Les preuves de la normalisation du lambda-calcul du premier et second ordre. Liens avec la consistance de l'arithmétique.
Proposé par Giuseppe Longo

Cédric Faure Axelle Ziegler

27 juin 2002

Un des domaines de recherche les plus actifs dans la logique actuelle est la preuve assistée par ordinateur, qui vise à automatiser et à formaliser autant que possible les mécanismes des démonstrations mathématiques. On sait pourtant, grâce aux deux théorèmes d'incomplétude de Gödel, qu'aucun système formel fondé sur l'arithmétique de Peano n'est complet, et même mieux (ou pire), qu'un tel systême ne peut démontrer sa propre cohérence. Ces théorèmes montrent donc qu'il est impossible de réunir de façon cohérente et exhaustive l'ensemble des mathématiques en un unique langage formel fondé sur ceux que l'on utilise actuellement, et toutes les méthodes de preuve automatique tombent sous les limites de ces théorèmes.

On connaît pourtant des résultats qui impliquent la cohérence de l'arithmétique du premier, voire du deuxième ordre, mais qui ne sont évidemment pas démontrables dans ces cadres.Nous allons tenter de nous intéresser à deux de ces résultats, les normalisations fortes du système T et du système F, qui codent respectivement $\mathcal{PA}_1$ et $\mathcal{PA}_2$ à l'aide du lambda-calcul. Ces résultat ont un intérêt *per se*, et pas seulement dans le cadre de la démonstration de la cohérence de l'arithmétique. En effet, le système F en particulier est un langage très expressif, qui permet de coder de façon effective de nombreuses structures utilisées dans les langages de programmation, et la preuve de sa normalisation forte fournit des résultats intéressants sur ces langages. Par ailleurs, ces systèmes permettent de coder, et donc d'automatiser, de façon effective les preuves dans $\mathcal{PA}$.

Afin de mieux comprendre le mécanisme des preuves, il faut toutefois faire une petite digression sur le concept de preuve, et en particulier sur les limites de l'induction, qui est la méthode de preuve formalisée dans l'arithmétique de Peano. L'induction, en effet est un outil remarquablement puissant, qui permet d'effectuer des démonstrations finies et facilement automatisables. Toutefois, le premier théorème d'incomplétude nous assure que c'est une technique de preuve incomplète, et qu'on ne peut faire l'économie de preuves "infinies". Ces preuves "infinies" sont souvent difficiles à automatiser car faisant largement appel à l'intuition et à l'imagination humaine (un exemple caractéristique est la démonstration par Gauss de la valeur de la somme des n premiers nombres.), mais il est toutefois parfaitement possible de décortiquer aussi mécaniquement que possible leur fonctionnement, dans un but de formalisation. Parmi ces méthodes de preuves, la structure la plus classique est celle de la "preuve par prototype" : il s'agit alors de raisonner non plus sur les éléments à propos desquels on veut démontrer quelque chose, mais sur leur type, et sur les caractéristiques connues comme communes à tous les éléments ayant ce type. Cette façon de procéder est de très loin la plus fréquente en informatique : ainsi, qui songerait à énumérer toutes les matrices de GL_n pour démontrer qu'elles ont un déterminant non nul ? Par ailleurs, cette méthode à un avantage cognitif certain pour un être humain, qui est que l'on peut "découvrir" la formule au cours de la démonstration, alors qu'il est indispensable de savoir précisément ce qu'on souhaite démontrer par in-

duction. Il faut toutefois noter que, pour puissante qu'elle est, cette méthode de preuve n'est pas codable dans $\mathcal{PA}$.

À la lumière de ces différentes remarques, nous allons donc essayer de montrer le principe et l'intérêt de ces résultats et de leur preuves, afin de différencier ce que l'on peut effectivement coder et automatiser et ce qui appartient à des codes d'ordre supérieur.

1 Lambda-calcul du premier ordre, Système T

On va s'intéresser en premier lieu au système T, une extension simple du lambda-calcul classique formalisée par Gödel dans l'espoir de démonter la cohérence de l'arithmétique,

a) Définitions

Le lambda-calcul classique

Nous allons tout d'abord rappeler, essentiellement afin de fixer les formalismes, la syntaxe et les règles de réduction du lambda-calcul classique. On se donne tout d'abord un ensemble $\mathcal{V}$ de variables, que l'on notera ici $x, y, z, \ldots$. On peut alors définir récursivement l'ensemble $\mathcal{T}$ des lambda-termes de la façon suivante :

Définition des lambda-termes. *L'ensemble $\mathcal{T}$ des lambda-termes est le plus petit ensemble tel que*

- $x \in \mathcal{V} \Rightarrow x \in \mathcal{T}$
- $(t \in \mathcal{T} \wedge u \in \mathcal{T}) \Rightarrow (tu) \in \mathcal{T}$
- $t \in \mathcal{T} \Rightarrow \forall x \in \mathcal{V}, \lambda x.t \in \mathcal{T}$
- $t \in \mathcal{T} \Rightarrow (t) \in \mathcal{T}$

On définit sur cet ensemble de termes une règle de réduction.

La β-réduction. $(\lambda x.tu) \rightarrow t[x := u]$

Où $t[x := u]$ dénote le lambda-terme t, où l'on remplace toutes les occurences de x par u. Cette définition informelle de la substitution fait évidemment abstraction de beaucoup de problèmes. Pour des définitions plus rigoureuses, voir l'annexe A.

On va enfin définir la notion de normalisation pour le lambda-calcul :

La normalisation. *On dit qu'un lambda-terme est normal s'il n'y a aucune réduction partant de ce terme.*

Une forme normale d'un lambda-terme est un lambda-terme normal en lequel ce terme se réduit. La propriété de Church-Rösser (ou confluence) nous assure que lorsqu'elle existe, cette forme est unique.

