Preuves Constructives
Corrigé de l'Examen
DEA de Programmation

13 Décembre 2000

1 Echauffement

L'étudiant Géo Codetout a construit en théorie de Martin-Löf un terme clos t de type suivant :

$$t : \Pi x : \mathrm{nat}.\Sigma y : nat.(x = S(y) \ + \ x = 0).$$

On a :

$$\begin{array}{rcl} \pi_1(t\ S(S(S(0)))) & \triangleright^* & S(S(0)) \\ \pi_1(t\ S(0)) & \triangleright^* & 0 \end{array}$$

Le terme $\pi_1(t\ 0)$ peut se réduire vers n'importe quel terme clos normal de type nat ; sa valeur n'est pas spécifiée par le type de t. Les termes clos normaux de types nat sont tous ceux de la forme $S(\ldots S(O)\ldots)$.

2 Listes en déduction modulo

a) Les listes sont obtenues à partir des constantes suivantes :
- $[]$ d'arité $(), \mathrm{list}$
- $\_ :: \_$ d'arité $(\mathrm{nat}, \mathrm{list}), \mathrm{list}$
- $\_@\_$ d'arité $(\mathrm{list}, \mathrm{list}), \mathrm{list}$
- $\mathrm{rev}(\_)$ d'arité $(\mathrm{list}), \mathrm{list}$

b) On donne une preuve que la fonction $\mathrm{rev}(\_)$ est involutive. On remarque d'abord que quels que soient les termes a et l, on a : $\mathrm{rev}(\mathrm{rev}(a :: l)) \triangleright^* a :: (\mathrm{rev}(\mathrm{rev}(l)))$. La dérivation est alors la suivante. L'inférence $\textsc{Conv}$ ne correspond pas à une règle a proprement parler, mais sert juste à indiquer à quel endroit intervient cette réécriture.

$$\dfrac{\dfrac{}{\vdash \mathrm{rev}(\mathrm{rev}([])) = []}\ (\textsc{Refl}) \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{rev}(\mathrm{rev}(l)) = l \vdash \mathrm{rev}(\mathrm{rev}(l)) = l}\textsc{Ax} \quad \dfrac{}{\mathrm{rev}(\mathrm{rev}(l)) = l \vdash a :: l = a :: l}\textsc{Refl}}{\mathrm{rev}(\mathrm{rev}(l)) = l \vdash a :: \mathrm{rev}(\mathrm{rev}(l)) = a :: l}\textsc{Leib}}{\mathrm{rev}(\mathrm{rev}(l)) = l \vdash \mathrm{rev}(\mathrm{rev}(a :: l)) = a :: l}\textsc{Conv}}{\vdash \mathrm{rev}(\mathrm{rev}(l)) = l \Rightarrow \mathrm{rev}(\mathrm{rev}(a :: l)) = a :: l}\to\text{-I}}{\vdash \forall a.\forall l.\mathrm{rev}(\mathrm{rev}(l)) = l \Rightarrow \mathrm{rev}(\mathrm{rev}(a :: l)) = a :: l}\forall\text{-I}}{\vdash \forall_{list} l.\mathrm{rev}(\mathrm{rev}(l)) = l}\textsc{RecL}$$

c) On ajoute à la théorie un prédicat discr sur les listes, ainsi que les deux règles de réductions suivantes :

$$\mathsf{discr}([]) \ \triangleright \ 0 = 0 \tag{1}$$

$$\mathsf{discr}(a :: l) \ \triangleright \ \bot \tag{2}$$

Ce prédicat permet d'introduire une discrimination entre les listes vides et non-vides[1]. On peut considérer que la proposition $\mathsf{discr}(l)$ signifie "l est vide".

Puisque ce prédicat est vérifié par la liste vide et facilement niable pour les listes formées par cons, il permet aisément de nier la propriété de Leibniz. Un preuve de $[] = x :: l \Rightarrow \bot$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{[] = x :: l \vdash 0 = 0}\textsc{Refl}}{[] = x :: l \vdash \mathsf{discr}([])}\textsc{Conv} \qquad \dfrac{}{[] = x :: l \vdash [] = x :: l}\textsc{Ax}}{\dfrac{[] = x :: l \vdash \mathsf{discr}(x :: l)}{[] = x :: l \vdash \bot}\textsc{Conv}}\textsc{Leib}}{\vdash [] = x :: l \Rightarrow \bot}\Rightarrow\!=\!\mathrm{I}$$

Encore une fois, les $\textsc{Conv}$ servent à indiquer où l'on se sert de la conversion.

d) Assez clairement, $\mathrm{rev}(l)$ est vide si et seulement si l l'est. On propose donc :

$$\mathsf{discr}(\mathrm{rev}(l)) \triangleright \mathsf{discr}(l).$$

Par contre, la liste $l@m$ est vide si et seulement si l et m sont vides toutes les deux. Il faudrait eventuellement choisir une règle comme $\mathsf{discr}(l@m) \triangleright \mathsf{discr}(l) \wedge \mathsf{discr}(m)$. Mais cela casserait la confluence du système de réécriture. Par exemple, on aurait à la fois :

$$\begin{array}{lcl} \mathsf{discr}([]@[]) & \triangleright & \mathsf{discr}([]) \wedge \mathsf{discr}([]) \;\triangleright\; \bot \wedge \bot \\ \mathsf{discr}([]@[]) & \triangleright & \mathsf{discr}([]) \;\triangleright\; \bot \end{array}$$

Or même si $\bot$ et $\bot \wedge \bot$ sont logiquement équivalentes, leurs preuves ont des structures différentes.

