Interprétation calculatoire de théorèmes mathématiques en logique classique.

Édouard Maurel-Segala

6 novembre 2003

Table des matières

Introduction

En programmation, l'utilisation du typage permet de contrôler dans quel domaine vivent les objets que l'on manipule. À un niveau plus élevé, il permet de vérifier si certains programmes se comportent bien comme prévus. Prenons par exemple un lambda-terme qui a pour type

$$\forall n(\mathrm{Ent}\, n \to \mathrm{Ent}\, f(n))$$

où Ent p est une notation remplaçant une formule que nous expliciterons plus loin qui signifie "p est un entier". Alors on peut montrer que le programme calcul effectivement la fonction f sur les entiers. On voit donc que le comportement du programme est en partie décrit dans son type. Le typage permet une programmation plus sûre en permettant de prouver qu'un programme suit bien un comportement donné. Reste à savoir quels types de comportement on peut imposer à l'aide du typage.

C'est le problème de la spécification, qui s'intéresse à partir d'une formule donnée à décrire le comportement commun des programmes qui ont pour type cette formule. Le but de cet exposé est d'introduire progressivement les différentes notions liées au problème de la spécification. Nous introduirons tout d'abord l'isomorphisme de Curry-Howard qui relie preuves et programmes, types et formules. Nous étudierons ensuite la réalisabilité, un outil puissant que nous utiliserons pour essayer de résoudre quelques problèmes de spécification.

1 L'isomorphisme de Curry-Howard.

Présentons tout d'abord le langage de programmation dans lequel nous allons travailler : le lambda-calcul. Ce langage de programmation est remarquable à la fois par sa simplicité (seulement 3 constructions, une seule règle d'évaluation) et son expressivité. C'est un outil privilégié pour l'étude théorique de programme. Pour le définir on se donne un ensemble infini de variables $x, y, z, \cdots$ et on autorise les constructions données par la grammaire suivante :

$$t = x|tt|\lambda x.t$$

L'ensemble Λ des termes ainsi créés représentent les programmes. La seule règle que d'évaluation est la suivante :

$$\lambda x.tu \to_\beta t[u/x] \quad (\beta)$$

où $t[u/x]$ représente le terme t où toutes les occurences de x ont été remplacée par u. Tout terme peut-être à la fois pris comme une fonction ou un argument de fonction, ce langage est dit «fonctionnel». L'aspect mathématique est quand à lui donné par les formules du langage propositionnel. Les règles de démonstration sont :

$$\frac{}{\Gamma, A \vdash A}\ (\mathrm{ax})$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \ (\to I)$$

$$\frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B} \ (\to E)$$

Il s'agit des règles de démonstration de la déduction naturelle. Voyons maintenant les règles de typage des lambdas-termes. Les jugements de typage sont de la forme $\Gamma \vdash t : A$ où t est un lambda-terme, A une formule et Γ une suite de jugements de typage sur des variables de terme de la forme $x_i : A_i$. Γ est appelé le contexte de typage. On relie les termes aux formules par les règles de typage suivantes :

$$\frac{}{\Gamma, x : A \vdash x : A} \ (\text{ax})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \ (\to I)$$

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B} \ (\to E)$$

On constate immédiatement qu'en effaçant les terme de ces règles on obtient les règles de démonstration données plus haut. Un terme t est typable de type A s'il existe un arbre de typage aboutissant à la conclusion $\vdash t : A$. En gommant les termes de cette arbre, on obtient une preuve de la formule A. Réciproquement, en partant de l'arbre d'une démonstration d'une formule A, on peut de manière évidente décorer cet arbre pour obtenir un terme t de type A. Le terme associé est unique au nom des variables près. C'est cette bijection qui constitue la forme la plus simple de l'isomorphisme de Curry-Howard.