Un lambda-terme est dit faiblement normalisant si il a une forme normale, fortement normalisant si toutes les réductions partant de ce terme terminent en cette forme normale.

On dit qu'un système basé sur le lambda-calcul est faiblement(fortement) normalisant, si tous les termes qui le composent le sont.

Le lambda-calcul classique n'est pas fortement, ni même faiblement normalisant : un terme comme $\Omega = (\lambda x.xx)(\lambda x.xx)$ n'a que des réductions infinies.

Le typage et le codage des entiers.

Le lambda-calcul classique tel que nous venons de le définir est un outil remarquable pour l'étude des langages de programmation. Pour notre usage, nous allons lui adjoindre des règles de typage, puis quelques fonctions extèrieures, afin d'obtenir le système T.

Le typage du lambda-calcul classique se fait à partir d'un contexte de typage, qui contient essentiellement le type des variables libres, et de règles, que nous allons exposer ici. On note de façon classique $u : F$ pour signifier que le lambda-terme u est de type F. Parmi les diffrentes solutions pour typer le lambda-calcul, on choisit celle consistant à attribuer un type fixe aux variables. Si cette méthode comporte quelques inconvénients, entre autre d'exiger une grande prudence lors des α-renommages, elle correspond parfaitement aux systèmes que l'on veut coder. On note alors x_F une variable x de type F. On a les règles suivantes :

Les règles de typage.

$$\frac{}{x_F : F}$$

$$\frac{u : G}{\lambda x_F.u : F \Rightarrow G}$$

$$\frac{u : F \Rightarrow G \qquad v : F}{uv : G}$$

Ces règles extrêmement simples suffisent pour rendre le lambda-calcul typé normalisant : en effet les termes qui "bouclent" comme Ω ne sont pas typés. On s'intéressera plus en détail à ces propriétés de normalisation ultérieurement. Par ailleurs, comme nous le verrons plus loin, ces règles sont en bijection avec les règles de déduction logique.

Pour finir le codage de l'arithmétique en lambda-calcul, il nous manque encore quelques outils. Ces outils vont nous être apportés par le système T de Gödel, qui va nous permettre de coder $\mathcal{PA}_1$.

Dans le système T, on a deux types de base `Int` et `Bool` à partir duquel on peut comme précédemment construire des types composés. On ajoute par ailleurs trois symboles de constantes : 0, T et F respectivement de type `Int`, `Bool` et `Bool` et trois opérateurs R, S et D.

S est tel que si t est de type `Int`, St est de type `Int` également. Intuitivement, S représente la fonction successeur, et 0 la constante 0.

R est tel que si u, v et t sont de types U, $U \to$(`Int`$\to U$) et `Int` alors $Ruvt$ est de type `Int`. (U est un type quelconque). Intuitivement, R est un opérateur de récursion.

D est tel que si u, v et t sont de types U, U, `Bool` alors Duvt est de type `U`. Intuitivement, D est un opérateur de branchement, T et F représentant repectivement la valeur Vrai et la valeur Faux.

On ajoute également 4 nouvelles règles de réduction, afin que nos opérateurs aient le sens que l'on veut leur donner :

Règles de réduction du système T.

$Ruv0 \to u$
$Ruv(St) \to v(Ruvt)$
$DuvT \to u$
$DuvF \to v$

On va enfin coder l'addition et la multiplication, qui sont deux opérateurs infixes binaires tels que si u et v sont de type `Int` alors $u(\underline{+}/\underline{\times})v$ est de type `Int`. Pour ce faire on définit :

$x \underline{+} y = Rx\ (\lambda n_{Int}.\lambda m_{Int}.Sn)\ y$
$x \underline{\times} y = R\ 0\ (\lambda n_{Int}.\lambda m_{Int}.(n \underline{+} x))\ y$

b) Le codage de $\mathcal{PA}_1$ en système T, lien entre normalisation et cohérence

On peut coder tous les élements de la théorie de Peano à l'aide du système T. On procède en identifiant les formules à des types, et un terme typé à une preuve. Ainsi, une réduction d'un lambda-terme correspond à l'élimination d'un détour de la preuve. Afin d'éclairer ce concept, nous allons donner un exemple en logique classique : On se donne deux axiomes F et $F \Rightarrow G$, et on veut montrer G. La preuve est de cette forme :

$$\frac{F \qquad F \Rightarrow G}{G}$$

Le fait que F et $F \Rightarrow G$ soient des axiomes se traduit par l'existence d'un terme de type F et d'un terme de type $F \Rightarrow G$, que l'on notera u et v. uv est alors une preuve de G, car c'est un terme de type G.

En plus des types ajoutés par le système T, on a besoin d'un type $\bot$, qui représente les formules contradictoires. Si F est une formule, on peut alors coder $\neg F$ par $F \Rightarrow \bot$. Si l'on veut pouvoir coder $\mathcal{PA}_1$, on a besoin des règles de déduction de la logique classique, et en particulier,

$$\neg\neg F \Rightarrow F$$

On ajoute donc un opérateur C tq

$$\frac{u : \neg\neg F}{Cu : F}$$

et

$$Cuv \to C(\lambda k.u(\lambda f.k(fv))) \qquad C(\lambda k.u) \to u(k \notin fv(u))$$

Pour coder les axiomes de Peano, on ajoute un terme τ_0 de type $0 = 0$, et les axiomes sont alors de simples règles de transformations de formules.

Enfin, le schéma d'induction se code en ajoutant une nouvelle règle de typage :

$$\frac{u : F[i := 0] \qquad v : F[i := j] \to F[i := Sj]}{Ruvt : F[i := t]}$$

Toutes ces règles conservent la normalisation et la confluence de notre système.

Alors, si $\mathcal{PA}_1$ n'est pas cohérente, $\mathcal{PA}_1$ démontre une contradiction, donc il existe un terme de type $\bot$. Ce terme a, par la normalisation forte, une forme normale, qu'on ne peut exhiber. Ce raisonnement nous prouve que la preuve de la normalisation forte se fait nécessairement en dehors de $\mathcal{PA}_1$.