3 Listes dans la théorie de Martin-Löf

Ce sont typiquement des fonctions qui se définissent bien par récurrence structurelle. Les concaténation ressemble à l'addition des nombres unaires.

Les définitions qui me semblent les plus simples sont :

$$\begin{array}{rcl} @ & \equiv & \lambda l : \mathrm{list}.\lambda m : \mathrm{list}.(\mathrm{RecL}_{\lambda\_:\mathrm{list}.\mathrm{list}}\ m\ \lambda a : \mathrm{nat}.\lambda l' : \mathrm{list}.\lambda r : \mathrm{list}.a :: r\ l) \\ \mathrm{rev}() & \equiv & \lambda l : liste.(\mathrm{RecL}_{\lambda\_:\mathrm{list}.\mathrm{list}}\ []\ \lambda a : \mathrm{nat}.\lambda l' : \mathrm{list}.\lambda r : \mathrm{list}.(r@(a :: []))\ l) \end{array}$$

b) Ces deux fonctions récurent sur leur premier argument. Elles vérifient donc les réductions suivantes :

$$\begin{array}{rcl} []@l & \triangleright & l \\ (a :: l)@m & \triangleright & a :: (l@m) \\ \mathrm{rev}([]) & \triangleright & [] \\ \mathrm{rev}(a :: l) & \triangleright & \mathrm{rev}(l)@[a] \end{array}$$

c) Les autres réductions de la partie 1 correspondent à des propriétés extentionnelles des fonctions. Ce sont des égalités qui doivent être prouvées par récurrence :

$\Pi l : \mathrm{list}.l@[] = l$ par récurrence sur l

$\Pi l, m : \mathrm{list}.(l@m)@n = l@(m@n)$ par récurrence sur l

$\Pi l, m : \mathrm{list}.\mathrm{rev}(l@m) = \mathrm{rev}(m)@\mathrm{rev}(l)$ par réccurrence sur l.

[1] C'est une technique assez standard en Coq et les théories des types plus évoluées.

On peut indiquer comment les réductions et les lemmes précédants sont utilisés dans le dernier lemme, dans le cas où $l = a :: l'$:

$$\begin{array}{rcl}
\text{rev}((a :: l')@m) & \rhd & \text{rev}(a :: (l'@m)) \\
& \rhd & \text{rev}(l'@m)@[a] \\
& = & (\text{rev}(m)@\text{rev}(l))@[a] \quad \text{(par H.R.)} \\
& = & \text{rev}(m)@(\text{rev}(l')@[a]) \quad \text{(lemme précédant)} \\
& = & \text{rev}(m)@\text{rev}(a :: l') \quad \text{(par } \beta\text{-expansion)}
\end{array}$$

d) Ensuite c'est pareil. On montre l'involutivité de rev() par récurrence sur son argument. Le cas de la liste vide est trivial. Le cas par récurrence utilise les étapes suivantes :

$$\begin{array}{rcl}
\text{rev}(\text{rev}(a :: l)) & \rhd & \text{rev}(\text{rev}(l)@[a]) \\
& = & \text{rev}([a])@\text{rev}(\text{rev}(l)) \qquad \text{par le lemme précédent} \\
& \rhd & [a]@\text{rev}(\text{rev}(l)) \\
& \rhd & a :: \text{rev}(\text{rev}(l)) \\
& = & a :: l \quad \text{par H. R.}
\end{array}$$

4 Imprédicativité

Je me suis en peu embrouillé dans cette question rajoutée sans doute un peu rapidement. Je voulais vous faire construire prédicat par codage imprédicatif. En fait il y a une solution triviale...mais ça n'a fait que rendre la question facile.

La solution triviale est :

$$div3 \equiv \lambda n : \text{nat}.(\exists_{\text{nat}} \lambda q : \text{nat}.n =_{\text{nat}} (\text{mult}\ (S(S(S\ O)))\ q))$$

La solution que j'avais en tête était :

$$div3 \equiv \lambda n : \text{nat}.(\forall_{\text{nat}\to o}\ \lambda P : \text{nat} \to o.(P\ O) \wedge (\forall_{\text{nat}}\ \lambda m : \text{nat}.(P\ m) \Rightarrow (P\ (S(S(S\ m))))) \Rightarrow (P\ n))$$

c'est-à-dire le plus petit prédicat vérifié par 0 et clos par l'opération "plus 3".