La part du calcul

Gilles Dowek

▶ To cite this version:

Gilles Dowek. La part du calcul. Computer Science [cs]. Université de Paris 7, 1999. tel-04114581

HAL Id: tel-04114581
https://inria.hal.science/tel-04114581v1

Submitted on 2 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Mémoire d'Habilitation à diriger des recherches

présenté à l'Université de Paris 7

par

Gilles DOWEK

LA PART DU CALCUL

Soutenu le 4 juin 1999 devant la commission d'examen composée de :

MM.	Guy	COUSINEAU	Président
	Henk	BARENDREGT	Rapporteurs
	Pierre-Louis	CURIEN	
	Jean-Pierre	JOUANNAUD	
	Paul	GOCHET	Examinateurs
	Gérard	HUET	
	Gilles	KAHN	
	Per	MARTIN-LÖF	

Ce mémoire est une introduction à un dossier d'habilitation composé d'articles écrits entre 1994 et 1998 et référencés ci-dessous [a, b, c, d, e, f, g, h, i, j]. Par ailleurs, une annexe contient deux articles d'exposition référencés [x, y] destinés essentiellement à des chercheurs de disciplines connexes [1].

On trouvera peu d'arguments techniques dans ce mémoire. J'ai tenté, en revanche, de donner les motivations de ce travail, de mettre les différents articles en perspective, de donner des exemples et de montrer quelle pourrait être une suite de ces recherches.

Les articles [a, b, c, d, e, f, h, i] ont été publiés dans des journaux ou des actes de conférences. L'article [g] n'est, en revanche, que soumis à publication. L'article [j] est un article invité au symposium *First order theorem proving* qui s'est tenu à Vienne en Novembre 1998. Les articles publiés dans des journaux apparaissent dans ce dossier dans leur forme définitive. Ceux qui sont publiés dans des actes de conférences ou soumis à publication apparaissent en version longue.

1. Le premier est la transcription d'un exposé présenté en avril 1996 au séminaire *Qu'est-ce qu'une logique ?* organisé à la Sorbonne par J.-B. Joinet et S. Berestovoy. On y présente ce qui est devenu, par la suite, la thèse principale de cette habilitation : une théorie est composée non seulement d'axiomes, mais aussi de règles de calcul. Le dernier paragraphe de cet article essaie de comprendre les répercussions de cette thèse sur la définition des notions de sens et de dénotation.

Le second a été présenté au colloque *Voir, Entendre, Raisonner, Calculer* à la Cité des Sciences et de l'Industrie, à Paris en juin 1997. À travers une comparaison du langage mathématique et des langages de programmation, on y propose l'idée de la programmation en langage mathématique et on y discute de l'impact de cette proposition sur la formalisation des mathématiques.

Je tiens, avant toute chose, à remercier Monsieur Guy Cousineau de l'honneur qu'il me fait en présidant ce jury. Enseigner dans le DEA *Programmation* qu'il dirige a été pour moi une source constante de questions et de remise en question.

Je tiens, ensuite, à remercier Messieurs Henk Barendregt, Pierre-Louis Curien et Jean-Pierre Jouannaud qui ont lu ces pages avec une grande patience. Je dois, entres autres choses, à Henk Barendregt de m'avoir signalé le passage dans lequel Poincaré oppose démonstration et vérification et plusieurs discussions "informelles" sur les rapports entre déduction et calcul lors des symposiums *Types*. Je suis reconnaissant à Pierre-Louis Curien de l'intérêt qu'il a porté très tôt à l'utilisation que nous avons faite des substitutions explicites, qui est assez éloignée - en apparence - de ce pour quoi il les avait inventées. Jean-Pierre Jouannaud m'a invité à plusieurs reprises au séminaire qu'il anime à Orsay. J'ai toujours trouvé, lors de ces exposés, une stimulante exigence.

Je suis également très reconnaissant à Messieurs Paul Gochet, Gilles Kahn et Per Martin-Löf de l'intérêt qu'ils portent à ce travail en formant le jury qui le sanctionne. La clarté de Paul Gochet, qui a beaucoup contribué à faire connaître Quine aux informaticiens et la démonstration automatique aux philosophes, est pour moi un exemple. L'encyclopédisme de Gilles Kahn est un modèle difficile à imiter. De nombreuses conversations avec Per Martin-Löf m'ont appris un certain sens de la synthèse, et aussi de la prudence.