Preuve de la normalisation forte

La preuve n'est pas du tout une preuve intuitive : elle se fait par induction sur les types : en effet, il est hors de question de faire une induction du premier ordre sur les termes, puisque on aurait une contradiction avec le théorême de Gödel. On doit donc utiliser une induction plus forte, et en particulier avoir une démonstration au moins d'ordre 2.

On va en fait définir pour chaque type T un ensemble RED_T, ensemble des termes "réductibles" de type T, que l'on construit par induction sur les types de la façon suivante :

Définition.

- *si t est de type T atomique, alors $t \in RED_T$ ssi t est fortement normalisant*
- *si t est de type $U \to V$, $t \in RED_{U \to V}$ ssi $\forall u \in RED_U$, $tu \in RED_V$. Cette définiton utilise l'axiome de compréhension et ne peut pas être codée dans $\mathcal{PA}_1$. En effet, on a besoin de l'axiome de compréhension pour coder $u \in RED_U$.*
- *Un terme est dit **neutre** si ce terme n'est pas une abstraction, i.e s'il est de la forme x, où x est une variable, ou de la forme tu.*

On va montrer chercher à montrer les propriétés suivantes sur les termes réductibles :

Propriétés des termes réductibles.
(P1) : Si $t \in RED_T$, t est fortement normalisant
(P2) : Si $t \in RED_T$ et $t \to t'$ alors $t' \in RED_T$
(P3) : Si t est neutre et que toute réduction d'un rédex de t donne un terme t', $t' \in RED_T$, alors $t \in RED_T$

Nous allons montrer ces propriétés par induction sur les types concernés. On peut remarquer que, même si c'est (P1) que nous cherchons à démontrer, (P2) et (P3) nous sont indispensables pour pouvoir construire les éléments de RED_T. La charge inductive utilisée est particulièrement lourde, et comme on l'a vu plus haut, ne peut être éxprimée par des formules arithmétiques d'ordre 1. C'est la lourdeur et la complexité logique de cette charge inductive qui permet à notre démonstration de fonctionner sans contredire le théorême de Gödel.

Démonstration.
$\underline{1^{er} \text{ cas}}$: T est un type atomique
(P1) est triviale par définition de RED_T
(P2) Si $t \in RED_T$ alors t est fortement normalisant. Or $t \to t'$, donc t' est clairement également fortement normalisant. D'où $t \in RED_T \Rightarrow t' \in RED_T$, ce qui est (P2)
(P3) Toute réduction partant de t passe par un des t', qui sont tous fortement normalisants, donc t est fortement normalisant également, et on a (P3).

$\underline{2^{eme} cas}$: T est un type flèche, de type $U \to V$
(P1) : Soit x_U une variable de type U. En appliquant (P3), $x_U \in RED_U$. Donc si $t \in RED_{U \to V}$, $tx_U \in RED_V$, donc d'après (P1) tx_U est fortement normalisant. Or une chaîne de réduction infinie de t induit une chaîne de réduction infinie de tx_U, donc si tx_U fortement normalisant alors t est fortement normalisant. On a donc démontré (P1)
(P2) Soit $u \in RED_U$, $tu \in RED_V$ et $tu \to tu'$. D'après (P2), $tu' \in RED_V$. Donc $t' \in RED_{U \to V}$ ce qui montre (P2)
(P3) Soit t neutre de type $U \to V$, tel que pour tout $t \to t'$, $t' \in RED_{U \to V}$. Soit $u \in RED_U$, d'après (P1), u est fortement normalisant. On peut donc raisonner par récurrence sur la longueur de la plus longue réduction partant de u, afin de montrer que $tu \in RED_V$ Une réduction de tu donne :

- $t'u$, où $t \to t'$ Or $t' \in RED_{U \to V}$, donc $t'u \in RED_V$
- tu', où $u \to u'$. D'après (P2), $u' \in RED_U$ et la plus grand réduction partant de u' est plus courte que celle partant de u. Par hypothèse de récurrence, on a donc $tu' \in RED_V$
  Comme t est neutre, tu n'est pas un rédex, et il n'y a donc pas d'autres possibilités de réduction. On a donc que tu est neutre de type V et ne se réduit qu'en terme réductibles. Donc, d'après (P3), $tu \in RED_V$, d'où $t \in RED_{U \to V}$ et d'où (P3).

□

Lemme d'abstraction.
Si $\forall u \in RED_U$, $v[x_U := u] \in RED_V$, *alors* $\lambda x_U.v \in RED_{U \to V}$

Ce lemme complète la propriété 3 pour les termes non neutres.

Démonstration. $u \in RED_U$, donc u fortement normalisant. $x_U \in RED_U$, donc $v[x_U := x_U]$, c'est à dire v est fortement normalisant.
On peut donc raisonner par récurrence sur la somme de la longueur de la plus longue réduction partant de u et de v, afin de montrer $(\lambda x_U.v)u \in RED_V$.
On peut réduire ce terme en :

- $v[x_U := u]$, qui appartient à RED_V par hypothèse.
- $\lambda x_U.v'.u$, où $v \to v'$. La plus longue réduction partant de v' est plus courte que celle partant de v, ce terme est donc réductible par hypothèse de récurrence.
- $\lambda x_U.v.u'$, où $u \to u'$. Pour les mêmes raisons que plus haut, ce terme est réductible par hypothèse de récurrence.

On a donc un terme qui ne se réduit qu'en des termes réductibles et qui est neutre, d'où $(\lambda x_U.v)u \in RED_V$. Et par définition, $\lambda x_U.v \in RED_{U \to V}$ □

On peut à présent démontrer le résultat qu'on recherche :

Théorême de normalisation forte.
Tous les termes typés sont réductibles, donc fortement normalisants.