2 Première extension de l'isomorphisme de Curry-Howard

L'isomorphisme tel qu'il est donné ne concerne que des types très simples. On veut pouvoir l'enrichir pour traiter des formules mathématiques plus complexes que celles de la logique propositionnelle. Plus d'expressivité sur les types permet un meilleur contrôle sur les lambdas-termes associés comme nous le verrons dans la partie suivante. Une première manière d'étendre l'isomorphisme est de passer à la logique du second ordre en ajoutant les règles suivantes :

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x A} \ (\forall^1 I),\ x \text{ n'est pas libre dans } \Gamma$$

$$\frac{\Gamma \vdash t : \forall x A}{\Gamma \vdash t : A[a/x]} \ (\forall^2 E),\ a \text{ terme quelconque}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X A} \ (\forall^2 I),\ X \text{ n'est pas libre dans } \Gamma$$

$$\frac{\Gamma \vdash t : \forall X A}{\Gamma \vdash t : A[\Phi/X]} \ (\forall^2 E),\ \Phi \text{ formule quelconque}$$

La logique du second ordre est une logique très riche dans laquelle il est possible de formaliser une bonne partie des mathématiques (arithmétique, analyse...). Pour l'isomorphisme de Curry-Howard il offre un cadre d'une grande souplesse qui mèle à la fois une grande expressivité et un jeu de connecteurs logiques très réduits. En toute rigueur si on se contente d'éffacer les termes des règles décrites ci-dessus on ne retombe pas exactement sur la logique du second ordre.

La seule différence est la présence de la règle $(\forall^2 E)$ et l'absence du shéma de compréhension :

$$(SC) : \exists Y \forall y_1, \ldots, y_q (F[X_1, \ldots, X_n, x_1, \ldots, x_p, y_1, \ldots, y_q] \Leftrightarrow Y y_1 \ldots y_q)$$

où F est une formule à paramètres. Mais on a la propriété :

Proposition 1 $(\forall^2 E) \Leftrightarrow (SC)$

On est donc dans un système équivalent à la logique du second ordre. Présentons maintenant un outil essentiel permettant de travailler avec le typage.

3 Réalisabilité intuitionniste

On va chercher un opérateur $|.|_i : Form \to \mathbb{P}(\Lambda)$ permettant d'interpréter toute formule F par un sous-ensemble $|F|_i$ de Λ. On dira qu'un terme t réalise F si $t \in |F|_i$, on le notera $t \Vdash_i F$.

Pour cela puisse nous être utile, on espère que la notion de réalisation sera plus maniable que celle de typage et que l'on aura

$$t \vdash_i F \Rightarrow t \Vdash_i F.$$

L'idée dérrière la réalisabilité est donc de déduire du typage un jugement sur le terme. La seule condition à imposer sur ces jugements est qu'ils se perpétuent via les règles de typage.

Introduisons quelques notations : On dit qu'une partie A de Λ est saturée si elle a la propriété :

$$\forall t, u, t[u/x] \in A \Rightarrow \lambda x.tu \in A$$

On note $\mathcal{S}$ l'ensemble des parties saturées de Λ. On se donne un modèle $\mathcal{M}$ de logique du second ordre avec le shéma de compréhension à valeurs de vérité dans $\mathcal{S}$. On va interpréter des formules closes à paramétres dans $\mathcal{M}$, c'est-à-dire que des constantes a de $\mathcal{M}$ peuvent remplacer des variables du premier ordre et que des fonctions $\mathcal{A}$ de $\mathcal{M}^n$ dans $\mathcal{S}$ peuvent remplacer des symboles de fonctions n-aires.

On définit, si $A, B \in \mathcal{S}$,

$$A \Rightarrow B = \{t \in \Lambda | \forall u \in A, tu \in B\}$$

On definit alors par induction l' interprétations $|F|_i \subset \Lambda$ de F formule close :

$$\begin{array}{lcl} |\mathcal{A}a_1, a_2, \ldots, a_n|_i & = & \mathcal{A}a_1, a_2, \ldots, a_n \\ |F \to G|_i & = & |F|_i \Rightarrow |G|_i \\ |\forall X F|_i & = & \bigcap_{\mathcal{A} \in \mathcal{M}^n \to \mathcal{S}} |F[X := \mathcal{A}]|_i \\ |\forall x F|_i & = & \bigcap_{a \in \mathcal{M}} |F[x := a]|_i \end{array}$$

Voici maintenant le théorème central qui est omniprésent pour la recherche du problème de la spécification dans le cas intuitionniste.

