Examen en “Programmation Applicative” - FLIN304

L2 Informatique

Session 1, année 2008-2009, janvier 2009.

Durée : 2h. Tous documents autorisés. Barème indicatif pouvant être modulé selon les résultats. Sections indépendantes.

1 Fonctions sur les listes (environ 5 points)

Questions

1. Écrire en SCHEME la fonction `oterRepet` qui rend une liste d'entiers dans laquelle il n'y a plus de répétitions consécutives. L'ordre des éléments dans le résultat compte. Attention au test pour le cas où il n'y a plus qu'un seul élément dans la liste.
Par exemple `(oterRepet '(1 5 5 5 4 3 3 5 1 1))` vaut `(1 5 4 3 5 1)`.
2. Ecrire une fonction `oterToutesRepet` qui supprime toutes les répétitions, mêmes non consécutives, dans une liste de nombres. L'ordre des éléments dans la liste résultat est indifférent.
Par exemple : `(oterToutesRepet '(1 5 5 5 4 3 3 5 1 1))` peut renvoyer `(4 3 5 1)` ou `(1 5 4 3)` ou ...
On rappelle que `(member e1 l)` rend la sous-liste de `l` commençant à l'élément `e1` si celui-ci se trouve dans la liste (comparaison effectuée avec `equal ?`) et rend `#f` sinon.
3. Donnez une version récursive terminale de `oterToutesRepet`.

2 Abstraction de données : les ensembles (environ 6 points)

Cet exercice propose de définir en scheme le type abstrait de données “ensemble”. Un ensemble est une collection de données ayant différentes caractéristiques comme celle de ne pas contenir deux fois le même élément. Il existe plusieurs problèmes à résoudre. Un premier problème est celui de la représentation, la première solution proposée ici est de représenter un ensemble à l'aide d'une liste contenant tous ses éléments. Un second problème est de savoir ce que l'on peut ranger dans un ensemble. Nos ensembles pourront contenir n'importe quelle sorte d'éléments représentables en SCHEME. Un troisième problème consiste à définir l'égalité entre éléments ; dans notre solution l'égalité sera l'égalité sémantique définie par la fonction `equal ?` de SCHEME.

Exemples :

```
> (define s1 (faire-ens '(1 2 2 3 #\a 5 "abc")))
= #t
> (define s2 (faire-ens '(1 2 #\a 4)))
> (define s3 (ajout "abc" s2))
= (3 5 "abc" 1 2 #\a 4)
> (vide? s1)
; l'ordre des élément peut être différent
= #f
> (retrait #\a s3)
= ("abc" 1 2 4)
```

Questions

1. Définissez, en supposant définie la fonction `oterToutesRepet` de la section précédente, la fonction `(faire-ens l)` qui rend une liste sans répétitions représentant l'ensemble contenant tous les éléments donnés dans `l`. La liste `l` peut être vide.
2. Définissez les fonctions booléennes `(vide? e)` et `(appartient? el e)`.
3. Définissez les fonctions `(ajout el e)` et `(retrait el e)` permettant d'ajouter et de retirer un élément d'un ensemble.
4. Définissez la fonction `(union e1 e2)`
5. Donnez une version récursive terminale (sans paramètre supplémentaire) de `union`.
6. Envisageons de ne plus représenter un ensemble par une liste mais par un autre type de collection comme vecteur ou tableau. Expliquez, sans les réécrire, quelles sont les fonctions, parmi les précédentes, que vous devez modifier pour réaliser ce changement. Justifiez Donnez un exemple de nouvelle définition d'une de ces fonctions.