On va démontrer une propriété plus forte :

Lemme. *Si t est un terme typé de variables libres $x_{1_{U_1}}, \ldots, x_{n_{U_n}}$, si $u_1, \ldots, u_n$ sont réductibles de type $U_1, \ldots, U_n$, alors $t[\underline{x} := \underline{u}]$ est réductible.*

Démonstration. Par induction sur T :

- t est une variable : Alors $t[x := u] = u$ d'où $t[x := u] \in RED_U$.
- t = vw : Alors par hypothèse d'induction, $w[\underline{x} := \underline{u}] et v[\underline{x} := \underline{u}]$ sont réductibles. Donc, par définition $t[\underline{x} := \underline{u}] = v[\underline{x} := \underline{u}]w[\underline{x} := \underline{u}]$ est également réductible.
- t=$\lambda y.w$ de type $V \to W$. Par hypothèse d'induction, $w[\underline{x} := \underline{u}][\underline{y} := \underline{v}]$ est réductible pour tout v réductible. Donc $t[\underline{x} := \underline{u}] = \lambda y.w[\underline{x} := \underline{v}]$ est réductbile

□

2 Lambda-calcul du deuxième ordre : le système F

a) Définitions

Le système F est une extension du lambda-calcul typé, inventée par Jean-Yves Girard (cf. [Gir]) permettant d'introduire des variables de type, donc de quantifier sur les types et de coder $\mathcal{PA}_2$.

Les ajouts

On se donne tout d'abord un ensemble de variables de type que l'on notera X, Y, Z,...En plus du type flèche déjà défini on définit un nouveau type $\forall X.V$ où X est une variable de type et V un type.

On a les même termes que précedemment, auxquels on ajoute $\Lambda X.M$, "l'abstraction sur les types" où X est une variable de type et M un terme.

Typage et Réduction

On garde pour les anciens termes les même règles de typage, auxquelles on ajoute les règles suivantes :

$$\frac{v : V}{\Lambda X.v : \forall X.V}$$

(Si X n'est libre dans le type d'aucune variable libre de v)

$$\frac{m : \forall X.V}{mU : V[X := U]}$$

(avec U type quelconque)

On garde également les réductions du lambda-calcul classique, auxquelles on ajoute :

$(\Lambda X.v)U \to v[X := U]$

On appelle parfois le système F "lambda-calcul polymorphe", ce qui s'explique assez facilement : en effet en système T comme en lambda-calcul classique, le typage que nous avions décrit ne permettait pas de définir une fonction sur plusieurs types : ainsi, il était nécessaire de définir une identité pour chaque type. En système F, la fonction $\Lambda X.\lambda x_X.x_X$ permet de définir une égalité sur n'importe quel type, tout en gardant une information forte sur le type des fonctions.

Ce système élargit de façon considérable le pouvoir expressif du lambda-calcul, en permettant de définir la plupart des types usuels utilisés dans les langages de programmation (listes,arbre,...), ainsi que comme nous le verrons plus loin les entiers, que nous pourrons ici directement coder dans le langage, sans extension.

Quelques remarques : première intuition de la généricité

Le type $\forall X.V$ introduit *a priori* une circularité dans la définition des types : ainsi, un type comme V[X :=$\forall X.V$] est parfaitement valide. Pour que l'on puisse définir des fonctions sur de tels types, il est nécessaire qu'une fonction fasse à peu près la même chose sur tous les types. Cette remarque, sur laquelle nous reviendrons, est fondamentale pour la compréhension du fonctionnement de la preuve de la normalisation forte du système F.

b) Pouvoir expressif

L'arithmétique de Heyting

Ici, au lieu de coder $\mathcal{PA}_2$ en système F, on va tenter de coder $\mathcal{HA}_2$, l'arithmétique de Heyting du second ordre. Ces deux systèmes ont de toutes façon la même "force", c'est à dire qu'elles définissent les mêmes fonctions. On peut en effet établir une bijection entre les formules de $\mathcal{PA}$ et celles de $\mathcal{HA}$, qui vérifient par ailleurs de la même façon le théorème de Gödel. Il existe d'ailleurs une méthode de traduction de l'arithmétique intuitionniste vers l'arithmétique classique, formalisée par Gödel lui-même dans les années 50.

Les entiers

On va coder directement en sytême F le type `Int` ainsi que 0 et S. On va en fait interpréter un entier n comme une fonction qui à un type U et à une fonction $f : U \to U$ associe f^n. Les entiers sont donc du type : $\forall X.(X \to X) \to (X \to X)$. On a alors :

$$O = \Lambda X.\lambda x_X.\lambda y_{X\to X}.x \qquad St = \Lambda X.\lambda x_X.\lambda y_{X\to X}.y(tXxy).$$

On peut de la même façon coder un opérateur R ayant la même sémantique que celui du système T. Nous épargnerons cependant cette formule au lecteur, et nous considérerons cet opérateur comme codé.

Codage des déductions

On va coder chaque formule par un type. Un terme ayant ce type nous fournira une preuve de la formule. On définit ainsi la correspondance C qui à chaque formule associe un type.

1. $C(a = b) = S$ où S est un type fixé de F, contenant au moins un terme clos. On définit donc arbitrairement un type pour chaque terme de la forme $(a = b)$ exprimé dans le langage.
2. $C(a \in X) = X$ où X est à la fois une variable d'ordre 2 de $\mathcal{HA}$ et une variable de type de F.
3. $C(A \Rightarrow B) = C(A) \to C(B)$
4. $C(\forall x\ A) = C(A)$ avec x une variable du 1^{er} ordre dans $\mathcal{HA}$.
5. C($\forall X\ A$) = $\forall X.C(A)$ c'est à dire qu'une quantification sur les ensembles se code en une abstraction sur les types.