Théorème 1

$$\begin{array}{l} y_1 : F_1, y_2 : F_2, \ldots, y_n : F_n \vdash_i t : G \\ s_1 \in |F_1|_i, s_2 \in |F_2|_i, \ldots, s_n \in |F_n|_i \end{array} \Rightarrow t[s_1/y_1, s_2/y_2, \ldots, s_n/y_n] \in |G|_i$$

Corollaire 1 $\vdash_i t : A \Rightarrow t \Vdash_i A$

Regardons maintenant quelques résultats que l'on peut obtenir à partir de cet outil.

4 Quelques exemples de spécifications intuitionniste

Losque l'on se restreind à l'aspect intuitionniste on obtient de très fortes caractérisations grâce au lemme d'adéquation.

Proposition 2 *Soit Ent n la formule définit par* $\forall X((Xp \to X(p+1)), X0 \to Xn)$ *et qui se lit "n est un entier". Et soit* $\overline{n} = \lambda f, x.f^n x$ *le* $n^{ième}$ *entier de Church. Alors :*

$$\begin{array}{lcl} t \vdash_i A \to A & \Rightarrow & t \to_{\beta\eta} \lambda x.x \\ t \vdash_i A, B \to A & \Rightarrow & t \to_{\beta\eta} \lambda x, y.x \\ t \vdash_i A, B \to B & \Rightarrow & t \to_{\beta\eta} \lambda x, y.y \\ t \vdash_i \forall X(X, X \to X) & \Rightarrow & t \to_{\beta\eta} \lambda x, y.x \ \textit{ou}\ t \to_{\beta\eta} \lambda x, y.y \\ t \vdash_i \forall X((X \to X), X \to X) & \Rightarrow & \exists n, t \to_{\beta\eta} \overline{n} \\ t \vdash_i Ent\ n & \Rightarrow & t \to_{\beta\eta} \overline{n} \\ t \vdash_i \forall n(Ent\ n \to Ent\, f(n)) & \Rightarrow & \forall n, t\overline{n} \to_{\beta\eta} \overline{f(n)} \end{array}$$

Ces quelques exemples montrent l'efficacité du lemme d'adéquation. La dernière propriété montre en plus à quel point le typage permet de contrôler le comportement d'un terme. Elle est aussi remarquable en ce sens qu'elle montre que toute fonction prouvablement totale dans Peano est représentable par un lambda-terme typable dans ce système. Les résultats de cette section sont les plus précis que nous obtiendrons. En effet, nous allons maintenant étendre notre cadre pour traiter toute la logique et non plus seulement son fragment intuitionniste. Mais ce faisant la structure des lambdas-termes va devenir beaucoup plus compliquée et nous ne pourront plus obtenir de résultats sur leur structure mais seulement sur leur comportement.

5 Extension de l'isomorphisme à la réalisabilité classique

La première extension de l'isomorphisme a été réalisé par Griffin mais nous présenterons seulement la version élaborée par Krivine qui est particulièrement adaptée à ce problème. Il nous faut tout d'abord étendre notre langage :

On commence par se donner trois ensembles infinis : un ensemble de variables de termes $x, y, z, \ldots$, un ensemble de variables de piles $k, k', k'' \ldots$ et un ensemble de constantes de piles $\sigma_1, \sigma_2, \sigma_3, \ldots$. Puis on définit simultanément les notions clés de notre langage : L'ensemble Λ des termes, l'ensemble Π des piles et l'ensemble $\mathcal{P}$ des executables dont la syntaxe est donnée par les règles mutuellement récursives suivantes :

$$\begin{array}{lrcl} \Lambda: & t & = & x|tt|\lambda x.t|cc\lambda k.t|kt|k_\pi \\ \Pi: & \pi & = & \sigma|t::\pi \\ \mathcal{P}: & \rho & = & t\star\pi \end{array}$$

