Enfin, on définit le type `List` comme la limite de cette suite

$\texttt{List} = \bigcup_i \texttt{L}_i$

La valeur `null` est essentielle dans cette construction. La même construction avec l'équation `List = int` $\times$ `List` donnerait un type vide.

Remarquons que le type ainsi construit n'est pas l'unique solution de l'équation `List = {null}` $\uplus$ `(int` $\times$ `List)`. L'ensemble de toutes les suites finies ou infinies est également une solution. Mais la solution construite ainsi est le plus petit ensemble qui est une solution de cette équation.

Plus généralement, si l'on considère un ensemble `E`, une famille d'ensembles $\texttt{A}_1, \ldots, \texttt{A}_n$ et une famille de fonctions $\texttt{f}_1, \ldots, \texttt{f}_n$ telles que $\texttt{f}_i$ soit une injection de $\texttt{A}_i \times \texttt{E}^{m_i}$ dans `E` et les images de $\texttt{f}_i$ soient disjointes, alors on peut construire ainsi le plus petit sous-ensemble de `E` clos par les fonctions $\texttt{f}_i$. Cela nous permet donc de comprendre toutes les définitions récursives de types.

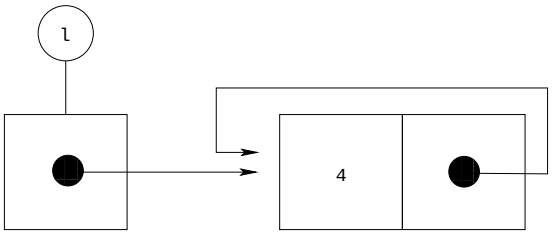
Parmi ces types récursifs, certains ont un unique champ récursif et d'autres plusieurs. On appelle *types de listes* les différents types qui ont un unique champ récursif, quel que soit le nombre de champs non récursifs, et *types d'arbres* les types qui en ont plusieurs.

5.1.5 Les valeurs infinies

Le type `List` est une solution de l'équation `List = {null}` $\uplus$ `(int` $\times$ `List)`, mais ce n'est pas exactement la plus petite solution que nous avons construite ci-dessus. En effet, ce type contient des valeurs qui ne peuvent pas se construire en un nombre fini d'étapes. Par exemple, le programme

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

construit la liste



c'est-à-dire la liste `4, 4, 4, 4, 4, ...` qui n'est pas vraiment finie.

Si `u` et `v` sont deux listes finies ou infinies, on dit que `v` est une sous-liste de `u` si et seulement s'il existe un entier `n` tel que `v` soit obtenue en effaçant les `n`