*en appel par nom si et seulement s'il a un résultat en sémantique opérationnelle à grands pas en appel par nom, et que ce résultat est le même dans les deux cas. Même question pour l'appel par valeur.*

*Cela est-il le cas pour PCF sans types ? Indication : quel est le résultat du calcul du terme* `((fun x -> x) 1) 2` *?*

## 5.3 La sémantique dénotationnelle de PCF typé

### 5.3.1 Une sémantique triviale

Nous avons dit que dans les langages fonctionnels, on cherchait à réduire la distance qui séparait la notion de programme de celle de fonction. Autrement dit, que l'on cherchait à réduire la distance entre un programme et sa sémantique dénotationnelle.

Cependant, nous avons remarqué que définir la sémantique dénotationnelle de PCF sans types était difficile du fait que les fonctions n'avaient pas de domaine de définition. Maintenant que nous avons ajouté des types à PCF, donner sa sémantique dénotationnelle est plus simple.

À chaque type, on associe un ensemble
- $[\![\texttt{nat}]\!] = \mathbb{N}$,
- $[\![\texttt{A -> B}]\!] = [\![\texttt{A}]\!] \texttt{ -> } [\![\texttt{B}]\!]$

À chaque terme `t` de type `A` on associe un élément $[\![\texttt{t}]\!]$ de $[\![\texttt{A}]\!]$. Si le terme `t` comporte des variables libres, il est nécessaire d'indiquer dans un *environnement sémantique* `e` les objets associés à ces variables.
- $[\![\texttt{x}]\!]_{\texttt{e}} = \texttt{a}$, si `e` contient le couple `x = a`,
- $[\![\texttt{fun x:A -> t}]\!]_{\texttt{e}} = \texttt{fun a:}[\![\texttt{A}]\!] \texttt{ -> } [\![\texttt{t}]\!]_{\texttt{e,x=a}}$,
- $[\![\texttt{t u}]\!]_{\texttt{e}} = [\![\texttt{t}]\!]_{\texttt{e}}\ [\![\texttt{u}]\!]_{\texttt{e}}$,
- $[\![\texttt{n}]\!]_{\texttt{e}} = \texttt{n}$,
- $[\![\texttt{t + u}]\!]_{\texttt{e}} = [\![\texttt{t}]\!]_{\texttt{e}} + [\![\texttt{u}]\!]_{\texttt{e}}$, $[\![\texttt{t - u}]\!]_{\texttt{e}} = [\![\texttt{t}]\!]_{\texttt{e}} - [\![\texttt{u}]\!]_{\texttt{e}}$,
  $[\![\texttt{t * u}]\!]_{\texttt{e}} = [\![\texttt{t}]\!]_{\texttt{e}} * [\![\texttt{u}]\!]_{\texttt{e}}$, $[\![\texttt{t / u}]\!]_{\texttt{e}} = [\![\texttt{t}]\!]_{\texttt{e}} / [\![\texttt{u}]\!]_{\texttt{e}}$,
- $[\![\texttt{ifz t then u else v}]\!]_{\texttt{e}} = [\![\texttt{u}]\!]_{\texttt{e}}$ si $[\![\texttt{t}]\!]_{\texttt{e}} = \texttt{0}$ et $[\![\texttt{v}]\!]_{\texttt{e}}$ sinon,
- $[\![\texttt{let x:A = t in u}]\!]_{\texttt{e}} = [\![\texttt{u}]\!]_{\texttt{e,x=}[\![\texttt{t}]\!]_{\texttt{e}}}$.

Tout cela est trivial : un programme est une fonction explicitement définie et sa sémantique est cette même fonction. Et cette trivialité est l'un des buts de la conception de langages fonctionnels.

On peut cependant faire deux remarques. Premièrement, la division par `0` produit une erreur en PCF et elle n'est pas définie en mathématiques. Pour que cette définition soit correcte, il faudrait ajouter une valeur `erreur` à chaque ensemble $[\![\texttt{A}]\!]$ et adapter un peu la définition ci-dessus. Seconde remarque : dans cette définition, nous avons oublié la construction `fix`.

### 5.3.2 La terminaison

La construction `fix` est la seule dont la sémantique dénotationnelle soit intéressante, car c'est la seule qui s'éloigne de la pratique mathématique, en