Exercice 3.11 *Que deviennent ces règles de la sémantique opérationnelle à grands pas avec des valeurs rationnelles si on remplace les variables par leur indice de De Bruijn — voir la section 3.3 ?*

Exercice 3.12 *Peut-on utiliser cette technique des valeurs rationnelles pour concevoir un interpréteur pour PCF en entier, c'est-à-dire dans le cas où on peut définir par point fixe non seulement des fonctions, mais des objets quelconques ? Indication : quelle serait la représentation rationnelle de la valeur du terme* `fix x x` *?*

Pour conclure cette section, nous avons vu que, dès que la variable `x` a une occurrence dans le terme `t`, la règle de réduction `fix x t` $\longrightarrow$ `(fix x t/x)t` peut s'appliquer à l'infini au terme `fix x t`, puisque le terme `(fix x t/x)t` contient le terme `fix x t` comme sous-terme. Cela correspond au fait que dans une définition récursive `f = G(f)` en remplaçant `f` par `G(f)` à l'infini on construit le programme infini `f = G(G(G(...)))`. Cela traduit l'intuition que les programmes récursifs sont des programmes infinis. Par exemple, le terme `fact` pourrait s'écrire `fun x -> ifz x then 1 else x * (ifz x - 1 then 1 else (x - 1) * (ifz x - 2 then 1 else (x - 2) * ⋯))`. Il faut donc opérer ce remplacement uniquement à la demande : de manière paresseuse.

Nous avons vu plusieurs manières d'exprimer cela dans la sémantique de PCF — et finalement dans le code d'un interpréteur de PCF — : substituer effectivement `x` par `fix x t` et ne pas toucher à ce radical dès qu'il se trouve sous un `fun` ou un `ifz`, stocker ce radical sous forme d'un glaçon ou d'une fermeture récursive de l'environnement et ne dégeler le glaçon qu'à la demande. Représenter le terme `f = G(G(G(...)))` par un arbre rationnel et ne parcourir cet arbre qu'à la demande. Une dernière méthode serait d'utiliser le codage du `fix` de l'exercice 2.10 et de ne réduire ce terme, ce qui mène à dupliquer un sous-terme, uniquement lorsque c'est nécessaire.

Exercice 3.13 (Une extension de PCF avec des couples) *On étend PCF en ajoutant les constructions suivantes :* `t,u` *qui est le couple dont la première composante est* `t` *et la seconde est* `u`*,* `fst t` *et* `snd t` *qui sont respectivement la première et la seconde composante du couple* `t`*. Donner les règles de sémantique opérationnelle à petits pas et à grands pas de cette extension de PCF. Écrire un interpréteur pour cette extension de PCF.*

Exercice 3.14 (Une extension de PCF avec des listes) *On étend PCF en ajoutant les constructions suivantes :* `nil` *qui est la liste vide,* `cons n l` *qui est la liste dont le premier élément est l'entier* `n` *et le reste la liste* `l`*,* `ifnil t then u else v` *qui teste si une liste est vide ou non,* `hd l` *qui est le premier élément de la liste* `l` *et* `tl l` *qui est la liste* `l` *privée de son premier élément. Donner les règles de sémantique opérationnelle à petits pas et à grands pas de cette extension de PCF. Écrire un interpréteur pour cette extension de PCF. Programmer un algorithme de tri sur ces listes.*