On peut facilement définir les connecteurs logiques manquant à partir de ceux que l'on a déjà définis.

Il nous faut également coder les différentes règles de déduction de $\mathcal{HA}$.

1.
$$\frac{}{A_i \vdash A_i}$$
se code en x_i où x_i est une variable libre de type $C(A_i)$.

2.
$$\frac{A \vdash B}{A \Rightarrow B}$$
se code en $\lambda x.u$ où x est une variable de type C(A) et u est le code de la preuve de B à partir de A.

3.
$$\frac{A \Rightarrow B,\ A}{B}$$
se code en uv où u est le code de la preuve de $A \Rightarrow B$ et v celui de la preuve de A.

4.
$$\frac{A}{\forall x A}$$
n'a pas de code car $\mathrm{C}(A) = \mathrm{C}(\forall x A)$

5. De même pour
$$\frac{\forall x A}{A[x := t]}$$
pour les mêmes raisons.

6.
$$\frac{A}{\forall X A}$$
se code en $\Lambda X.u$ où u est le code de la preuve de A.

7.
$$\frac{\forall X A}{A[X := x.C]}$$
se code en en uv où u est le code de la preuve de $\forall X A$ et v = C(C).

On n'a pas besoin de coder de façon spécifique l'axiome de compréhension et l'induction sur les entiers : il suffit de remarquer qu'on a dans $\mathcal{HA}$ déjà implicitiment l'axiome de compréhension : Si on prend la formule valide :

$$\forall X \exists Y. \forall x (x \in X \Leftrightarrow x \in Y)$$

on obtient

$$\exists Y. \forall x (C \Leftrightarrow x \in Y)$$

ce qui est exactement l'axiome cherché. De la même façon, le schéma d'induction est une formule valide sur les entiers, il n'y a donc pas besoin d'ajouter de règle de déduction : on peut définir en $\mathcal{HA}_2$ une formule

$Nat(x) = \forall X.(0 \in X \Rightarrow \forall y.(y \in X \Rightarrow Sy \in X) \Rightarrow x \in X)$. Cette formule caractérise naturellement les élements auxquels on peut appliquer un schńa d'induction, et est naturellement vérifiée par tout élément de la forme $S^n(0)$

On peut montrer que dans ce cadre de codage, toute preuve dans $\mathcal{HA}_2$ peut être codé dans F, en particulier que l'on peut coder les axiomes, et les fonctions totales de $\mathcal{HA}_2$ dans F. Cela irait, là encore, bien loin de notre sujet. Nous nous contenterons donc de remarquer que le codage que nous venons de donner code bien le système de déduction de la logique du second ordre intuitionniste, et que l'on peut arriver à coder de façon complète $\mathcal{HA}_2$ dans ce système. Pour une preuve détaillée de ce résultat, se référer à [GLT]

Lien entre normalisation et cohérence

Reste à prouver à présent que ce codage "suffit" bien, c'est à dire que la normalisation forte du système F implique bien la cohérence de $\mathcal{HA}_2$

Si l'on suppose $\mathcal{HA}_2$ incohérente, on a un terme u qui prouve, par exemple, $\forall$ P P. On aurait alors un terme en forme normale de type $\forall X.X$. u est alors de la forme $\Lambda X.v$ où v est un terme de type X. Or comme X est une variable de type, v ne peut être de type X qu'en contenant une variable libre de type X, ce qui est impossible par définition. On n'a donc pas de terme u prouvant $\forall X.X$ et $\mathcal{HA}_2$ est bien cohérente.

c) La preuve de la normalisation forte du système F

Intuition de la structure de la preuve. Un exemple de généricité.

On aimerait intuitivement pouvoir réutiliser la méthode de preuve appliquée au lambda-calcul classique. Toutefois, dans le cas de l'ordre deux, une simple induction sur les types va échouer pour les raisons de circularité évoquées plus haut : ainsi une définition naïve de $RED_{\forall X.P}$ comme l'ensemble des u tels que pour tout type G $uG \in RED_{P[X:=G]}$ ne peut pas fonctionner, car $P[X := G]$ n'est pas toujours une sous formule de $\forall X.P$. C'est donc le caractère imprédicatif du système F et de la logique d'ordre deux qui entre en jeu.

Pour échapper à cette difficulté, on définit des "candidats de réducibilité", qui seront des ensembles de termes d'un type donné vérifiant (P1), (P2) et (P3). Il est clair que le "vrai" ensemble RED fera partie de ces candidats. On peut alors définir la réducibilité pour $\forall X.P$ uniquement par rapport aux candidats de réducibilité de type U. Ainsi, on peut montrer que l'identité universelle $I = \Lambda X.\lambda x_X.x$ est réductible : elle est réductible si et seulement si pour tout type U et pour tout candidat de réducibilité $\mathcal{R}$ de U, IU est réductible de type $U \to U$. Montrer cela revient à montrer que $\forall u \in \mathcal{R}$, $IUu \in \mathcal{R}$ ce que l'on peut démontrer à partir de (P3). Il est toutefois clair que ce raisonnement fait entrer en jeu un argument d'ordre 3 : en effet, on a quantifié sur des sous-ensembles d'un ensemble (pour tout type et pour

tout candidat de réducibilité). Comme prévu par le théorème de Gödel, cette preuve va au-delà de l'ordre deux.

On voit dans l'exemple précédent que le raisonnement n'utilise aucune information sur U. Tout se passe comme si l'abstraction de types était suffisament uniforme pour que l'on puisse déterminer la réducibilité sans aucune information sur son arguement. Cette uniformité, qui s'exprime par le théorême de généricité(cf. [LMS], est fondamentale.

On peut à présent en venir à la preuve elle-même, qui se fait nécessairement au-delà de l'ordre deux. En fait, on ne peut pas écrire de formule du second ordre traduisant le fait que t est réductible de type T.

