dynamique. Si on veut que deux objets `t` et `u` de la même classe `C` ne partagent pas la même fonction d'impression, il est nécessaire de définir deux sous-classes `T` et `U` de `C` qui héritent des champs de `C` mais redéfinissent le champ `imprimer`.

8.2.1 Les méthodes et les champs fonctionnels

Un objet est donc simplement un enregistrement dont certains champs sont fonctionnels. Dans un langage comme Java, où les fonctions ne sont pas des objets de première classe, on est obligé de distinguer les champs non fonctionnels des champs fonctionnels, que l'on appelle *méthodes* de l'objet.

Dans un langage dans lequel les fonctions sont des objets de première classe, comme PCF, cela n'est pas nécessaire. Les objets sont donc des enregistrements comme les autres et on peut donc déjà commencer à programmer avec des objets en utilisant l'extension de PCF avec des enregistrements définie dans ce chapitre.

Exercice 8.6 *Un programme de gestion de la billetterie d'un concert est un objet dont les champs sont*

- *une référence sur un entier : le nombre de places disponibles à l'orchestre,*
- *une référence sur un entier : le nombre de places disponibles au balcon,*
- *une fonction qui prend en argument un objet et un entier — `0` pour l'orchestre et `1` pour le balcon — et qui retourne l'entier `0` ou `1` selon que la réservation est close ou qu'il reste des places dans cette catégorie,*
- *une fonction qui prend en argument un objet et un entier — `0` pour l'orchestre et `1` pour le balcon —, qui réserve une place en décrémentant d'une unité le nombre de places disponibles dans cette catégorie et qui retourne conventionnellement la valeur `0`.*

Programmer cet objet en PCF avec des enregistrements.

La question du typage des objets, pas plus que celle des enregistrements, n'est traitée dans ce livre. On peut néanmoins remarquer que si on donne le type `A` à l'objet de l'exercice 8.6, alors `A` doit être le produit cartésien de `nat ref`, `nat ref`, `A -> nat -> nat` et `A -> nat -> nat`. On ne peut pas définir le type `A` comme `(nat ref)` × `(nat ref)` × `(A -> nat -> nat)` × `(A -> nat -> nat)`, car cette définition est circulaire. Pour définir ce type, on doit donc introduire un opérateur de point fixe sur les types.

Si `X -> Y` désigne l'espace des fonctions de `X` vers `Y` et `B` est un ensemble qui a au moins deux éléments, alors l'équation au point fixe `A = (A -> B)` n'a pas de solution. En effet, d'après le théorème de Cantor, le cardinal de l'ensemble `A -> B` est strictement supérieur à celui de `A`. L'équation au point fixe `A = (nat ref)` × `(nat ref)` × `(A -> nat -> nat)` × `(A -> nat -> nat)`, n'a pas non plus de solution. Comme dans le cas de la construction `fix` de PCF, donner un sens à cet opérateur de point fixe sur les types, dans le cadre d'une sémantique dénotationnelle, n'est donc pas trivial.