Programmer en temps élémentaire

Marc Lasson

26 février 2009

Résumé

Si à l'origine ce document aurait dû traiter du temps polynomial, ses ambitions ont été revues à la baisse : on y traitera exclusivement du temps élémentaire (les tours d'exponentiels). Sa présentation a l'avantage de déployer le même genre d'idées tout en limitant les complications techniques. Nous commencerons par une introduction à la logique linéaire élémentaire puis nous introduirons une version intuitionniste de celle-ci et nous verrons comment y programmer.

1 Logique Linéaire Élémentaire

Nous allons étudier dans cette section le fragment de la logique linéaire dans lequel on a interdit la déréliction et restreint la promotion et dont voici, sans plus attendre, les règles.

$$\frac{}{\vdash A^\perp, A}\ \text{IDENTITÉ}$$

$$\frac{}{\vdash 1}\ \text{UN} \qquad \frac{}{\vdash \top, \Gamma}\ \text{VRAI} \qquad \frac{\vdash \Gamma}{\vdash \bot, \Gamma}\ \text{FAUX}$$

$$\frac{\vdash A, \Gamma}{\vdash A \oplus B, \Gamma}\ \text{GAUCHE} \qquad \frac{\vdash B, \Gamma}{\vdash A \oplus B, \Gamma}\ \text{DROITE}$$

$$\frac{\vdash A, \Gamma \qquad \vdash B, \Gamma}{\vdash A \& B, \Gamma}\ \text{AVEC} \qquad \frac{\vdash A, \Gamma \qquad \vdash B, \Delta}{\vdash A \otimes B, \Gamma, \Delta}\ \text{FOIS} \qquad \frac{\vdash A, B, \Gamma}{\vdash A \parr B, \Gamma}\ \text{PAR}$$

$$\frac{\vdash A, \Gamma}{\vdash !A, ?\Gamma}\ \text{PROMOTION} \qquad \frac{\vdash ?A, ?A, \Gamma}{\vdash ?A, \Gamma}\ \text{CONTRACTION} \qquad \frac{\vdash \Gamma}{\vdash ?A, \Gamma}\ \text{AFFAIBLISSENT}$$

$$\frac{\vdash \Gamma, C \qquad \vdash C^\perp, \Delta}{\vdash \Gamma, \Delta}\ \text{COUPURE}$$

$$\left(\frac{\vdash \Gamma, A}{\vdash \Gamma, \forall X\, A}\ \text{POUR TOUT}(*) \qquad \frac{\vdash \Gamma, A[B/X]}{\vdash \Gamma, \exists X\, A}\ \text{IL EXISTE} \qquad (*)\ \text{X n'est pas libre dans } \Gamma \right)$$

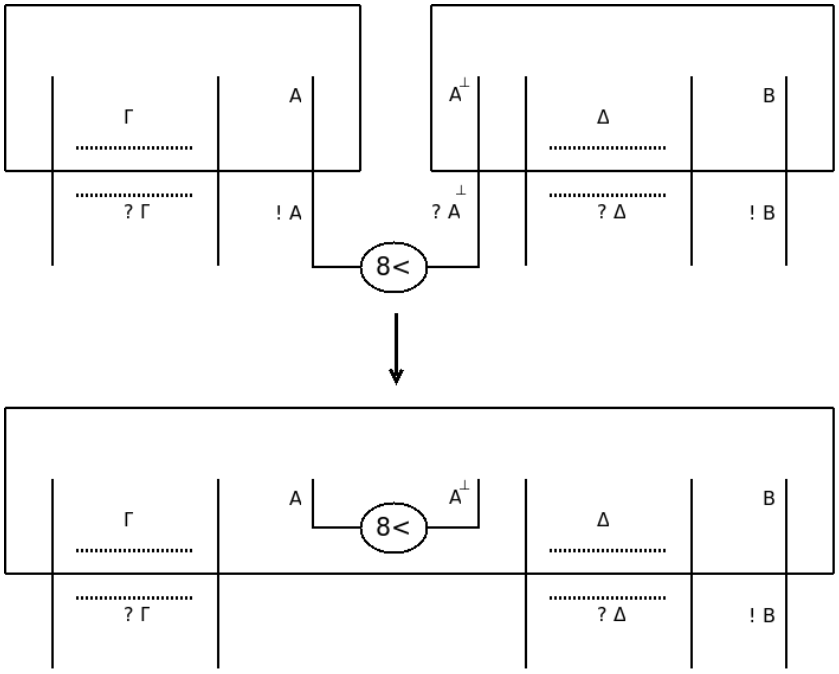
On cherche à montrer que dans ce système on peut éliminer les coupures tout en contrôlant la taille des preuves. L'idée n'est pas de limiter la duplication des « ! » mais

plutôt de surveiller leur utilisation. Ainsi, l'utilisation (comprendre introduction à gauche) des hypothèses qui portent un « ! » est régie par notre nouvelle règle, appelée *promotion fonctorielle*, et qui introduit également un « ! » à droite. Le « ! » devient un marqueur de la complexité, il est le prix à payer pour utiliser des hypothèses duplicables.