Définitions préliminaires

On définit comme précédemment un terme neutre comme n'étant pas une abstraction.

Définition d'un candidat de réducibilité.

*Un **candidat de réducibilité** est un ensemble $\mathcal{R}$ de termes de type U vérifiant (P1), (P2) et (P3).*

En particulier, d'après (P3) un tel ensemble n'est jamais vide car il contient toutes les variables de type U. On peut également remarquer que l'ensemble des termes fortment normalisant de type U est une candidat de réducibilité.

On peut définir un ensemble $\mathcal{R} \to \mathcal{T}$ par

$$t \in \mathcal{R} \to \mathcal{T} \text{ ssi } u \in \mathcal{R} \Rightarrow tu \in \mathcal{T}.$$

On peut réutiliser les arguments utilisés dans la première partie pour montrer que si $\mathcal{R}$ et $\mathcal{T}$ sont des candidats de réducibilité de types respectifs U et V, alors $\mathcal{R} \to \mathcal{T}$ est un candidat de réducibilité de type $U \to V$.

On va maintenant montrer une série de lemmes qui nous amèneront à la preuve finale.

Définition de RED

Pour un type T de variables libre $\underline{X}$, pour $\underline{U}$ une séquence de types de la longueur correspondantes, pour $\underline{\mathcal{R}}$ suite de candidats de réduciblilité pour chacun des types U_i, on peut définir un ensemble $RED_T[\underline{X} := \underline{\mathcal{R}}]$ de termes de type $T[\underline{X} := \underline{U}]$ par :

1. $RED_{X_i}[\underline{X} := \underline{\mathcal{R}}] = \mathcal{R}_i$
2. $RED_{F \to G}[\underline{X} := \underline{\mathcal{R}}] = RED_F[\underline{X} := \underline{\mathcal{R}}] \to RED_G[\underline{X} := \underline{\mathcal{R}}]$
3. $RED_{\forall Y.F}[\underline{X} := \underline{\mathcal{R}}]$ est défini par $t \in RED_{\forall Y.F}[\underline{X} := \underline{\mathcal{R}}]$ ssi $\forall V$, $\forall \mathcal{S}$ tq V est un type et $\mathcal{S}$ un candidat de réducibilité de ce type, $tV \in RED_F[\underline{X} := \underline{\mathcal{R}}, Y := \mathcal{S}]$ C'est ici que la généricité et l'ordre 3 interviennent, comme nous l'avons expliqué au début de cette partie.

Lemme 1. *$RED_T[\underline{X} := \underline{\mathcal{R}}]$ est un candidat de réducibilité de type T.*

Démonstration. On peut raisonner par induction sur T.

- Si T est un type atomique $RED_T[\underline{X} := \underline{\mathcal{R}}]$ est RED_T
- Si T est une variable de type, le lemme est vrai par définition
- Si T est de la forme $F \to G$, c'est une candidat de réducibilité d'après la définition de $RED_F[\underline{X} := \underline{\mathcal{R}}] \to RED_G[\underline{X} := \underline{\mathcal{R}}]$
- Si T est de la forme $\forall Y.F$.
  - (P1) Si $t \in RED_T[\underline{X} := \underline{\mathcal{R}}]$, pour tout type V et pour tout candidat de réducibilité $\mathcal{S}$, $tV \in RED_F[\underline{X} := \underline{\mathcal{R}}, Y := \mathcal{S}]$ par hypothèse d'induction, tV est fortement normalisant, donc t l'est également.
  - (P2) Si $t \in RED_T[\underline{X} := \underline{\mathcal{R}}]$, pour tout type V et pour tout candidat de réducibilité $\mathcal{S}$, $tV \in RED_F[\underline{X} := \underline{\mathcal{R}}, Y := \mathcal{S}]$. Si $t \to t'$, $tV \to t'V$ par hypothèse d'induction, $t'V \in RED_F[\underline{X} := \underline{\mathcal{R}}, Y := \mathcal{S}]$ donc $t' \in RED_T[\underline{X} := \underline{\mathcal{R}}]$.
  - (P3) Soit t neutre, et tout t' tel que $t \to t'$ dans $RED_T[\underline{X} := \underline{\mathcal{R}}]$. Soit V et $\mathcal{S}$ un type et un candidat de réducibilité de ce type. Une réduction dans tV donne un terme t'V puisque t est neutre, et t'V est dans $RED_F[\underline{X} := \underline{\mathcal{R}}, Y := \mathcal{S}]$, puisque t' y est. Par hypothèse d'induction, tV est dans $RED_F[\underline{X} := \underline{\mathcal{R}}, Y := \mathcal{S}]$ et donc $t \in RED_T[\underline{X} := \underline{\mathcal{R}}]$

□

On veut à présent vérifier que nos ensembles $RED_T[\underline{X} := \underline{\mathcal{R}}]$ ont un bon comportement vis à vis de la substitution, c'est à dire :

Lemme 2. $RED_{T[Y:=V]}[\underline{X} := \underline{\mathcal{R}}] = RED_T[\underline{X} := \underline{\mathcal{R}}, Y := RED_V[\underline{X} := \underline{\mathcal{R}}]]$

Démonstration. Par induction sur T à nouveau :

1. T est un type atomique il n'y a rien à prouver, puisqu'il n'y a pas de substitution dans T
2. T est une variable de type. Si $T \neq Y$, il n'y a rien à prouver. Si $T = Y$, on a $RED_T[\underline{X} := \underline{\mathcal{R}}, Y := RED_V[\underline{X} := \underline{\mathcal{R}}]] = RED_V[\underline{X} := \underline{\mathcal{R}}] = RED_{T[Y:=V]}[\underline{X} := \underline{\mathcal{R}}]$
3. T est de la forme $F \to G$: $RED_{T[Y:=V]}[\underline{X} := \underline{\mathcal{R}}]$
   $= RED_{F[Y:=V]}[\underline{X} := \underline{\mathcal{R}}] \to RED_{G[Y:=V]}[\underline{X} := \underline{\mathcal{R}}]$
   $= RED_F[\underline{X} := \underline{\mathcal{R}}, Y := RED_V[\underline{X} := \underline{\mathcal{R}}]] \to RED_G[\underline{X} := \underline{\mathcal{R}}, Y := RED_V[\underline{X} := \underline{\mathcal{R}}]]$
   $= RED_{F \to G}[\underline{X} := \underline{\mathcal{R}}, Y := RED_V[\underline{X} := \underline{\mathcal{R}}]]$
   $= RED_T[\underline{X} := \underline{\mathcal{R}}, Y := RED_V[\underline{X} := \underline{\mathcal{R}}]]$
   Le cas restant se montre selon le même principe.

