Examen LISP '99

Écrire les définitions des fonctions numérotées de 1. à 13. ci-dessous.

1. Nombres binaires

1. `Num_Bin`
   argument : deux entiers n et M (en base 10)
   retourne : une liste *binaire* (formée uniquement de *0* et *1*) de longueur M, qui représente l'entier n en base deux. **Exemple**:
   ```
   >(Num_Bin 6 4)
   (0110)
   ```

2. `Bin_Num`, la fonction inverse
   argument : une liste binaire
   retourne : l'entier représenté par cette liste. **Exemple :**
   ```
   >(Bin_Num (0110))
   6
   ```

3. `NumList_BinList` (en utilisant éventuellement la fonction `Num_Bin`)
   argument : une liste l d'entiers en base 10 et un entier M
   retourne : une liste de listes binaires (de taille M chacune) représentant les entiers de l.
   Exemple :
   ```
   >(NumList_BinList (3 5) 3)
   ((011) (101))
   ```

2. Listes

4. `NTh`
   argument : une liste quelconque l et un entier n
   retourne : l'élément à la position n dans la liste l.

5. `IsOne`
   argument : une liste binaire l et un entier n
   retourne : `t` (vrai) si dans la liste l, à la position n (de gauche à droite), il y a un '1' et `nil` autrement. **Exemple :**
   ```
   >(IsOne (011) 3)
   t
   ```
   et:
   ```
   >(IsOne (011) 1)
   nil
   ```

6. `FirstOne`
   argument : une liste binaire l
   retourne : la position, de gauche à droite, du premier élément égal à 1 dans l. **Exemple :**
   ```
   >(FirstOne (011))
   2
   ```

7. `Zeros`
   argument : un entier n
   retourne : une liste de longueur n formée uniquement de '0'.

8. `AllZero`
   argument : une liste binaire l
   retourne : `t` (vrai) si tous les éléments de la liste l sont 0 et `nil` autrement.

9. `SumMod2`
   argument : deux listes binaires *l1* et *l2*
   retourne : la somme *modulo 2* des éléments respectifs de chaque liste. On suppose que les deux listes ont la même longueur. On remarque que la somme modulo 2 de deux valeurs binaires est en même temps leur différence modulo 2. **Exemple :**
   ```
   >(SumMod2 (0110) (0101))
   (0011)
   ```

3. Liste de listes

10. `PosOne` (en utilisant éventuellement la fonction `IsOne`)
    argument : une liste de listes binaires l et un entier n
    retourne : la position dans la liste l de la première sous-liste qui a un '1' à la position n. **Exemple :**
    ```
    >(PosOne ((011) (101)) 1)
    2
    ```
    et :
    ```
    >(PosOne ((011) (101)) 2)
    1
    ```

11. `SumMod2List` (en utilisant éventuellement la fonction `SumMod2`)
    argument : une liste de listes binaires l
    retourne : une liste qui a à la position i la somme, modulo 2, des éléments à la position i des sous-listes de l. On suppose que les listes binaires ont la même longueur. **Exemple :**
    ```
    >(SumMod2List (0 0 1) (0 1 1) (1 0 1) (1 1 1))
    (0 0 0 )
    ```
    et :
    ```
    >(SumMod2List (1 0 1) (0 1 1) (1 0 0))
    (0 1 0)
    ```

12. `ParityList`
    argument : une liste de listes binaires l
    retourne : `t` si la somme modulo 2 des éléments respectifs des sous-listes de l est la liste formée uniquement des 0 et `nil` autrement. On suppose que les listes binaires ont la même longueur. **Exemple :**
    ```
    >(ParityList (0 0 1) (0 1 1) (1 0 1) (1 1 1))
    t
    ```
    et :
    ```
    >(ParityList (1 0 1) (0 1 1) (1 0 0))
    nil
    ```

4. Jeux de Nim (de Conway)

On dispose sur une table plusieurs tas d'allumettes. Deux joueurs piochent, chacun à son tour, dans l'un des tas pour en retirer une ou plusieurs allumettes. Il choisissent le tas et le nombre d'allumettes qu'ils en retirent, mais la règle les oblige à piocher dans l'un des tas. Ils poursuivent ainsi leur jeu jusqu'à ce que l'un des joueurs ramasse la ou les allumettes restantes : ce joueur-là gagne la partie.

Position paire. Écrivons dans le système binaire le nombre d'allumettes dans chaque tas, et disposons ces nombres en colonnes. Si chaque colonne comporte un nombre pair de fois le chiffre '1' alors un tel état du jeu s'appelle *position paire*; autrement *position impaire*. Voir les figures 1 et 2.

On appelle *liste de parité* la liste binaire qui est la somme, modulo 2, des éléments respectifs des listes binaires représentant le nombre d'allumettes dans chaque tas. Autrement dit, si la liste de parité

n'est formée que de '0' alors il s'agit d'une position paire, sinon d'une position impaire. La position dans la figure 1 et la position finale dans la figure 2 sont paires et la position initiale dans la figure 2 est impaire.

Stratégie gagnante. Si Bob joue conte Alice, et c'est le tour de Bob et la position est impaire alors Bob a une stratégie gagnante. Pour cela il suffit qu'il identifie une ligne (*pivot*) où le premier '1' est en même position que le premier '1' dans la liste de parité (une telle ligne existe !), et en retirant des allumettes dans le tas correspondant à cette ligne Bob peut modifier la parité du jeu et rendre a Alice une position paire. Voir figure 2. Puisque les positions qui précèdent immédiatement la position finale sont celles où il ne reste plus qu'un seul tas non vide (qui est une position impaire) il s'ensuit donc que si (1) la position initiale et paire et (2) c'est Alice qui commence alors Bob a une stratégie gagnante.

13. `Coup`

argument : une liste d'entiers représentant le nombre d'allumettes dans chaque tas
retourne : (1) si la position est impaire, une liste de deux entiers représentant le numéro du tas et le nombre d'allumettes qui doivent être retirées dans ce tas pour rendre une position paire; (2) `nil` autrement. **Exemple** (voir figures 1 et 2) :

```
>(Coup (1 3 5 7))
nil
```

et :

```
>(Coup (5 3 4) )
(2 2)
```

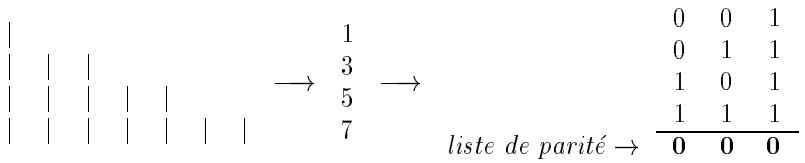

FIG. 1 – *Position paire*

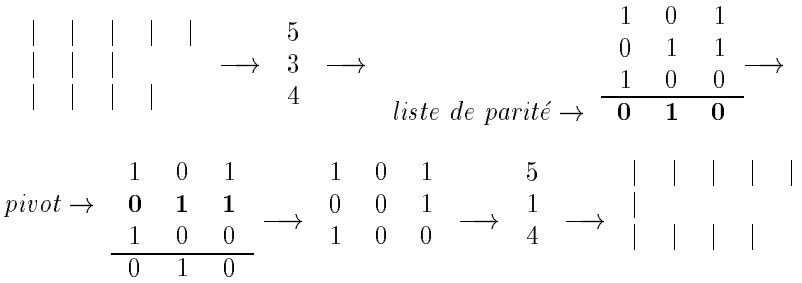

FIG. 2 – *Stratégie pour une position impaire*