premiers termes de `u`, c'est-à-dire si pour tout `i`, `i` est dans le domaine de `v` si et seulement si `i + n` est dans le domaine de `u` et $v_i = u_{i+n}$.

Une liste est dite *rationnelle* si elle n'a qu'un nombre fini de sous-listes. Par exemple la liste `4, 4, 4, 4, 4, ...` est rationnelle, ainsi que la liste `4, 5, 4, 5, 4, 5, 4, 5, ...` ou la liste `1, 2, 3, 4, 5, 4, 5, 4, 5, 4, 5, 4, 5, ....` mais pas la liste `3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, ...` Il n'est pas difficile de démontrer qu'une liste est rationnelle si elle est finie ou infinie et périodique à partir d'un certain rang, et que, de ce fait, la liste des décimales d'un réel est rationnelle si ce réel est rationnel. On peut montrer que l'ensemble des listes rationnelles est aussi une solution de l'équation `List = {null}` $\uplus$ `(int` $\times$ `List)`. C'est cet ensemble, et non l'ensemble des listes finies, que la déclaration du type `List` définit. Le nombre minimal de cellules pour construire une liste est le nombre de ses sous-listes distinctes.

Cette notion de liste rationnelle peut se généraliser à n'importe quel type récursif et on montre, de même, que les valeurs d'un tel type sont les valeurs rationnelles.

Certains programmeurs choisissent de ne jamais construire d'objets infinis. Certains langages de programmation ne permettent pas la construction de tels objets. Par exemple, si l'on ajoute au noyau fonctionnel de Java le constructeur binaire pour les listes, on ne peut pas construire de listes infinies, puisque dans une expression `new List(x,l) ;` l'objet `l` doit être déjà construit.

Il est important de savoir quelle convention on choisit : autoriser ou interdire la construction d'objets infinis, car la fonction

```
static int somme (final List l) {
 if (l == null) return 0;
 return l.hd + somme(l.tl);}
```

termine si on ne l'applique qu'à des listes finies, mais pas si on l'applique à des listes infinies.

Exercice 5.2

Écrire une fonction qui teste si une liste rationnelle est finie ou infinie.

5.2 Les types disjonctifs

Bien souvent, on veut définir un type, non comme un produit cartésien, mais comme une union disjointe de produits cartésiens. Par exemple, comme nous l'avons vu, le type `List` se définit, non comme le produit cartésien `int`