Université de Bordeaux - Collège S&T — Licence S&TS Mention Informatique S5
2015/2016

UE INF5011 — Programmation 3

Programmation Fonctionnelle et Symbolique

Devoir surveillé

Tous documents autorisés. Mercredi 14 octobre 2015 Durée : 1h20.
Le barème est donné à titre **indicatif**.

Exercice 1 *(3pts)*

Soit la structure de paires de la figure 1.

1. Donner la valeur affichée par le printer pour une telle structure.
2. Écrire une expression contenant uniquement des atomes et des appels à la fonction `cons`, et qui retourne un telle structure.
3. Simplifier autant que possible l'expression précédente en utilisant aussi la fonction `list`.

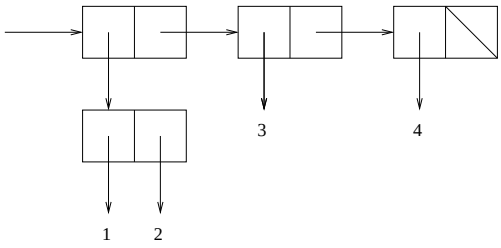

FIGURE 1 – Structure de paires

Exercice 2 *(3pts)*

4. Écrire une fonction `couples-elements (couples)` qui étant donnée une liste de couples d'entiers `couples`, chaque couple étant une liste de deux nombres, retourne l'ensemble des nombres présents dans `couples`. Exemples :

```
CL-USER> (couples-elements '())
NIL
CL-USER> (couples-elements '((1 2) (2 3) (4 2)))
(1 3 4 2)
```

On souhaite manipuler des *relations binaires* sur l'ensemble des entiers naturels autrement dit des sous-ensembles de $\mathbb{N} \times \mathbb{N}$. On considère l'ensemble des fonctions (API) permettant de construire et de manipuler des relations donné en Annexe page 3. On trouvera aussi en Annexe, une utilisation de cette API.

Exercice 3 *(7pts)*

5. En supposant la fonction `rel-add` connue, implémenter la fonction `rel-adds`.
6. En utilisant la fonction `rel-adds`, implémenter la fonction
   `add-reflexive-couples (elements relation)`
   qui retourne la relation qui contient tous les couples de `relation` plus les couples (i, i) pour tout i appartenant à la liste `elements`.
7. En utilisant, la fonction `rel-elements` et la fonction `add-reflexive-couples` du point précédent, implémenter la fonction `rel-make-reflexive`.

Dans un premier temps, on va implémenter les relations à l'aide de listes. Ainsi la relation vide est représentée par la liste vide et on peut définir [1]

```
(defun rel-empty () '())

(defun rel-elements (relation)
  (couples-elements relation))
```

8. En utilisant la représentation par liste, implémenter la fonction `rel-add`.

On propose une deuxième implémentation dans laquelle une relation est représentée par une *fonction d'un argument* définie comme suit :

- quand l'argument vaut `NIL`, la fonction retourne l'ensemble des nombres entiers apparaissant dans les couples de la relation.
- quand l'argument est un couple d'entiers (une liste de deux entiers), la fonction retourne vrai si le couple appartient à la relation et faux sinon.

Ainsi, la fonction `rel-empty` s'écrira :

```
(defun rel-empty ()
  (lambda (c)
    (if (null c)
        '()
         nil)))
```

et la fonction `rel-elements`

```
(defun rel-elements (relation)
  (funcall relation nil))
```

Exercice 4 *(7pts)*

En utilisant la représentation fonctionnelle, implémenter les fonctions suivantes :

9. `rel-member`,
10. `rel-add`,
11. `rel-make-symmetric`.

FIN

1. en utilisant la fonction `couples-elements` vue précédemment.

Annexe

API pour les relations binaires

`rel-empty ()`	la relation vide.
`rel-add (couple relation)`	la relation contenant tous les couples de la relation `relation` ainsi que le couple `couple`.
`rel-adds (couples relation)`	la relation contenant tous les couples de `relation` ainsi que les couples de la liste `couples`.
`rel-member (couple relation)`	vrai si `couple` appartient à `relation`, faux sinon.
`rel-elements (relation)`	ensemble (sous forme de liste) des entiers apparaissant dans `relation`.
`rel-make-reflexive (relation)`	relation contenant tous les couples de `relation` ainsi que tous les couples (i, i) pour tout $i \in$ `rel-elements(relation)`.
`rel-make-symmetric (relation)`	relation contenant tous les couples (i, j) de `relation` ainsi que le couple symétrique (j, i).

Exemple d'utilisation de l'API

```
CL-USER> (defparameter *r1*
           (rel-adds '((1 2)  (2 1)  (3 3)  (3 1)) (rel-empty)))
*R1*
CL-USER> (rel-elements *r1*)
(2 3 1)
CL-USER> (rel-member '(2 1) *r1*)
T
CL-USER> (rel-member '(1 3) *r1*)
NIL
CL-USER> (rel-member '(1 1) (rel-make-reflexive *r1*))
T
CL-USER> (rel-member '(4 4) (rel-make-reflexive *r1*))
NIL
CL-USER> (rel-member '(1 3) (rel-make-symmetric *r1*))
T
CL-USER> (rel-member '(1 4) (rel-make-reflexive *r1*))
NIL
```