+ (* x (- 5 (+ 3 2))) (/ (- y 0) (/ z z)))`)  y

## Exercices Complémentaires

**Exercice 4** Programmez la fonction (`miroir A`) retournant l'image inversée [à tous les niveaux] de l'arbre `A` :

(`miroir '(+ (/ (+ 2 3) 4) (+ x y))`)  (+ (+ y x) (/ 4 (+ 3 2)))


**Exercice 5** Programmez (`sous-arbres A op`) retournant la liste de tous les **sous-arbres** de `A` de racine `op` :

(`sous-arbres '(+ (/ (+ 2 3) 4) (+ x y)) '+`)  ((+ (/ (+ 2 3) 4) (+ x y)) (+ 2 3) (+ x y))

**Exercice 6** Terminez le dérivateur (`diff A v`) du cours page 16 en programmant les deux spécialistes `diff*` et `diff/`. Exemple :

(`diff '(+ 3 (- (* x x) (/ x 5))) 'x`)  (- (\* 2 x) 0.2) ; 0.2 ou 1/5

**Exercice 7** [Examen] Programmez une fonction (`ens-vars A`) retournant l'ensemble des variables de l'arbre `A`. On entend ici par « ensemble » une liste plate sans répétition, dans un ordre quelconque. Exemple :

(`ens-vars '(+ (* x (- z x)) (* (+ y 1) z))`)  (y x z) à l'ordre près...