La promotion fonctorielle s'élimine de la façon suivante :

$$\dfrac{\dfrac{\vdash \Gamma, A}{\vdash ?\Gamma, !A} \qquad \dfrac{\vdash A^{\perp}, \Delta, B}{\vdash ?A^{\perp}, ?\Delta, !B}}{\vdash ?\Gamma, ?\Delta, !B} \longrightarrow \dfrac{\dfrac{\vdash \Gamma, A \qquad \vdash A^{\perp}, \Delta, B}{\vdash \Gamma, \Delta, B}}{\vdash ?\Gamma, ?\Delta, !B}$$

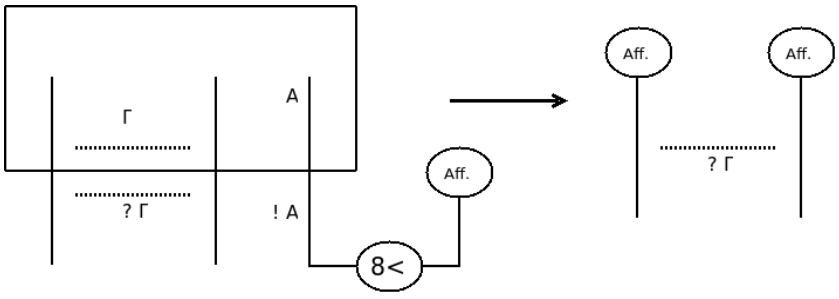
Ce qui, traduit en terme de réseaux, nous donne le joli dessin ci-dessous.



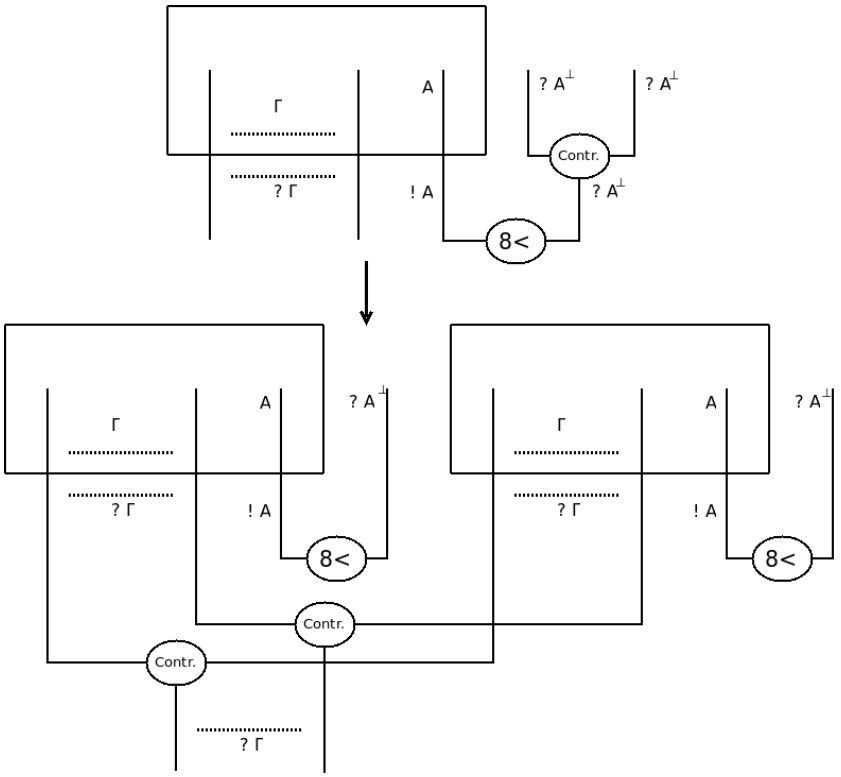
Les boîtes servent à enfermer le contexte pour qu'il soit dupliqué le jour où l'on en aura besoin.

Et les deux autres cas d'élimination qui correspondent dans les réseaux à l'effacement et à la duplication de boîtes sont tout aussi naturels (ce sont les mêmes qu'en logique linéaire habituelle) :

$$\dfrac{\dfrac{\vdash \Gamma, A}{\vdash ?\Gamma, !A} \qquad \dfrac{\vdash \Delta}{\vdash ?A^{\perp}, \Delta}}{\vdash ?\Gamma, \Delta} \longrightarrow \dfrac{\dfrac{\vdash \Delta}{\vdots}}{\vdash ?\Gamma, \Delta}$$



$$\dfrac{\dfrac{\vdash \Gamma, A}{\vdash ?\Gamma, !A} \qquad \dfrac{\vdash ?A^{\perp}, ?A^{\perp}, \Delta}{\vdash ?A^{\perp}, \Delta}}{\vdash ?\Gamma, \Delta} \quad \longrightarrow \quad \dfrac{\dfrac{\dfrac{\vdash \Gamma, A}{\vdash ?\Gamma, !A} \qquad \dfrac{\vdash \Gamma, A}{\vdash ?\Gamma, !A} \qquad \vdash ?A^{\perp}, ?A^{\perp}, \Delta}{\vdash ?\Gamma, ?A^{\perp}, \Delta}}{\vdash ?\Gamma, ?\Gamma, \Delta}}{\vdots}$$



Ces règles de réduction de réseaux ont une propriété remarquable (dite de *stratification*) : le nombre de boîtes qui contiennent un nœud donné (on appelle ça la *profondeur* d'un nœud) reste inchangé après une étape d'élimination de coupures (sauf pour les noeuds de coupures évidemment).