On définit ensuite les règles de réduction suivantes :

$$\begin{array}{rcll} tu\star\pi & \succ & t\star u.\pi & \text{(Push)} \\ \lambda x.t\star u.\pi & \succ & t[u/x]\star\pi & \text{(Pop)} \\ cc\star t.\pi & \succ & t\star k_\pi.\pi & \text{(Save)} \\ k_\pi\star t.\tilde{\pi} & \succ & t\star\pi & \text{(Restore)} \end{array}$$

On voit que la partie programme de notre système est profondément modifié. Même s'il on retrouve en partie la β-réduction, l'évaluation est désormais déterministe et on a ajouté des règles non locales qui agissent sur l'environnement entier. Enfin on rajoute la règle de typage suivante qui nous permet de passer de la logique intuitionniste à la logique classique :

$$\frac{}{\Gamma\vdash cc:((A\to B)\to A)\to A}\ \text{(Pierce)}$$

Développons maintenant un outil similaire à la réalisabilité qui puissent s'appliquer ici :

6 Réalisabilité classique

On va chercher un opérateur $|.| : Form \to \mathbb{P}(\Lambda)$ permettant d'interpréter toute formule F par un sous-ensemble $|F|$ de Λ. On dira qu'un terme t réalise F si $t \in |F|$, on le notera $t \Vdash F$.

Pour cela puisse nous être utile, on espère que la notion de réalisation sera plus maniable que celle de typage et que l'on aura

$$t \vdash F \Rightarrow t \Vdash F.$$

Cependant, en raison de la nouvelle construction cc, on doit apporter des modifications à la méthode utiliéé en logique intuitionniste. Tout est subordonné au choix capital d'un ensemble de processus $\perp\!\!\!\perp \subset \Lambda \times \Pi$. On imposera toujours le fait que $\perp\!\!\!\perp$ est clos par antiréduction :

$$t \star \pi \in \perp\!\!\!\perp , t' \star \pi' \succ t \star \pi \Rightarrow t' \star \pi' \in \perp\!\!\!\perp .$$

On supposera aussi toujours pour des raisons de commodité qu'il est clôt par $(PUSH)$. On définit immédiatement, pour P un ensemble d'executables :

$$P^{\succ^{-1}} = \{\rho \in \mathcal{P} | \exists \rho' \in P, \rho \succ \rho'\}$$

L'ensemble $\perp\!\!\!\perp$ définit une relation d'orthogonalité entre piles et termes on peut définir :

$$\begin{array}{ll} \text{Si } P \subset \Pi, & P^{\perp} = \{t \in \Lambda | \forall \pi \in P, t \star \pi \in \perp\!\!\!\perp \} = P \to \perp\!\!\!\perp \\ \text{Si } T \subset \Lambda, & T^{\perp} = \{\pi \in \Pi | \forall t \in T, t \star \pi \in \perp\!\!\!\perp \} \end{array}$$

On ne peut observer directement un programme, tout ce que l'on peut essayer de faire c'est regarder comment il s'évalue sur certaines piles. Le fait d'être dans $\perp\!\!\!\perp$ est ce qui est observable. Si $P \subset \Pi$ on veut être capable d'émettre des jugements du type : dès qu'on passe une pile de P en argument à t on se retrouve dans $\perp\!\!\!\perp$. Autrement dit, $t \in P^{\perp}$. Un ensemble de la forme P^bot est appelé une valeur de vérité, leur ensemble est noté $\Re_{\perp\!\!\!\perp}$. On va associer à chaque formule une valeur de vérité (ce qui étend la notion classique de modèle). On va donc interpréter une formule F dans un premier temps comme un sous-ensemble Π_F de Π, et on posera :

$$|F| = \Pi_F^{\perp}$$

.

