peut se formuler en disant que pour calculer une approximation à `n` *termes de* `f x` *il suffit de connaître une certaine approximation de* `x`.

Soit `u` *une suite entière et* `U` *l'élément de* $[\![$`nat -> nat`$]\!]$ *qui associe* $\bot$ *à* $\bot$ *et* u_i *à* `i`.

Soit `V` *la suite à valeur dans* $[\![$`nat -> nat`$]\!]$

$$\begin{array}{c}
[\bot \mapsto \bot,\ 0 \mapsto \bot,\ 1 \mapsto \bot,\ 2 \mapsto \bot,\ 3 \mapsto \bot,\ \ldots],\\
[\bot \mapsto \bot,\ 0 \mapsto u_0,\ 1 \mapsto \bot,\ 2 \mapsto \bot,\ 3 \mapsto \bot,\ \ldots],\\
[\bot \mapsto \bot,\ 0 \mapsto u_0\ ,\ 1 \mapsto u_1,\ 2 \mapsto \bot,\ 3 \mapsto \bot,\ \ldots],\\
[\bot \mapsto \bot,\ 0 \mapsto u_0\ ,\ 1 \mapsto u_1,\ 2 \mapsto u_2,\ 3 \mapsto \bot,\ \ldots],\\
\ldots
\end{array}$$

Montrer que la suite `V` *converge vers* `U`. *Soit* `F` *une fonction continue de* $[\![$`nat -> nat`$]\!]$ *dans lui-même. Montrer que la suite* `F` V_i *converge vers* `F U`. *Montrer que la suite* `F` V_i `p` *converge vers* `F U p`. *Montrer qu'il existe un entier* `k` *tel que* `F` V_k `p = F U p`. *Montrer que pour calculer* `F U p`, *il suffit de connaître les* `k` *premiers termes de* `U`. *Montrer que pour calculer les* `n` *premiers termes de* `F U` *il suffit de connaître un nombre fini de termes de* `U`.

La fonction qui à une suite `u` *associe* `0` *si* `u` *est identiquement nulle et* `1` *sinon est-elle continue ? Est-elle programmable en PCF ?*

On pourra enfin remarquer que dans ces deux exemples, les approximations — nombres décimaux ou suites finies — contiennent une quantité finie d'information alors que les objets approximés — nombres réels ou suite infinies — contiennent une quantité infinie d'information.

Exercice 5.13 *(Le système T de Gödel) Pour éviter que certains programmes bouclent, on peut remplacer la construction* `fix` *par une construction* `rec`, *qui permet de définir des fonctions par récurrence. On obtient alors un langage dans lequel tous les programmes terminent, mais qui n'est plus complet au sens de Turing. Il faut cependant une certaine expertise en logique pour construire un exemple de programme qui n'est pas exprimable dans ce langage.*

La fonction `f` *définie par* `f 0 = a` *et* `f (n + 1) = g n (f n)` *est notée* `rec a g`. *Les règles de sémantique opérationnelle à petits pas associées à cette construction sont*

$$\begin{array}{c}
\texttt{rec a g 0} \longrightarrow \texttt{a}\\
\texttt{rec a g n} \longrightarrow \texttt{g (n - 1) (rec a g (n - 1))}
\end{array}$$

si `n` *est une constante entière différente de* `0`.

Programmer la factorielle dans ce langage. Donner des règles de typage pour la construction `rec`. *Donner la sémantique dénotationnelle de ce langage.*