Nous allons construire, dans un réseau sans quantificateur (on admettra le résultat avec les quantificateurs) avec N noeuds, une stratégie de réduction des coupures assez contre-intuitive dont on bornera le nombre d'étapes. L'idée générale de la réduction est que l'on réduit les coupures profondeur par profondeur en partant des coupures de plus faible profondeur.

1. On commence par réduire toutes les coupures multiplicatives (les coupures que l'on a vues en cours, c-à-d sans boîtes) à profondeur nulle. On ne va pas en faire plus que que le nombre de nœuds multiplicatifs dans le réseau (je rappelle que l'élimination des coupures multiplicatives fait diminuer strictement le nombre de nœuds).
2. On réduit ensuite les coupures à profondeur nulle de la forme de celles que l'on a vues au dessus («prommotion/promotion», «promotion/contraction», «promotion/affaiblissement») en réduisant à chaque fois une coupure de plus long chemin

«sans rebond» maximal qui part du côté « ?» de la coupure. Un *chemin «sans rebond»* est un chemin qui, quand il passe par un nœud, va des prémisses à la conclusion ou de la conclusion à l'un des prémisses et quand il doit rentrer dans une boîte par une porte « ?» saute à la porte « !» et réciproquement saute de la porte « !» à une porte « ?». Ces réductions ne crées pas de coupures multiplicatives et on ne peut pas en faire plus que le nombre de noeuds exponentiels dans le réseau initial. En effet, en réduisant une telle coupure on a réduit d'au moins un le nombre de nœuds de conclusion « ?» joignables par un chemin «sans rebond» qui part d'une coupure (à vérifier en méditant les dessins).

À la fin de ce procédé, on a éliminé toutes les coupures à profondeur nulle en au plus N étapes. Et chaque étape multiplie au plus la taille du réseau par trois (à cause bien sûr de l'élimination de type «promotion/contraction»). Donc à la fin de l'élimination de toutes les coupures à profondeur nulle, la taille du réseau est bornée $3^N \times N$ ce qui est inférieur à 3^{3^N}.

On peut maintenant appliquer récursivement ce même procédé à l'intérieur de toutes les boîtes à profondeur nulle. Et montrer par récurrence que si la profondeur du réseau est majorée par d, alors on peut éliminer toutes les coupures en moins de

$$\left.\begin{array}{c} 3^{\cdot^{\cdot^{3^N}}} \\ 3 \end{array}\right\} \text{tour de hauteur } 2d+1$$

étapes. On peut montrer également que la stratégie décrite est la pire, c'est à dire que tout autre façon de réduire prendrait moins de temps et en déduire que la borne que l'on a trouvée est vérifiée quelque soit la stratégie d'élimination que l'on adopte. Mais je ne pense pas que ce soit si facile.

Si on voit les réseaux de preuves comme un modèle de calcul, il est alors possible de voir les preuves de logique linéaire élémentaire comme des programmes. Mais il faut bien admettre que c'est un modèle de calcul peu pratiqué et que l'on n'est pas prêt de voir les programmeurs troquer leur éditeur de texte favori contre un logiciel de dessin vectoriel.

La section suivante présentera donc un fragment intuitionniste de la logique linéaire élémentaire dans lequel on peut extraire des λ-termes qui permettront entre autre de représenter des fonctions qui calculent sur les entiers.

2 Système $\mathcal{F}_E$

Considérons le petit système de règles suivant que l'on pourrait appeler la logique linéaire élémentaire minimale du second ordre.

$$\frac{}{A \vdash A}$$

$$\frac{\Gamma \vdash A \qquad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$$

$$\frac{\Gamma \vdash A}{!\Gamma \vdash !A} \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B}$$

$$\frac{\Gamma \vdash C \qquad \Delta, C \vdash A}{\Gamma, \Delta \vdash A}\ \text{COUPURE}$$

$$\frac{\Gamma, \forall X\, A \vdash B}{\Gamma, A[C/X] \vdash B} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall X\, A}\ (*)$$

En interprétant la $A \multimap B$ comme $A^{\perp} \parr B$, on peut voir que ce système est bien un fragment de la logique linéaire élémentaire et on hérite gratuitement du résultat précédent sur l'élimination des coupures.

