Programmation Fonctionnelle I, Printemps 2017 – TD10

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

Exercice 10.1 a) Programmez le parcours en profondeur *postfixe* produisant la liste plate de tous les objets rencontrés :

`(arbre->postfixe '(+ (* x 2) y))` → `(x 2 * y +)`

b) Programmez le parcours *infixe* :

`(arbre->infixe '(+ (* x 2) y))` → `(x * 2 + y)`

c) 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. Modifiez-le pour qu'il produise un résultat non ambigü, avec des parenthèses :

`(arbre->infixe-par '(+ (* x 2) y))` → `((x * 2) + y)`

Exercice 10.2 a) Complétez la fonction `(arboriser L)` du cours 10 page 20.

b) En déduire la fonction `(prefixe->arbre L)` prenant un parcours préfixe plat `L` et retournant l'arbre `A` dont `L` est le parcours. En d'autres termes, on veut la réciproque de la fonction `(arbre->préfixe A)` du cours page 16. Exemple :

```
> (prefixe->arbre '(+ * - x y z + u v))
(+ (* (- x y) z) (+ u v))
```

Exercice 10.3 [*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 10.4 [*Examen*]. Etudiez le problème `solve` du livre de cours PCPS page 206 exercice 10.2.3 :

```
> (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 10.1 **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 occurrence 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 10.2 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 10.3 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 10.4 **a)** Complétez la fonction `(valeur A AL)` du cours page 11, qui retourne la valeur d'un arbre `A` en présence d'une liste de couples `(variable valeur)` représentée par la A-liste `AL` :

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

b) Programmez une fonction `(remplacer A AL)` prenant un arbre algébrique `A` et replaçant dans `A` chaque variable par la valeur indiquée dans la A-liste `AL`. On déclenchera l'erreur "Variable inconnue !" si une variable 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 10.5 [*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…*

Exercice 10.6 Programmez une **fonction d'ordre supérieur** `(map-feuilles A f)` prenant un arbre A et une fonction f à une variable, et retournant une copie de l'arbre `A` dans laquelle chaque feuille aura été remplacée par son image par la fonction `f`.
Par exemple, remplaçons toutes les variables d'un arbre par un point d'interrogation :

```
> (map-feuilles '(+ (* 2 (- 3 x)) (* (+ y 1) z)) (λ (u) (if (number? u) u '?)))
(+ (* 2 (- 3 ?)) (* (+ ? 1) ?))
```