□

On va maintenant prouver que nos $RED_T[\underline{X} := \underline{\mathcal{R}}]$ vérifient le lemme suivant, qui nous assure que l'on pourra "remonter" vers les abstractions sur les types :

Lemme 3. *Si pour tout V, $\mathcal{S}$ où $\mathcal{S}$ est un candidat de réducibilité de V, $u[Y := V] \in RED_W[\underline{X} := \underline{\mathcal{R}}, Y := \mathcal{S}]$, alors $\Lambda Y.u \in RED_{\forall Y.W}[\underline{X} := \underline{\mathcal{R}}]$*

Démonstration. $\Lambda Y.u \in RED_{\forall Y.W}[\underline{X} := \underline{\mathcal{R}}]$ si et seulement si pour tout V, $\mathcal{S}$ où $\mathcal{S}$ est un candidat de réducibilité de V, $(\Lambda Y.u)V \in RED_W[\underline{X} := \underline{\mathcal{R}}]$. Si on raisonne par récurrence sur la plus longue réduction partant de u, on a le résultat de façon immédiate grâce à (P3) : toute réduction de $(\Lambda Y.u)V$ donne un élement de $RED_W[\underline{X} := \underline{\mathcal{R}}]$, donc $(\Lambda Y.u)V \in RED_W[\underline{X} := \underline{\mathcal{R}}]$ □

Il nous reste à montrer que nos ensembles se comportent bien vis à vis de l'application universelle, ce qui est quasi immédiat grâce au lemme 2 :

Lemme 4. *Si $t \in RED_{\forall Y.W}[\underline{X} := \underline{\mathcal{R}}]$, alors $tV \in RED_{W[Y:=V]}[\underline{X} := \underline{\mathcal{R}}]$ pour tout type V.*

Démonstration. Par définition des ensembles RED, $tV \in RED_W[\underline{X} := \underline{\mathcal{R}}, Y := \mathcal{S}]$ pour tout candidat de réducibilité $\mathcal{S}$. Or $RED_V[\underline{X} := \underline{\mathcal{R}}]$ est un candidat de réducibilité, d'où $tV \in RED_W[\underline{X} := \underline{\mathcal{R}}, Y := RED_V[\underline{X} := \underline{\mathcal{R}}]]$ d'après le lemme 2, on a le résultat. □

Le théorème de normalisation

On va maintenant montrer le résultat cherché :

Théorème. *Tous les termes du système F sont fortement normalisants.*

On va pour cela donner une nouvelle définition de la réducibilité, et montrer que tous les termes sont réductibles.

On dit que t de type T est réductible, si $t \in RED_T[\underline{X} := \underline{\mathcal{SN}}]$, où $\mathcal{SN}_i$ est l'ensemble des termes fortement normalisants de type X_i. Comme précédemment, on va montrer un résultat plus fort, qui est le suivant :

Théorème. *Soit t un terme de type T, dont les variables (d'ordre 1) libres sont $\underline{x}$, de type $\underline{U}$. On suppose que toutes les variables de type libres dans T et dans les U_i sont parmi $\underline{X}$. Soit $\underline{\mathcal{R}}$ une suite de candidats de réducibilité de types $\underline{V}$, où $\underline{V}$, $\underline{\mathcal{R}}$ et $\underline{X}$ ont même longueur, et soient $\underline{u}$ des termes de types $U_1[\underline{X} := \underline{V}]$, ..., $U_n[\underline{X} := \underline{V}]$ qui sont dans $RED_{U_1}[\underline{X} := \underline{\mathcal{R}}]$,..., $RED_{U_n}[\underline{X} := \underline{\mathcal{R}}]$. Alors $t[\underline{X} := \underline{V}][\underline{x} := \underline{u}] \in RED_T[\underline{X} := \underline{\mathcal{R}}]$.*

Démonstration. On peut raisonner par induction sur t :

1. t est une variable : $t = u_i$ et on a le résultat par définition.
2. t = vw par hypothèse d'induction,$v[\underline{X} := \underline{V}]$ $[\underline{x} := \underline{u}]$ et $w[\underline{X} := \underline{V}]$ $[\underline{x} := \underline{u}]$ réductibles, et donc $t \in RED_T[\underline{X} := \underline{\mathcal{R}}]$ par définition.
3. t=$\lambda x_F.v$ par hypothèse d'induction, $v[\underline{X} := \underline{V}]$ $[\underline{x} := \underline{u}]$ est réductible, et donc t l'est.
   Ces trois premiers cas se traitent en réalité excatement comme pour le système T.

4. t=vF, où F est un type. Par hypothèse d'induction, $v[\underline{X} := \underline{V}]\ [\underline{x} := \underline{u}]$ est réductible et par le lemme 3, t est réductible.
5. t=$\Lambda X.v$. On a de même $v[\underline{X} := \underline{V}]\ [\underline{x} := \underline{u}]$ réductible et donc t réductible.