On peut le transformer facilement en un système de typage du λ-calcul en rajoutant des termes par-ci par-là (un terme par élément du contexte et un pour le but) et en s'inspirant du système $\mathcal{F}$ de Girard (voir [4] par exemple) :

$$\frac{}{t : A \vdash t : A}$$

$$\frac{\Gamma \vdash t : A \qquad \Delta, f\, t : B \vdash s : C}{\Gamma, \Delta, f : A \multimap B \vdash s : C} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ (+)$$

$$\frac{\Gamma \vdash t : A}{!\Gamma \vdash t :!A} \qquad \frac{\Gamma, t :!A, t :!A \vdash s : B}{\Gamma, t :!A \vdash s : B} \qquad \frac{\Gamma \vdash s : B}{\Gamma, t :!A \vdash s : B}$$

$$\frac{\Gamma \vdash u : C \qquad \Delta, x : C \vdash t : A}{\Gamma, \Delta \vdash t[u/x] : A}\ \text{COUPURE}$$

$$\frac{\Gamma, t : \forall X\, A \vdash s : B}{\Gamma, t : A[C/X] \vdash s : B} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X\, A}\ (*)$$

(*) X n'est pas libre dans Γ

(+) x n'est pas libre dans Γ

On remarque aussi que les termes traversent les règles des deux dernières lignes sans être affectés (on dit que le typage n'est pas dirigé par la syntaxe). Dans la suite quand on dira qu'un terme t est typable dans système $\mathcal{F}_E$ cela voudra dire qu'il existe un énoncé A et une preuve de $\vdash t : A$ (dans un contexte vide à gauche). Étant donné un énoncé A et une preuve de $\vdash A$ en logique linéaire élémentaire minimale du second ordre, il est facile d'extraire en suivant la correspondance de Curry et d'Howard un λ-terme t et une preuve dans système $\mathcal{F}_E$ que $\vdash t : A$. Le terme extrait d'une preuve sans coupure est sous forme normal car sans coupure on ne pas introduire de λ à gauche et on ne peut donc pas

construire de redex. Réciproquement, si on dispose d'une preuve de la forme $\vdash t : A$ on peut en extraire trivialement une preuve de A en logique linéaire élémentaire en oubliant les termes.

Les coupures «flèche gauche/flèche droite» s'éliminent de la façon suivante :

$$\dfrac{\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B} \qquad \dfrac{\Sigma \vdash s : A \qquad \Delta, (\lambda x.t)\, s : B \vdash u : C}{\Sigma, \Delta, f : A \multimap B \vdash u : C}}{\Gamma, \Sigma, \Delta \vdash u[\lambda x.t/f] : C}$$

$$\longrightarrow \quad \dfrac{\dfrac{\Sigma \vdash s : A \qquad \Gamma, s : A \vdash t[s/x] : B}{\Gamma, \Sigma \vdash t[s/x] : B} \qquad \Delta, t[s/x] : B \vdash u : C}{\Gamma, \Sigma, \Delta \vdash u : C}$$

On voit donc que l'élimination des coupures de type «intro flèche gauche/intro flèche droite» correspond à la β réduction du λ-terme. Ainsi, une stratégie d'élimination des coupures nous donnera une stratégie de réduction du λ-terme en au plus autant d'étapes. Pour ce qui est le réciproque, j'y réfléchirai demain.

C'est bien beau tout ça, mais nous n'avons pas encore écrit notre premier programme. Pour ça, il nous faudrait une façon de représenter les données. En système $\mathcal{F}$, on utilise l'énoncé $\forall X, (X \multimap X) \multimap X \multimap X$ pour typer les entiers de Church. Dans notre système, seul l'entier de Church $\lambda f\, x.f\, x$ est typable avec ce type (à cause de la linéarité). C'est donc pour pouvoir appliquer plusieurs fois ou zéro fois la fonction f que l'on aimerait utiliser le type $\forall X, !(X \multimap X) \multimap X \multimap X$. Ce type nous suffirait si nous faisions de la logique linéaire habituelle, mais en logique linéaire élémentaire cet énoncé n'admet que la preuve qui correspond à l'entier de Church zéro. Il nous faut donc rajouter un ploum pour pouvoir faire une promotion et on obtient le type suivant (que je noterai N dans la suite) :

$$\forall X, !(X \multimap X) \multimap !(X \multimap X).$$

À titre d'exemple voici une preuve de la typabilité du troisième entier de Church.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{f\ (f\ x)} : X \vdash \mathrm{f\ (f\ x)} : X} \quad \dfrac{}{\mathrm{f\ (f\ (f\ x))} : X \vdash \mathrm{f\ (f\ (f\ x))} : X}}{\mathrm{f\ x} : X \vdash \mathrm{f\ x} : X} \quad \mathrm{f\ (f\ x)} : X, \mathrm{f} : X \multimap X \vdash \mathrm{f\ (f\ (f\ x))} : X}}{\mathrm{x} : X \vdash \mathrm{x} : X} \quad \mathrm{f\ x} : X, \mathrm{f} : X \multimap X, \mathrm{f} : X \multimap X \vdash \mathrm{f\ (f\ (f\ x))} : X}}{\mathrm{x} : X, \mathrm{f} : X \multimap X, \mathrm{f} : X \multimap X, \mathrm{f} : X \multimap X \vdash \mathrm{f\ (f\ (f\ x))} : X}}{\mathrm{f} : X \multimap X, \mathrm{f} : X \multimap X, \mathrm{f} : X \multimap X \vdash \lambda\ \mathrm{x.\ f\ (f\ (f\ x))} : X \multimap X}}{\mathrm{f} :!(X \multimap X), \mathrm{f} :!(X \multimap X), \mathrm{f} :!(X \multimap X) \vdash \lambda\ \mathrm{x.\ f\ (f\ (f\ x))} :!(X \multimap X)}}{\mathrm{f} :!(X \multimap X), \mathrm{f} :!(X \multimap X) \vdash \lambda\ \mathrm{x.\ f\ (f\ (f\ x))} :!(X \multimap X)}}{\mathrm{f} :!(X \multimap X) \vdash \lambda\ \mathrm{x.\ f\ (f\ (f\ x))} :!(X \multimap X)}}{\vdash \lambda\ \mathrm{f\ x.\ f\ (f\ (f\ x))} :!(X \multimap X) \multimap !(X \multimap X)}}{\vdash \lambda\ \mathrm{f\ x.\ f\ (f\ (f\ x))} : N}$$

