Maintenant, voyons comment un interpréteur qui commande une machine abstraite interprète un tel terme : pour interpréter le terme `t u`, la machine abstraite commence par interpréter le terme `u` et met le résultat sur le sommet de la pile. Puis elle interprète le terme `t`, ce qui donne une fermeture ⟨`f, x, t', e'`⟩, elle met alors dans le registre environnement l'environnement `e', f =` ⟨`f, x, t', e'`⟩`, x = W`, où `W` est la valeur qui est au sommet de la pile, et elle retire cette valeur du sommet de la pile. Elle interprète enfin le terme `t'`. Pour que le contenu du registre environnement soit le même à la fin de ces opérations qu'au début, il est nécessaire de l'empiler avant le début de ces opérations et de le dépiler à la fin.

Passons maintenant à la compilation d'un tel terme. L'interprétation du terme `u` est remplacée par l'exécution de la suite d'instructions `|u|`, de même l'interprétation du terme `t` est remplacée par l'exécution de la suite d'instructions `|t|`. L'interprétation du terme `t'` doit être remplacée par l'exécution de la suite d'instructions `|t'|`. La difficulté est ici que le terme `t'` n'est pas un sous-terme de `t u` mais qu'il est fourni par la fermeture, résultat de l'interprétation de `t`. Il faut alors modifier la notion de fermeture et dans ⟨`f, x, t, e`⟩ remplacer le terme `t` par une suite d'instructions `i`. De ce fait, les termes de la forme `fun x -> t` et `fixfun f x -> t` ne doivent pas se compiler, comme nous l'avons dit plus haut, en **Mkclos**`(f,x,t)`, mais en **Mkclos**`(f,x,|t|)` qui fabrique la fermeture ⟨`f, x, |t|, e`⟩ où `e` est le contenu du registre environnement.

Ensuite, nous avons besoin d'ajouter une instruction **Apply** à notre machine qui prend une fermeture ⟨`f, x, i, e`⟩ dans l'accumulateur, met l'environnement `e, f =` ⟨`f, x, i, e`⟩`, x = W`, où `W` est le sommet de la pile, dans le registre environnement, retire le sommet de la pile et ajoute la suite d'instructions `i` au registre code.

Le terme `t u` se compile alors en la suite d'instructions **Pushenv**, `|u|`, **Push**, `|t|`, **Apply**, **Popenv**.

Pour résumer, notre machine abstraite contient les instructions **Ldi** `n`, **Push**, **Add**, **Extend** `x`, **Search** `x`, **Pushenv**, **Popenv**, **Mkclos**`(f,x,i)` et **Apply**. Il ne nous reste qu'à ajouter les opérations arithmétiques **Sub**, **Mult**, **Div** et le test **Test**`(i,j)` qui servent à compiler les constructions `-`, `*`, `/` et `ifz` pour qu'elle soit complète.

4.3.4 L'utilisation des indices de De Bruijn

Cette machine peut être simplifiée en utilisant des indices de De Bruijn — voir la section 3.3. L'instruction **Search** `x` est produite par la compilation des variables de PCF, or nous avons vu que l'on peut déterminer statiquement les indices de De Bruijn de chaque occurrence d'une variable d'un tel terme. Ainsi, on peut compiler une variable, non en l'instruction **Search** `x` où `x` est une variable, mais en l'instruction **Search** `n` où `n` est un entier.

Le calcul des indices de De Bruijn peut se faire en même temps que la compilation, cela demande simplement de compiler un terme dans un environnement de variables, et de compiler la variable `x` dans l'environnement `e` par l'instruction **Search** `n`, où `n` est la place de la variable `x` dans l'environnement `e`, en