Université de Bordeaux - Collège S&T — Licence S&TS Mention Informatique S5
2014/2015

UE INF5011 — Programmation 3

Programmation Fonctionnelle et Symbolique

Devoir surveillé

Tous documents autorisés. Mercredi 15 octobre 2014 Durée : 1h20.
Le barème est donné à titre **indicatif**.

Exercice 1 *(2pts)*

Évaluer les expressions suivantes :

1. `(cons 1 (cons nil 2))`
2. `(list 1 nil 2)`
3. `(last '((1) (2 3)))`
4. `(mapcan (lambda (x) (if (= x 3) '() (list x))) '(1 3 5 4 6))`

Exercice 2 *(3pts)*

5. Écrire une fonction `list-to-pairs (l)` qui prend en paramètre une liste d'entiers et retourne un ensemble de paires (e, k) où e est un élément de `l` et k son nombre d'occurrences[1] dans `l`. Il ne peut y avoir qu'une paire par élément. Exemple :

```
CL-USER> (list-to-pairs '(1 2 3 2 4 1 3 2))
((4 . 1) (1 . 2) (3 . 2) (2 . 3))
```

1. On pourra se servir de la fonction `count (e l)` qui compte le nombre d'occurrences de `e` dans une liste `l`.

Un *multi-ensemble* est un ensemble pouvant contenir plusieurs occurrences d'un même élément. Le nombre d'occurrences d'un élément est appelé sa *cardinalité*. On considère des multi-ensembles d'entiers naturels uniquement.
Soit l'ensemble de fonctionnalités (API) donné par la Figure **??** en Annexe (page **??**) et permettant de construire et de manipuler des multi-ensembles. On trouvera aussi en Annexe des exemples d'utilisation de cette API Figure **??**. Ces exemples doivent fonctionner dans toutes les implémentations.

Exercice 3 *(6pts)*

Dans un premier temps, on va implémenter les multi-ensembles à l'aide de listes pouvant contenir des doublons. Par exemple, la fonction qui retourne un multi-ensemble vide s'écrit :

```
(defun m-empty ()
  '())
```

et la fonction qui donne la cardinalité d'un élément

```
(defun m-cardinality (e mset)
  (count e mset))
```

6. Implémenter la fonction `m-set (mset)` qui retourne la liste des éléments d'un multi-ensemble (sans doublon). L'ordre est sans importance. Exemple :

   ```
   CL-USER> *m1*
   (1 1 1 0 0 0 0 1 2 2 2)
   CL-USER> (m-set *m1*)
   (0 1 2)
   CL-USER> *m2*
   (3 3 0 0 3 4)
   CL-USER> (m-set *m2*)
   (0 3 4)
   ```

Implémenter les fonctions de l'API suivantes :

7. `m-adjoin`,
8. `m-union`,
9. `m-intersection`.

Exercice 4 *(7pts)*

La représentation sous forme de liste ne permet pas de manipuler des multi-ensembles **infinis**. Pour cette raison, on propose une nouvelle implémentation dans laquelle un multi-ensemble est représenté par la **fonction** qui donne la cardinalité de ses éléments (et 0 si l'élément n'appartient pas au multi-ensemble). Ainsi, la fonction qui retourne un multi-ensemble vide s'écrit :

```
(defun m-empty ()
  (lambda (e)
    (declare (ignore e))
    0))
```

et la fonction `m-cardinality`

```
(defun m-cardinality (e mset)
  (funcall mset e))
```

10. Implémenter la fonction `m-full (&optional (count 1))` qui retourne le multi-ensemble infini contenant tous les entiers naturels ayant chacun pour cardinalité `count`. Exemple :

    ```
    CL-USER> (m-cardinality 100 (m-full 10))
    10
    CL-USER> (m-cardinality 100 (m-remove 100 (m-full 10) 2))
    8
    ```

Implémenter les fonctions

11. `m-adjoin`,
12. `m-union`.

L'opération `m-set` n'est pas implémentable car l'ensemble des éléments n'est pas accessible et peut être infini.

13. Implémenter une variante de `m-set`, `m-subset (n mset)` qui retourne la liste (avec multiplicité) des éléments du multi-ensemble `mset` qui sont inférieurs ou égaux à `n`. Exemple :

    ```
    CL-USER> (m-subset 10 (m-adjoin 0 (m-adjoin 10 (m-adjoin 11 *m2*))))
    (0 0 0 3 3 3 4 10)
    ```

FIN

Annexe

fonction	valeur retournée
`m-empty ()`	multi-ensemble vide
`m-cardinality (e mset)`	cardinalité d'un élément dans un multi-ensemble
`m-adjoin (e mset &optional (count 1))`	adjonction d'un élément dans un multi-ensemble
`m-remove (e mset &optional (count 1))`	suppression d'un élément d'un multi-ensemble
`m-union (mset1 mset2)`	union de deux multi-ensembles
`m-intersection (mset1 mset2)`	intersection de deux multi-ensembles

FIGURE 1 – API pour les multi-ensembles

Remarque : Un élément de cardinalité c_1 dans `mset1` et c_2 dans `mset2` aura pour cardinalité $c_1 + c_2$ dans l'union et $min(c_1, c_2)$ dans l'intersection.

```
CL-USER> (defparameter *m1*
           (m-adjoin 1 (m-adjoin 0 (m-adjoin 1 (m-adjoin 2 (m-empty) 3)) 4) 3))
*M1*
CL-USER> (m-cardinality 1 *m1*)
4
CL-USER> (m-cardinality 1 (m-adjoin 1 *m1* 10))
14
CL-USER> (m-cardinality 1 (m-remove 1 *m1* 2))
2
CL-USER> (defparameter *m2*
           (m-adjoin 3 (m-adjoin 0 (m-adjoin 3 (m-adjoin 4 (m-empty))) 2) 2))
*M2*
CL-USER> (m-cardinality 0 (m-union *m1* *m2*))
6
CL-USER> (m-cardinality 0 (m-intersection *m1* *m2*))
2
CL-USER> (m-cardinality 2 (m-intersection *m1* *m2*))
0
```

FIGURE 2 – Exemples d'utilisation de l'API