Maintenant que l'on dispose d'un type pour les entiers on peut implémenter les fonctions arithmétiques habituelles. On dira d'un λ-terme t qu'il est de *complexité élémentaire* (ou qu'il s'exécute en *temps élémentaire*) s'il existe un entier h tel que pour tout entier

N, il existe une façon de normaliser t appliqué au N-ième entier de Church en moins de

$$\left.\begin{matrix} 2^{\cdot^{\cdot^{2^N}}} \\ 2 \end{matrix}\right\} \text{tour de hauteur } h$$

étapes. Tout ce qu'on a fait jusqu'à présent nous montre que les termes typables en système $\mathcal{F}_E$ s'exécutent en temps élémentaire.

On peut implémenter l'addition

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{X \vdash X} \quad \dfrac{}{X \vdash X}}{X, X \multimap X \vdash X}}{X, X \multimap X, X \multimap X \vdash X}}{X \multimap X, X \multimap X \vdash X \multimap X}}{!(X \multimap X), !(X \multimap X) \vdash !(X \multimap X)}}{!(X \multimap X) \vdash !(X \multimap X) \quad !(X \multimap X), !(X \multimap X), !(X \multimap X) \multimap !(X \multimap X) \vdash !(X \multimap X)}}{!(X \multimap X), !(X \multimap X), !(X \multimap X) \multimap !(X \multimap X), !(X \multimap X) \multimap !(X \multimap X) \vdash !(X \multimap X)}}{!(X \multimap X), !(X \multimap X), N, !(X \multimap X) \multimap !(X \multimap X) \vdash !(X \multimap X)}}{!(X \multimap X), !(X \multimap X), N, N \vdash !(X \multimap X)}}{!(X \multimap X), N, N \vdash !(X \multimap X)}}{N, N \vdash !(X \multimap X) \multimap !(X \multimap X)}}{N, N \vdash N}}{N \vdash N \multimap N}}{\vdash N \multimap N \multimap N}$$

dont on extrait bien le λ-terme habituel : plus $= \lambda n\, m\, f\, x.n\, f\, (m\, f\, x)$. Ainsi que la multiplication

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!(X \multimap X) \vdash !(X \multimap X)} \quad \dfrac{}{!(X \multimap X) \vdash !(X \multimap X)}}{!(X \multimap X), !(X \multimap X) \multimap !(X \multimap X) \vdash !(X \multimap X)}}{!(X \multimap X) \vdash !(X \multimap X) \quad !(X \multimap X), !(X \multimap X) \multimap !(X \multimap X) \vdash !(X \multimap X)}}{!(X \multimap X), !(X \multimap X) \multimap !(X \multimap X), !(X \multimap X) \multimap !(X \multimap X) \vdash !(X \multimap X)}}{!(X \multimap X), !(X \multimap X) \multimap !(X \multimap X), N \vdash !(X \multimap X)}}{!(X \multimap X), N, N \vdash !(X \multimap X)}}{N, N \vdash !(X \multimap X) \multimap !(X \multimap X)}}{N, N \vdash N}}{N \vdash N \multimap N}}{\vdash N \multimap N \multimap N}$$

dont on extrait bien le λ-terme mult $= \lambda n\, m\, f.n\, (mf)$.

Plus intéressant, maintenant. Si l'on dispose d'un terme $\vdash$ step : $N \multimap N$ et d'un terme $\vdash$ base : N, on peut prouver que le terme $\lambda n.n\,\text{step}\,\text{base}$ est typable de type $N \multimap !N$. Maintenant, si on prends step $=$ mult 2 la fonction qui double son entrée et base égal au premier entier de Church, alors le terme typable $\lambda n.n\,\text{step}\,\text{base}$ nous donne l'exponentiation de base 2. Fort heureusement, son type est $N \multimap !N$ et non $N \multimap N$

(sinon on pourrait réappliquer la même genre d'itération et fabriquer un terme qui calcule des tours d'exponentielles, ce qui contredirait le fait qu'il puisse s'exécuter en temps élémentaire).

Dans [3], les auteurs montrent qu'en rajoutant le connecteur & au système que j'ai décrit ainsi qu'en rajoutant un constructeur de couples et les deux projections au λ-calcul, on peut implémenter toutes les fonctions qui s'exécutent en temps élémentaire.

