$$\frac{\texttt{(e, x : A)} \vdash \texttt{t : B}}{\texttt{e} \vdash \texttt{fun x:A -> t : A -> B}}$$

$$\frac{}{\texttt{e} \vdash \texttt{n : nat}}$$

$$\frac{\texttt{e} \vdash \texttt{u : nat} \quad \texttt{e} \vdash \texttt{t : nat}}{\texttt{e} \vdash \texttt{t} \otimes \texttt{u : nat}}$$

$$\frac{\texttt{e} \vdash \texttt{t : nat} \quad \texttt{e} \vdash \texttt{u : A} \quad \texttt{e} \vdash \texttt{v : A}}{\texttt{e} \vdash \texttt{ifz t then u else v : A}}$$

$$\frac{\texttt{(e, x : A)} \vdash \texttt{t : A}}{\texttt{e} \vdash \texttt{fix x:A t : A}}$$

$$\frac{\texttt{e} \vdash \texttt{t : A} \quad \texttt{(e, x : A)} \vdash \texttt{u : B}}{\texttt{e} \vdash \texttt{let x:A = t in u : B}}$$

Bien entendu, dans la première règle, seule la déclaration de `x` la plus à droite doit être prise en compte, car les autres sont cachées.

Le langage contient des variables de toutes les sortes, en particulier des variables de type, que l'on note avec une lettre majuscule. Cependant, comme il n'y a pas de symbole permettant de lier une variable de type, un terme clos ne peut pas contenir de variables de type. De plus, si un terme clos `t` a le type `A` dans l'environnement vide, alors le type `A` est lui-même clos. De ce fait, les variables de type ne sont pas vraiment utilisées. Elles le seront, en revanche, dès le prochain chapitre.

Soit `e` un environnement et `t` un terme. En faisant un raisonnement par récurrence structurelle sur `t`, on montre que le terme `t` a au plus un type dans l'environnement `e`.

En se laissant guider par les règles de typage, on peut construire un algorithme de *vérification de types*, c'est-à-dire un algorithme qui indique si le terme `t` a un type ou non dans l'environnement `e` et donne ce type quand il existe. Cet algorithme consiste à typer récursivement les sous-termes immédiats du terme dont on calcule le type et à calculer ce type en fonction du type de ces sous-termes.

Exercice 5.1 *Écrire un vérificateur de types pour PCF.*

La réduction reste confluente sur les termes typés. En revanche, les types amènent une propriété nouvelle : la terminaison de tous les termes ne contenant pas la construction `fix` — théorème de Tait. Il n'est donc pas possible de construire un terme comme `(fun x -> (x x)) (fun x -> (x x))` qui boucle, bien qu'il ne contienne pas la construction `fix`.

Exercice 5.2 *Que deviennent ces règles de typage si on remplace les variables par leur indice de De Bruijn — voir la section 3.3 ?*

Exercice 5.3 *On étend PCF avec les constructions pour les couples de l'exercice 3.13 et on ajoute au langage des types un symbole $\times$ pour le produit cartésien de deux types. Donner les règles de typage de cette extension de PCF. Écrire un vérificateur de types pour cette extension de PCF.*