premier argument.

Exercice 6.5

Que deviennent les dessins ci-dessus quand les éléments de la liste ne sont pas d'un type scalaire, mais d'un type composite, comme le type des chaînes de caractères ?

6.2.3 L'utilisation de la récursivité

Au lieu d'utiliser une boucle `while`, on peut programmer la concaténation avec recopie du premier argument récursivement

```
static List append (List x, List y) {
 if (x == null) return y;
 return new List(x.hd,append(x.tl,y));}
```

Dans ce cas, on n'utilise pas l'affectation et la fonction est programmée dans le noyau fonctionnel de Java.

6.2.4 Les réactions chimiques et les fonctions mathématiques

Le choix entre ces différentes manières de programmer la concaténation illustre un choix fréquent quand on écrit des programmes : modifier les données ou alors les copier. Cette opposition est la même que celle qui distingue les notions de réaction chimique et de fonction mathématique. En mathématiques, si on pose `x = 32` et que l'on applique la fonction `y` $\mapsto$ `y + 1` à `x`, on obtient le résultat `33`, mais la variable `x` garde la valeur `32`, si bien que l'on peut aussi appliquer la fonction `y` $\mapsto$ `2` $\times$ `y` à `x`, et obtenir le résultat `64`. On peut également appliquer la fonction `+` aux variables `x` et `x`. La variable `x` est une corne d'abondance de `32`.

En chimie, en revanche, quand on fait réagir une molécule d'oxygène avec deux molécules d'hydrogène pour obtenir deux molécules d'eau, les molécules d'oxygène et d'hydrogène sont consommées par la réaction et on ne peut pas les utiliser ensuite pour une autre réaction. De plus, les deux molécules d'hydrogène doivent être distinctes : on ne peut pas faire réagir une molécule avec elle-même. Rien ne se perd, rien ne se crée, il n'y a pas de cornes d'abondance.

Quand on travaille avec de petites données, il est toujours préférable de les copier. Cela permet de considérer la fonction `append` comme une fonction