Passons maintenant à la formalisation. On se donne avant tout un modèle $\mathcal{M}$ de logique du second ordre avec le shéma de compréhension. On va interpréter des formules closes à paramétres dans $\mathcal{M}$, c'est-à-dire que des constantes a de $\mathcal{M}$ peuvent remplacer des variables du premier ordre et que des fonctions $\mathcal{A}$ de $\mathcal{M}^n$ dans $\mathbb{P}(\Pi)$ peuvent remplacer des symboles de fonctions n-aires.

On definit alors par induction mutuelle les interprétations $\Pi_F \subset \Pi$ et $|F| \subset \Lambda$ de F :

$$\begin{array}{lcl} \Pi_{\mathcal{A}a_1,a_2,\ldots,a_n} & = & \mathcal{A}a_1, a_2, \ldots, a_n \\ \Pi_{F \to G} & = & |F|.\Pi_G \\ \Pi_{\forall X F} & = & \bigcup_{\mathcal{A} \in \mathcal{M}^n \to \mathbb{P}(\Pi)} \Pi_{F[X:=\mathcal{A}]} \\ \Pi_{\forall x F} & = & \bigcup_{a \in \mathcal{M}} \Pi_{F[x:=a]} \\ |F| & = & \Pi_F^{\perp} \end{array}$$

On peut maintenant énoncer ce lemme, analogue à celui de la logique intuitionniste :

Théorème 2 *Soit* $\perp\!\!\!\perp \subset \Lambda \times \Pi$ *clos par aniti-réduction, alors :*

$$\begin{array}{l} y_1 : F_1, y_2 : F_2, \ldots, y_n : F_n \vdash t : G \\ s_1 \in |F_1|, s_2 \in |F_2|, \ldots, s_n \in |F_n| \end{array} \Rightarrow t[s_1/y_1, s_2/y_2, \ldots, s_n/y_n] \in |G|$$

Corollaire 2 $\vdash t : A \Rightarrow t \in |A|$

7 Quelques exemples de recherche de spécification

L'idée du comportement de la constante permettant les raisonnements par l'absurde se justifie par sa bonne conduite vis-à-vis de la réduction où on a dans des systèmes voisins l'invariance du typage par réduction (on ne l'a pas ici car on ne type pas les constantes de piles). Mais on peut se demander quel est le comportement d'autres termes ayant pour type la loi de Pierce. La propriété suivante montre qu'ils ont dans une certaine mesure le même comportement que *cc*.

Proposition 3 *Soit* Θ *tel que* $\vdash \Theta : ((A \to B) \to A) \to A$ *alors si* t *est un terme tel que* $t \star \alpha.\pi \succ \alpha \star u.\pi'$ *pour une certaine constante* α*, alors :*

$$\Theta \star t.\pi \succ u \star \pi$$

Dans cette propriété α joue le rôle de la constante de pile qui capture la pile π, si t la fait passer devant, suivi d'un terme u, elle efface la pile courante pour la remplacer par π.

Preuve

La démonstration est guidée par le lemme d'adéquation. Ce que l'on veut vérifier c'est que $\Theta \star t.\pi \in \{\rho | \rho \succ u \star \pi\}$. On pose donc naturellement

$$\perp\!\!\!\perp = \{\rho | \rho \succ u \star \pi\}.$$

On a $\vdash \Theta : ((A \to B) \to A) \to A$. Par le lemme d'adéquation, on en déduit $\Theta \Vdash ((A \to B) \to A) \to A$. Pour avoir notre résultat, il suffit donc que $t \Vdash (A \to B) \to A$ et $\pi \in \|A\|$. Mais on peut fixer l'interprétation des variables propositionnelles, on peut donc décider $\|A\| = \{\pi\}$. La seule question devient alors $t \Vdash (A \to B) \to A$? Pour y répondre, prenons v tel que $v \Vdash A \to B$. A t-on $t \star v.\pi \in \perp\!\!\!\perp$? Mais $t \star \alpha.\pi \succ \alpha \star u.\pi'$ où α est une constante, d'où

$$t \star v.\pi \succ v \star u.\pi'$$

Or $u \star \pi \in \perp\!\!\!\perp$ et $\|A\| = \{\pi\}$ d'où $u \Vdash A$. Choisissons maintenant une interprétation pour B, soit $\|B\| = \Pi$, $\pi' \in \|B\|$. On en déduit

$$v \star u.\pi' \in \perp\!\!\!\perp$$

puis par antiréduction :

$$t \star v.\pi \in \perp\!\!\!\perp$$

d'où $t \Vdash (A \to B) \to A$, d'où le résultat.

