Devoir maison de Programmation 2

Manon BLANC

question 1. Les règles de typage de *succ* et *pred* sont les suivantes :

$$\frac{\Gamma \vdash n : nat}{\Gamma \vdash \ succ(n) : nat} \qquad \frac{\Gamma \vdash n : nat}{\Gamma \vdash \ pred(n) : nat}$$

Les sémantiques opérationnelles respectives des ces opérateurs sont les suivantes :

$$\frac{t \hookrightarrow 0}{succ(t) \hookrightarrow succ(0)} \qquad \frac{t \hookrightarrow u}{succ(t) \hookrightarrow succ(u)}$$

$$\frac{t \hookrightarrow 0}{pred(t) \hookrightarrow 0} \qquad \frac{t \hookrightarrow u}{pred(t) \hookrightarrow pred(u)}$$

question 2. On implémente d'abord les règles de typage des opérateurs $plus$ et $fois$:

$$\frac{\Gamma \vdash n_1 : nat \qquad \Gamma \vdash n_2 : nat}{\Gamma \vdash n_1 + n_2 : nat} \qquad \frac{\Gamma \vdash n_1 : nat \qquad \Gamma \vdash n_2 : nat}{\Gamma \vdash n_1.n_2 : nat}$$

D'où des implémentations en Mini-ML :

$plus$:

$$\lambda n_1^A.\lambda n_2^A.(if\ n_2 = 0\ then\ n_1\ else\ plus(succ(n_1), pred(n_2)))$$

$fois$:

$$\lambda n_1^A.\lambda n_2^A.(if\ (n_1 = 0 || n_2 = 0)\ then\ 0\ else\ if\ n_1 = 1\ then\ n_2\ else\ plus(fois(pred(n_1), n_2), n_2))$$

question 3.

$$f : \lambda x^A.(if\ x = 0\ then\ 1\ else\ fois(x, f(pred(x))))$$

On donne l'arbre de dérivation du jugement $f\ 1 \hookrightarrow 1$. On pose $succ(0) = 1$ et :

$$\frac{t \hookrightarrow 0}{t \hookrightarrow pred(succ(0))}$$

D'où :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{1 \hookrightarrow 1}}{if\ (1 = 1)\ then\ 1\ else\ plus(fois(pred(n_1), n_2), n_2) \hookrightarrow 1}}{if\ (1 = 0 || 1 = 0)\ then\ 0\ else\ if\ (1 = 1)\ then\ 1\ else\ plus(fois(pred(n_1), n_2), n_2) \hookrightarrow 1}}{fois(1, 1) \hookrightarrow 1}}{fois(1, if\ (0 = 0)\ then\ 1\ else\ fois(1, f(pred(0))) \hookrightarrow 1}}{fois(1, f(0)) \hookrightarrow 1}}{if\ (1 = 0)\ then\ 1else\ fois(1, f(0)) \hookrightarrow 1}}{if\ (1 = 0)\ then\ 1\ else\ \ fois(1, f(pred(1))) \hookrightarrow 1}$$

question 4. On utilise la commande fst, qui est déjà implémentée en Mini-ML, pour extraire la première composante du couple.

question 5. Le typage des expressions permet d'effectuer des opérations légales et bien typées. Cela permet aussi d'obtenir le résultat voulu (et non un résultat incohérent si le programme termine).

question 6. Supprimée

question 7. Si on a pu donner dans les premières questions des règles de typage et une sémantique à petit pas de *pred*, il n'existe cependant pas d'implémentation en Mini-ML de cette fonction. En effet, cela supposerait que nous avons à notre disposition des opérateurs d'addition et/ou de soustraction et de comparaison autres que l'égalité, ce qui n'est pas le cas ici.

question 8.

question 9. Règle concernée :

$$\frac{t\textbf{value} \qquad u\ \textbf{value} \qquad t \not\equiv u}{(t = u) \hookrightarrow false}$$

$\not\equiv$ signifie ici que t et u ont des valeurs différentes ou des types différents (?). On a besoin de ce symbole car sinon, on remplacerait $t \not\equiv u$ par ce qu'on veut démontrer, c'est-à-dire $(t = u) \hookrightarrow false$, et donc obtenir un arbre de dérivation infini, ce qui est absurde.

question 10. Supprimée.

question 11. Le ML étant un langage fonctionnel, on a besoin de pouvoir définir des fonctions. On doit donc implémenter une structure de donnée qui prend en argument des paramètres, un corps et renvoie un résultat, et ceci ne peut pas être codé directement à partir des opérations données.

question 12.

question 13. La règle de typage de && est la suivante :

$$\frac{\Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash u\ \&\&\ v : \textbf{bool}}$$

La règle de typage de + est similaire à celle de *plus*, définie en question 2.

question 14. L'instruction *let rec... in...* peut inclure des programmes récursifs qui ne vont pas terminer, et donc des arbres de syntaxe abstraite infinis. L'exclure permet d'éviter ce phénomène.

question 15. On utilise ici une option pour signifier qu'il peut éventuellement y avoir un symbole de type ty, mais ce n'est pas nécessaire pour la compilation. L'intérêt d'utiliser le type $ty\ option\ ref$ et qu'on manipule directement des pointeurs, et des objets mutables, ce qui peut-être utile si on envisage un α-renommage pas exemple.

question 16. Partiellement faite, je n'ai pas réussi à la finir, le code qui a posé problème est en commentaire. Le typage d'une fonction n'est donc pas implémenté...

question 17. Faite.

question 18. Faite.

question 19.

question 20. ;