5.1 Les types enregistrements récursifs

5.1.1 Les listes

Au chapitre précédent, nous avons construit un type enregistrement dont les champs étaient d'un type scalaire `double`. Il est également possible de définir un type enregistrement `T` dont les champs sont eux-mêmes d'un type enregistrement, en particulier le type `T` lui-même. Par exemple, un triplet d'entiers `(a,b,c)` peut se définir comme le couple `(a,(b,c))`, et plus généralement une *liste* non vide d'entiers peut se définir comme un couple formé d'un entier, la *tête* de la liste, et d'une liste plus courte, la *queue* de la liste. Cela amène à définir le type `List` ainsi

```
class List {
 int hd;
 List tl;}
```

La tête de la liste `1, 2, 3, 4`, par exemple, est l'entier `1`. La queue de cette liste est la liste `2, 3, 4` — et non l'entier `4`.

5.1.2 La valeur `null`

Cette définition n'est pas tout à fait exacte puisque si `List` est le type des listes, le produit cartésien `int` $\times$ `List` ne contient que les listes non vides. Le type `List` n'est donc pas le produit cartésien `int` $\times$ `List`, mais l'union disjointe `{vide}` $\uplus$ `(int` $\times$ `List)`.

Cependant, comme une valeur de type `List` est ou bien `null` ou bien une référence associée dans la mémoire à un enregistrement, le type `List` que nous avons défini est égal à `{null}` $\uplus$ `(int` $\times$ `List)`, si bien que la liste vide s'identifie miraculeusement à la valeur `null`.

5.1.3 Un exemple

Observons maintenant ce qui se passe quand on exécute le programme

```
List l;
l = new List();
l.hd = 4;
l.tl = new List();
l.tl.hd = 5;
l.tl.tl = null;
```