□

Nous allons maintenant essayer de voir quel est la spécification liée à un théorème arithmétique simple. Le travail que nous allons faire se généralise à toute les formules arithmétiques. Soit la formule

$$\mathrm{Ent}\, n = \forall X(\forall p(Xp \to Xp+1), X0 \to Xn)$$

qui signifie que n est un entier. Si Φ est une fonction récursive, on veut connaitre la spécification liée à la formule

$$\exists x \forall y[\Phi(x,y) = 0]$$

Le symbôle $=$ est une notation dont la signification est donnée par :

$$a = b \equiv \forall X(Xx \to Xy)$$

Pour que l'on puisse travailler dans Peano, il faut rajouter à cette formule le fait que x et y sont des entiers, nous travaillerons donc avec la traduction suivante de la formule :

$$F \equiv \forall x(\mathrm{Ent}\, x, \forall y(\mathrm{Ent}\, y \to \Phi(x,y) = 0) \to \bot) \to \bot$$

On s'est aussi arrangé pour supprimer le $\exists$ mais le sens reste le même.

Introduisons maintenant un opérateur très utile lorsque l'on veut faire de la réalisabilité avec des entiers, soit

$$T = \lambda f, n.(n(\lambda g.g \circ s))f0$$

On a alors la propriété suivante,

Proposition 4 *Si* $fs^n0 \Vdash X$ *alors* $Tf \Vdash Ent\, n \to X$

Il est faux que les seules termes qui ont pour type $\mathrm{Ent}\, n$ ressemblent à des entiers de Church mais l'opérateur T permet de s'y ramener. Il permet d'évaluer l'argument d'une fonction si celui-ci est un entier et ce en appel par valeur ce qui remarquable vis-à-vis de notre machine qui fonctionne en appel par noms.

Preuve

Soit $\|Yj\| = \{s^{n-j}0.\pi | \pi \in \|X\|\}$ si $j \leq n$, $\|Yj\| = \emptyset$ sinon. On a de manière évidente, $f \Vdash Y0$ et $\lambda g.g \circ s \Vdash \forall p(Yp \to Y(p+1))$. D'où si $\nu \Vdash \mathrm{Ent}\, n$,

$$\nu(\lambda g.g \circ s)f \Vdash Yn$$

et $0.\pi \in \|Yn\|$ pour $\pi \in \|X\|$ d'où

$$Tf\nu \Vdash X$$

C.Q.F.D.

□

On peut maintenant décrire la spécification de la formule F. Rappelons pour mémoire sa forme originelle

$$\exists x \forall y[\Phi(x,y) = 0]$$

Les programmes qui auront pour type F vont agir comme des joueurs toujours gagants du jeu dont les règles sont les suivantes :

– étape 1 : le programme propose un entier x
– étape 2 : le programme attend qu'on lui propose un entier y, si $\Phi(x,y) = 0$ alors le programme a gagné sinon on repasse à l'étape 1.