□

3 Conclusion

On a donc étudié les preuves de deux résultats montrant la cohérence de certains système formels, et on a vu que ces preuves ne pouvaient s'exprimer que dans des systèmes formels plus puissants. On peut toutefois remarquer que ces preuves pourraient se faire dans le système formel lui-même, à condition de renoncer à l'induction : dans les deux cas, les parties d'ordre supérieure des preuves étaient celles qui généralisait de façon inductive la preuve à tous les termes. Si l'on se contente de faire une preuve par prototype, par exemple sur tous les élements de type `Int`, on obtiendra une démonstration codable dans la système lui-même, qui nous dira essentiellement que un n donné a une forme normale. Cependant, cette preuve par protoype a deux points faibles : elle n'est pas automatisable, et elle ne fournit pas un résultat global.

Toutefois, son existence pourrait donner l'idée que l'on peut prouver formellement et automatiquement des résultats autrement que par induction, et être une des nouvelles orientations de la théorie de la démonstration.

Annexe A

Le lambda-calcul

On se contentera ici de définir quelques concepts de base. Pour des références plus complète, voir, par exemple [Gou]

1 Syntaxe

L'ensemble $\mathcal{T}$ des lambda-termes est construit par point fixe :
- On se donne un ensemble (infini) de variables $\mathcal{V}$. $\mathcal{V} \subset \mathcal{T}$
- $App : (x, y) \to xy$
- $Abs : u \to \lambda x.u$, $x \in \mathcal{V}$
- $Par : u \to (u)$
- $\mathcal{T}$ est le plus petit ensemble contenant $\mathcal{V}$ stable à la fois par App, Abs et Par

Le parenthèsage sert à éviter les ambiguités.

2 Règles

a) α-renommage

On veut, par exemple, pouvoir identifier $\lambda x.x$ et $\lambda y.y$. On définit l'α – *renommage* de la façon suivante :

$$\lambda x.u \ \alpha \ \lambda y.u[x := y]$$

On étend cette relation binaire à sa clôture réfléxive, transitive et symétrique passant au contexte et compatible avec l'α-équivalence. On obtient une relation d'équivalence sur les termes que l'on note $=_\alpha$. On identifiera désormais toujours deux termes α-équivalents.

b) β-réduction

On définit la β-réduction de la façon suivante :

$$\lambda x.uv \ \beta \ u[x := v]$$

On étend cette relation par passage au contexte, et on la note alors $\rightarrow$, ou $\rightarrow_\beta$, puis en prend la clôture réflexive transitive, que l'on note $\rightarrow^*$ et enfin la clôture symétrique, que l'on note $=_\beta$.

c) La substitution

On a considéré jusqu'ici intuitive la signification de la substitution. Il est pourtant indispensable de la définir de façon précise, pour éviter d'obtenir des erreurs syntaxiques ou sémantiques.

On définit donc deux fonctions fv et bv, qui à un lambda-terme associent un ensemble de variables, de la façon suivante :

- $x \in \mathcal{V}$: fv(x) = x, bv(x) = $\emptyset$
- uv, $(u, v) \in \mathcal{T}^2$: fv(uv) =fv(u)$\cup$ fv(v), bv(uv) = bv(u)$\cup$ bv(v)
- $\lambda x.u$, $x \in \mathcal{V}$, $u \in \mathcal{T}$ fv($\lambda x.u$) = fv(u)$\setminus$ x, bv($\lambda x.u$) = bv(u) $\cup$ x.

Définition. *On dit que x est substituable par v dans u ssi $x \notin bv(u)$ et $fv(v) \cap bv(u) = \emptyset$.*

On peut alors définir l'α-renommage et la β-réduction de façon rigoureuse, quitte à prendre un α-équivalent où x est substituable.

d) L'η-réduction

On introduit une deuxième règle de réduction, intuitivement valide et pourtant complètement indépendante de la β-réduction. :

$$\lambda x.ux \ \eta \ u(x \notin fv(u))$$

On peut alors définir $\rightarrow_{\beta\eta}$, $\rightarrow^*_{\beta\eta}$, $=_{\beta\eta}$. On voit assez facilement que cette règle ne change pas les problèmes de normalisation, puisqu'elle devient redondante avec la β-réduction dès que l'on applique le terme à un autre terme.

3 Confluence

Il s'agit ici de montrer que la forme normale est unique quand elle existe. On se contentera de donner l'énoncé de la proposition et une idée de la preuve.

Théorème. *Si on a $u \in \mathcal{T}$ tq $u \rightarrow^* v$ et $u \rightarrow^* w$, alors $\exists$ t tq $v \rightarrow^* t$ et $w \rightarrow^* t$*

Pour montrer cette propriété, la difficulté est qu'on ne maîtrise pas le nombre de réduction entre v et t, par exemple. On définit donc une nouvelle réduction, contenue dans $\rightarrow^*$ mais plus forte que $\rightarrow$, qui nous assure que l'on a une réduction en une étape. Cette réduction, que l'on note $\Rightarrow$ nous permet de montrer :

$$u \Rightarrow v_1 \wedge u \Rightarrow v_2 \text{ implique } \exists \ t \text{ tq } v_1 \Rightarrow t \wedge v_2 \Rightarrow t$$

Bibliographie

[GLT] **Jean-Yves Girard, Yves Lafont, Paul Taylor**
Proofs and Types Cambridge Tracts in Theoretical Computer Science

[Lon] **Longo G.**
On the Proofs of some formally unprovable propositions an Prototype Proofs in Type Theory LNCS 2277, Springer

[LMS] **Longo G., Milsted K. and Soloviev S.**
The genericity theorem and parametricity in polymorphic Lambda-calculus Theoretical Computer Science 121.

[Gou] **Jean Goubault-Larrecq**
Notes du cours de logique informatique du MMFAI

[Gir] **Jean-Yves Girard**
*Une extension de l'interprétation fonctionnelle de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types*Proceedings of th 2nd *Scandinvavian Logic Symposium*, North-Holland.