Pour cela, ils utilisent un résultat dû à Kalmar qui affirme que pour qu'une classe de fonctions contienne toutes les fonctions élémentaires il suffit qu'elle contienne les constantes, les projections, l'addition, la multiplication et la soustraction et qu'elle soit close par des schémas de composition, de produit et de somme bornés. Dans leurs encodages, le & est utilisé pour implémenter le prédécesseur (en utilisant la méthode habituelle qui consiste à itérer la fonction $(x, y) \mapsto (Sx, x)$ n fois sur $(0, 0)$ et à prendre la seconde composante). Dans le système sans &, je ne sais pas si la question est ouverte, mais j'ai tout de même réussi à typer un terme qui implémente le calcul du prédécesseur mais avec le type $N \multimap !!N$ (ce qui ne nous permet d'appliquer les schémas). Je reste pourtant convaincu qu'il existe un moyen de l'implémenter, ce serait à mon avis surprenant que le système sans & ne soit pas capable de typer un prédécesseur.

3 Conclusion : Vers l'Arithmétique Fonctionnelle Élémentaire ?

Au final, essayer de typer des termes dans un système de typage qui n'est pas dirigé par la syntaxe n'est pas beaucoup plus simple que d'écrire des réseaux preuves directement. C'est pour cette raison que j'ai essayé de développer une nouvelle approche pour programmer en temps élémentaire : L'idée est d'essayer de concevoir un système de preuve dont la logique sous-jacente est une logique linéaire élémentaire. Ainsi, dans un tel système de preuve, pour programmer le prédécesseur, l'utilisateur devra par exemple essayer de prouver un énoncé qui ressemblerait à :

$$\forall x, Nx \multimap \exists p, (Sp = x) \oplus (x = 0)$$

.

Nous avons essayé d'élémentariser *l'arithmétique fonctionnelle du second ordre* qui est une extension de système $\mathcal{F}$ dans laquelle Jean-Louis Krivine a rajouté une quantification au premier ordre ainsi que des constantes et des symboles de fonction pour fabriquer les termes du premier ordre. Le comportement des termes du premier ordre est alors dicté par des règles de réécriture (les termes peuvent être réécrits dans les preuves) comme par exemple pour l'addition :

```
plus(0,x) = x
plus(succ(x), y) = succ(plus(x,y))
```

et pour obtenir un terme qui implémente l'addition il faut réussir à prouver dans cette logique que la somme de deux entiers est un entier :

$$\forall xy, Nx \multimap Ny \multimap N\mathrm{plus}(x, y)$$

. Il n'est pas surprenant que la bonne façon de définir le prédicat Nx qui affirme que x est un entier soit :

$$N\,x := \forall X, (\forall y, X\,y \multimap X\,\mathrm{succ}(y)) \multimap X\,0 \multimap X\,x$$

D'ailleurs en oubliant le premier ordre, on retombe sur la définition de N précédente.

Je n'ai hélas plus le temps ni l'énergie pour ce soir de développer plus avant la formalisation de ces concepts d'autant plus que certains d'entre eux ne me sont pas encore très familiers. Mais vous serez peut-être amusé d'apprendre que j'ai implémenté un assistant de preuve qui vérifie les preuves et procède à l'extraction du λ-terme le tout dans cette version élémentarisée de l'arithmétique fonctionnelle de Krivine.

La preuve que la somme de deux entiers est un entier ne résiste pas longtemps. Par contre après une journée de travail, je n'ai pas réussi à prouver que le produit de deux entiers est un entier (sauf en rajoutant « ! » sur la conclusion ce qui n'est pas satisfaisant puisque ça m'empêche de coder les fonctions de Kalmar). Je crains de plus en plus que ce ne soit pas possible de le prouver et que cette voie puisse être une impasse.

Références

[1] Laurent O. *Théorie de la démonstration*, notes de cours.