Formalisons cela, il faut se donner une instruction supplémantaire qui joue le rôle de l'opposant dans le jeux, pour ce faire on rajoute une constante κ dans notre langage et on lui donne la règle de réduction suivante :

$$\kappa \star s^n 0.\zeta.\pi \succ \zeta \star s^p 0.\kappa_{np}.\pi'$$

où p et π' sont décidés d'après une stratégie inconnue. Intuitivement, n correspond à la réponse du programme au moment où il laisse la main à l'opposant et ζ ce qu'il convient de faire ensuite. Les κ_{np} sont des symboles rajoutés qui mémorisent la dernière suite de coup jouée. Le résultat est le suivant :

Théorème 3 *Si* $\vdash \Theta : F$ *alors il existe* n, p *tels que* $\Phi(n,p) = 0$ *et*

$$\Theta(T\kappa) \star \pi \succ \kappa_{np}\pi'$$

Preuve

Il est naturel de poser

$$\perp\!\!\!\perp = \{\rho | \exists \pi', n, p, \text{ tels que } \Phi(n,p) = 0 \text{ et } \rho \succ \kappa_{np} \star \pi'\}$$

On a par le lemme d'adéquation :

$$\theta \Vdash \forall x(\mathrm{Ent}\, x, \forall y(\mathrm{Ent}\, y \to \Phi(x,y) = 0) \to \perp) \to \perp$$

Il suffit donc de prouver que

$$T\kappa \Vdash \forall x(\mathrm{Ent}\, x, \forall y(\mathrm{Ent}\, y \to \Phi(x,y) = 0) \to \perp)$$

D'après la proposition énoncée sur T il suffit pour cela de montrer le résultat suivant, pour tout entier n :

$$\kappa s^n 0 \Vdash \forall y(\mathrm{Ent}\, y \to \Phi(n,y) = 0) \to \perp$$

Soit ζ tel que $\zeta \Vdash \forall y(\mathrm{Ent}\, y \to \Phi(n,y) = 0)$, comme

$$\kappa \star s^n 0.\zeta.\pi \succ \zeta \star s^p 0.\kappa_{np}.\pi$$

il suffit de montrer que l'on a

$$\zeta \star s^p 0.\kappa_{np}.\pi \Vdash \perp\!\!\!\perp$$

Il faut maintenant distinguer deux cas

– Si $\Phi(n,p) = 0$,

$$|\Phi(n,p) = 0| = |\forall X(X\Phi(n,p) \to X0)| = |\forall X(X \to X)|$$

Or $\kappa_{np} \Vdash \perp$ d'où $\zeta s^p 0 \kappa_{np} \Vdash \perp$, d'où le résultat.

– Si $\Phi(n,p) \neq 0$,

$$|\Phi(n,p) = 0| = \Lambda \to \bot$$

alors $\zeta s^p 0 \Vdash \bot$, Le résultat est immédiat

□

Ce résultat se généralise à tout les théorèmes arithmétiques. En effet, partant d'un théorème arithmétique, on se ramène à une forme où tous les quantificateurs se trouvent à gauche. La spécification associée sera à nouveau le joueur d'un jeu similaire au précédant mais dont le nombre d'étape dépendra du nombre d'alternance de quantificateurs $\forall$ et $\exists$.

Conclusion

L'outil de la réalisabilité permet donc parfois de résoudre le problème de la spécification de manière très simple. Des travaux ont aussi été réalisés pour comprendre la spécification liée aux axiomes de la théorie des ensembles et à l'axiome du choix. On peut donc mieux voir au travers de ces résultats à quel type de comportement on peut contraindre un programme via son typage.

Références

[1] J.-L. Krivine, *Une preuve formelle et intuitionniste du théorème de complétude de la logique classique*, 1996, http ://www.pps.jussieu.fr/ krivine/articles/completude.pdf

[2] J.-L. Krivine, *Une preuve formelle et intuitionniste du théorème de complétude de la logique classique (suite)*, 1996

[3] J.-L. Krivine, *Lambda-calcul types et modèles*, Masson, Paris (1990)

[4] J.-L. Krivine, *Two talks about specification and objects*, 2002, http ://www.pps.jussieu.fr/ krivine/articles/Agay.pdf

[5] J.-L. Krivine, *Dependent choice, 'quote' and the clock*, 2002, http ://www.pps.jussieu.fr/ krivine/articles/quote.pdf

[3] G.Griffin, *A formulae-as-types notion of control*