$$\frac{\texttt{e} \vdash \texttt{u} \hookrightarrow \texttt{W} \qquad \texttt{e} \vdash \texttt{t} \hookrightarrow \langle \texttt{t'}, \texttt{e'} \rangle \qquad (\texttt{e'}, \texttt{W}) \vdash \texttt{t'} \hookrightarrow \texttt{V}}{\texttt{e} \vdash \texttt{t u} \hookrightarrow \texttt{V}}$$

$$\frac{}{\texttt{e} \vdash \texttt{fun \_ -> t} \hookrightarrow \langle \texttt{t}, \texttt{e} \rangle}$$

$$\frac{}{\texttt{e} \vdash \texttt{n} \hookrightarrow \texttt{n}}$$

$$\frac{\texttt{e} \vdash \texttt{u} \hookrightarrow \texttt{q} \qquad \texttt{e} \vdash \texttt{t} \hookrightarrow \texttt{p}}{\texttt{e} \vdash \texttt{t} \otimes \texttt{u} \hookrightarrow \texttt{n}} \text{ si } \texttt{p} \otimes \texttt{q} = \texttt{n}$$

$$\frac{\texttt{e} \vdash \texttt{t} \hookrightarrow \texttt{0} \qquad \texttt{e} \vdash \texttt{u} \hookrightarrow \texttt{V}}{\texttt{e} \vdash \texttt{ifz t then u else v} \hookrightarrow \texttt{V}}$$

$$\frac{\texttt{e} \vdash \texttt{t} \hookrightarrow \texttt{n} \qquad \texttt{e} \vdash \texttt{v} \hookrightarrow \texttt{V}}{\texttt{e} \vdash \texttt{ifz t then u else v} \hookrightarrow \texttt{V}} \quad \begin{array}{l}\text{si n constante} \\ \text{entière} \neq \texttt{0}\end{array}$$

$$\frac{(\texttt{e}, \langle \texttt{fix \_ t}, \texttt{e} \rangle) \vdash \texttt{t} \hookrightarrow \texttt{V}}{\texttt{e} \vdash \texttt{fix \_ t} \hookrightarrow \texttt{V}}$$

$$\frac{\texttt{e} \vdash \texttt{t} \hookrightarrow \texttt{W} \qquad (\texttt{e}, \texttt{W}) \vdash \texttt{u} \hookrightarrow \texttt{V}}{\texttt{e} \vdash \texttt{let \_ = t in u} \hookrightarrow \texttt{V}}$$

**Exercice 3.6** *Écrire un programme qui remplace chaque variable par son indice de De Bruijn. Écrire un interpréteur pour ce langage.*

**Exercice 3.7** *Écrire les règles de la sémantique opérationnelle à grands pas de l'interprétation en appel par nom avec des indices de De Bruijn.*

Nous verrons l'avantage que l'on tire de cette élimination des variables quand nous aborderons la compilation au prochain chapitre.

En attendant, nous pouvons remarquer que deux termes ont des traductions de De Bruijn identiques si et seulement s'ils sont alphabétiquement équivalents, ce qui nous fournit une nouvelle définition de l'équivalence alphabétique. Cette élimination des variables et leur remplacement par un indice qui indique l'endroit où elles sont liées est un point de vue très radical sur le mutisme des variables liées.

## 3.4 La construction de fonctions par point fixe

Dans la plupart des langages de programmation, on ne peut définir récursivement que des fonctions. Cela revient à ne pouvoir appliquer une construction `fix` qu'à un terme de la forme `fun`, ou encore à remplacer le symbole `fix` par un symbole `fixfun f x -> t` qui lie deux variables dans son argument. La règle de la sémantique opérationnelle à grands pas en appel par valeur de cette construction se déduit de celle de la construction `fix` et de la construction `fun`

$$\frac{}{\texttt{e} \vdash \texttt{fixfun f x -> t} \hookrightarrow \langle \texttt{x}, \texttt{t}, (\texttt{e}, \texttt{f} = \langle \texttt{fixfun f x -> t}, \texttt{e} \rangle) \rangle}$$

Dans ce cas, on peut proposer des variantes plus simples pour les règles de la sémantique opérationnelle à grands pas d'un interpréteur en appel par valeur.