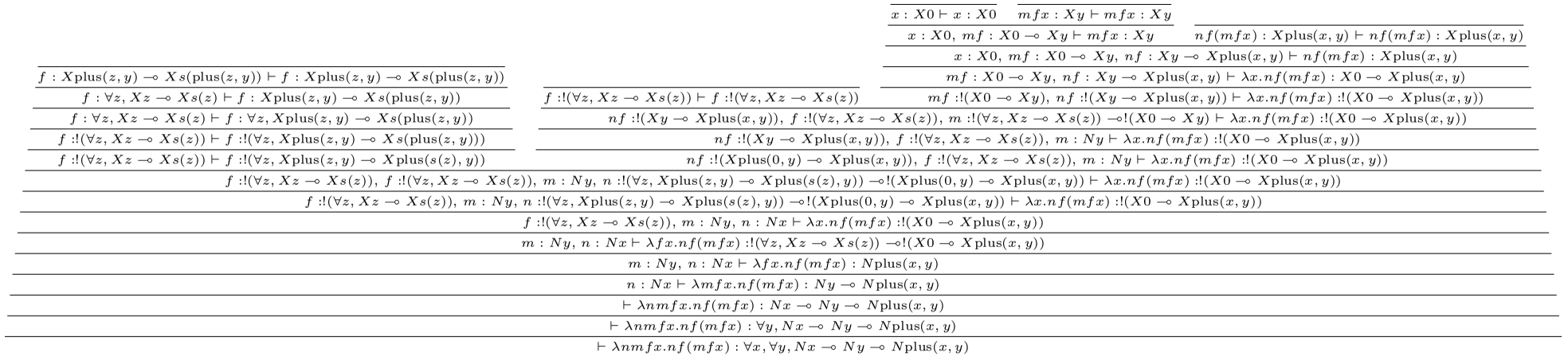
[2] Girard J.-Y. *Light Linear Logic.*

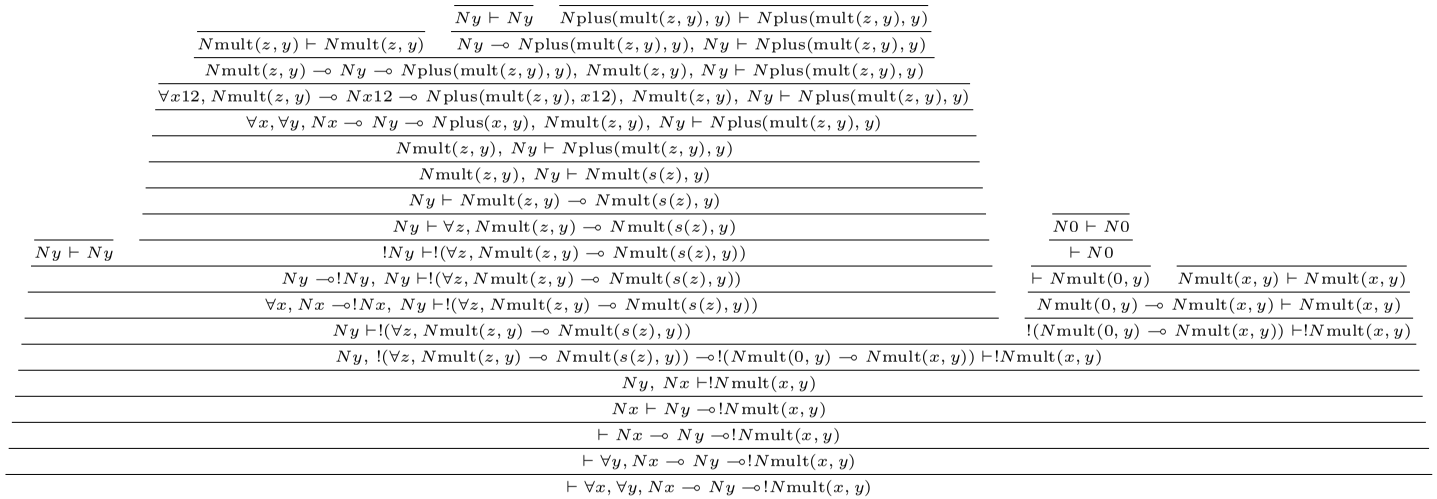
[3] Danos V. Joinet J.-B., *Linear Logic & Elementary Time.*

[4] Girard J.-Y., Lafont Y., Taylor P., *Proofs and Types.*

[5] Krivine J.-L. *Lambda-calcul, types et modèles.*

Je vous laisse en compagnie d'une preuve «d'arithmétique élémentaire» que la somme de deux entiers est un entier et d'une preuve «d'arithmétique élémentaire» que le produit de deux entiers est PLOUM ! un entier.


$x : X0 \vdash x : X0$
$mfx : Xy \vdash mfx : Xy$
$x : X0,\ mf : X0 \multimap Xy \vdash mfx : Xy$
$nf(mfx) : X\text{plus}(x, y) \vdash nf(mfx) : X\text{plus}(x, y)$
$x : X0,\ mf : X0 \multimap Xy,\ nf : Xy \multimap X\text{plus}(x, y) \vdash nf(mfx) : X\text{plus}(x, y)$
$f : X\text{plus}(z, y) \multimap Xs(\text{plus}(z, y)) \vdash f : X\text{plus}(z, y) \multimap Xs(\text{plus}(z, y))$
$mf : X0 \multimap Xy,\ nf : Xy \multimap X\text{plus}(x, y) \vdash \lambda x.nf(mfx) : X0 \multimap X\text{plus}(x, y)$
$f : \forall z, Xz \multimap Xs(z) \vdash f : X\text{plus}(z, y) \multimap Xs(\text{plus}(z, y))$
$f :!(\forall z, Xz \multimap Xs(z)) \vdash f :!(\forall z, Xz \multimap Xs(z))$
$mf :!(X0 \multimap Xy),\ nf :!(Xy \multimap X\text{plus}(x, y)) \vdash \lambda x.nf(mfx) :!(X0 \multimap X\text{plus}(x, y))$
$f : \forall z, Xz \multimap Xs(z) \vdash f : \forall z, X\text{plus}(z, y) \multimap Xs(\text{plus}(z, y))$
$nf :!(Xy \multimap X\text{plus}(x, y)),\ f :!(\forall z, Xz \multimap Xs(z)),\ m :!(\forall z, Xz \multimap Xs(z)) \multimap !(X0 \multimap Xy) \vdash \lambda x.nf(mfx) :!(X0 \multimap X\text{plus}(x, y))$
$f :!(\forall z, Xz \multimap Xs(z)) \vdash f :!(\forall z, X\text{plus}(z, y) \multimap Xs(\text{plus}(z, y)))$
$nf :!(Xy \multimap X\text{plus}(x, y)),\ f :!(\forall z, Xz \multimap Xs(z)),\ m : Ny \vdash \lambda x.nf(mfx) :!(X0 \multimap X\text{plus}(x, y))$
$f :!(\forall z, Xz \multimap Xs(z)) \vdash f :!(\forall z, X\text{plus}(z, y) \multimap X\text{plus}(s(z), y))$
$nf :!(X\text{plus}(0, y) \multimap X\text{plus}(x, y)),\ f :!(\forall z, Xz \multimap Xs(z)),\ m : Ny \vdash \lambda x.nf(mfx) :!(X0 \multimap X\text{plus}(x, y))$
$f :!(\forall z, Xz \multimap Xs(z)),\ f :!(\forall z, Xz \multimap Xs(z)),\ m : Ny,\ n :!(\forall z, X\text{plus}(z, y) \multimap X\text{plus}(s(z), y)) \multimap !(X\text{plus}(0, y) \multimap X\text{plus}(x, y)) \vdash \lambda x.nf(mfx) :!(X0 \multimap X\text{plus}(x, y))$
$f :!(\forall z, Xz \multimap Xs(z)),\ m : Ny,\ n :!(\forall z, X\text{plus}(z, y) \multimap X\text{plus}(s(z), y)) \multimap !(X\text{plus}(0, y) \multimap X\text{plus}(x, y)) \vdash \lambda x.nf(mfx) :!(X0 \multimap X\text{plus}(x, y))$
$f :!(\forall z, Xz \multimap Xs(z)),\ m : Ny,\ n : Nx \vdash \lambda x.nf(mfx) :!(X0 \multimap X\text{plus}(x, y))$
$m : Ny,\ n : Nx \vdash \lambda fx.nf(mfx) :!(\forall z, Xz \multimap Xs(z)) \multimap !(X0 \multimap X\text{plus}(x, y))$
$m : Ny,\ n : Nx \vdash \lambda fx.nf(mfx) : N\text{plus}(x, y)$
$n : Nx \vdash \lambda mfx.nf(mfx) : Ny \multimap N\text{plus}(x, y)$
$\vdash \lambda nmfx.nf(mfx) : Nx \multimap Ny \multimap N\text{plus}(x, y)$
$\vdash \lambda nmfx.nf(mfx) : \forall y, Nx \multimap Ny \multimap N\text{plus}(x, y)$
$\vdash \lambda nmfx.nf(mfx) : \forall x, \forall y, Nx \multimap Ny \multimap N\text{plus}(x, y)$


$Ny \vdash Ny$
$Nplus(mult(z, y), y) \vdash Nplus(mult(z, y), y)$
$Nmult(z, y) \vdash Nmult(z, y)$
$Ny \multimap Nplus(mult(z, y), y),\ Ny \vdash Nplus(mult(z, y), y)$
$Nmult(z, y) \multimap Ny \multimap Nplus(mult(z, y), y),\ Nmult(z, y),\ Ny \vdash Nplus(mult(z, y), y)$
$\forall x12,\ Nmult(z, y) \multimap Nx12 \multimap Nplus(mult(z, y), x12),\ Nmult(z, y),\ Ny \vdash Nplus(mult(z, y), y)$
$\forall x, \forall y,\ Nx \multimap Ny \multimap Nplus(x, y),\ Nmult(z, y),\ Ny \vdash Nplus(mult(z, y), y)$
$Nmult(z, y),\ Ny \vdash Nplus(mult(z, y), y)$
$Nmult(z, y),\ Ny \vdash Nmult(s(z), y)$
$Ny \vdash Nmult(z, y) \multimap Nmult(s(z), y)$
$Ny \vdash \forall z, Nmult(z, y) \multimap Nmult(s(z), y)$
$N0 \vdash N0$
$Ny \vdash Ny$
$!Ny \vdash !(\forall z, Nmult(z, y) \multimap Nmult(s(z), y))$
$\vdash N0$
$Ny \multimap !Ny,\ Ny \vdash !(\forall z, Nmult(z, y) \multimap Nmult(s(z), y))$
$\vdash Nmult(0, y)$
$Nmult(x, y) \vdash Nmult(x, y)$
$\forall x, Nx \multimap !Nx,\ Ny \vdash !(\forall z, Nmult(z, y) \multimap Nmult(s(z), y))$
$Nmult(0, y) \multimap Nmult(x, y) \vdash Nmult(x, y)$
$Ny \vdash !(\forall z, Nmult(z, y) \multimap Nmult(s(z), y))$
$!(Nmult(0, y) \multimap Nmult(x, y)) \vdash !Nmult(x, y)$
$Ny,\ !(\forall z, Nmult(z, y) \multimap Nmult(s(z), y)) \multimap !(Nmult(0, y) \multimap Nmult(x, y)) \vdash !Nmult(x, y)$
$Ny,\ Nx \vdash !Nmult(x, y)$
$Nx \vdash Ny \multimap !Nmult(x, y)$
$\vdash Nx \multimap Ny \multimap !Nmult(x, y)$
$\vdash \forall y, Nx \multimap Ny \multimap !Nmult(x, y)$
$\vdash \forall x, \forall y, Nx \multimap Ny \multimap !Nmult(x, y)$