Toute ma gratitude va à Gérard Huet qui a accepté le rôle de directeur de recherche pour cette habilitation. L'influence de ses recherches sur ce travail qui touche à la résolution d'ordre supérieur, à la réécriture et à l'interprétation fonctionnelle des démonstrations est immense.

Ce travail, qui est largement collectif, n'aurait jamais pu voir le jour si je n'avais pas rencontré Thérèse Hardin, Claude Kirchner, Frank Pfenning et Benjamin Werner avec qui nombre des articles qui constituent ce dossier sont signés. Je suis conscient de la chance que j'ai eu de pouvoir travailler avec chacun d'eux et de ce qu'ils m'ont appris.

Enfin, ce travail n'aurait pas non plus pu se faire, si je n'avais pas eu la chance de travailler dans le projet Coq de l'INRIA que dirige Christine Paulin. La richesse intellectuelle et la liberté que procure cet exceptionnel environnement de travail méritent d'être préservées.

Je crois que deux et deux sont quatre, Sganarelle,
et que quatre et quatre sont huit.
Molière – *Dom Juan*

Comme toute proposition mathématique, la proposition $2 + 2 = 4$ est susceptible d'être démontrée. La figure 1 donne une démonstration possible. Certains, pourtant, comme H. Poincaré, contestent le fait qu'un tel raisonnement soit véritablement une *démonstration* : "On ne saurait nier que ce raisonnement soit purement analytique. Mais interrogez un mathématicien quelconque : 'Ce n'est pas une démonstration proprement dite, vous répondra-t-il, c'est une vérification' " [81].

Le problème de savoir si deux et deux font quatre, ou plus généralement de trouver la somme de deux nombres, leur produit ou leur plus grand diviseur commun, est en effet d'une nature très différente de celui de savoir si, par exemple, un carré peut être le double d'un autre ou la somme de deux puissances $n^{\text{ème}}$, une puissance $n^{\text{ème}}$. Résoudre le premier problème demande uniquement d'effectuer un *calcul*, alors que résoudre le second demande de construire un *raisonnement*, car la quantification sur une quantité infinie d'objets rend la vérification impossible.

Les mathématiques assyrio-babyloniennes et égyptiennes étaient essentiellement constituées de méthodes de calcul : méthodes pour effectuer des opérations comptables ou calculer des aires agricoles, ... Puis la géométrie grecque introduisit la notion d'espace indéfini, elle cessa alors de parler de telle ou telle parcelle agricole, pour parler d'objets abstraits : les triangles, les cercles, ... De même, l'arithmétique cessa de parler de telle ou telle quantité, pour parler, elle aussi, d'objets abstraits : les nombres. Avec ces objets abstraits arrivèrent de nouveaux problèmes que le calcul ne suffisait pas à résoudre. On introduisit alors un nouvel outil : le raisonnement. Les *Éléments* d'Euclide restèrent longtemps le prototype de la méthode mathématique : la solution de problèmes par le raisonnement. C'est encore ce point de vue qui domine, au début du vingtième siècle dans les travaux de G. Peano, G. Frege, D. Hilbert, E. Zermelo, A.N. Whitehead, B. Russell ou N. Bourbaki. Dans l'arithmétique de Peano, par exemple, il n'y a pas de règle de calcul qui permette de vérifier que $2 + 2 = 4$, mais il y a des axiomes qui permettent de démontrer cette proposition. La spécificité de ce problème, qui peut se résoudre par un simple calcul, est gommée, au profit d'une méthode uniforme : le raisonnement. Il en est de même dans la *Begriffschrift* de Frege, la géométrie de Hilbert, la théorie des ensembles de Zermelo, les *Principia mathematica* de Whitehead et Russell ou les *Éléments de mathématique* de Bourbaki.

Si le raisonnement domine dans ces travaux qui concernent tous la formalisation des mathématiques ou de l'une de ses branches, on peut noter que dans les mathématiques informelles, en revanche, une place importante a toujours été accordée à la construction de méthodes de calcul, comme en témoignent les algorithmes de multiplication des nombres en notation décimale (al-Uqlidisi, ...), de résolution d'équation algébriques du

	$2+2=2+(1+1)$	définition de 2
	$2+(1+1)=(2+1)+1$	définition de +
	$(2+1)+1=3+1$	définition de 3
	$3+1=4$	définition de 4
d'où		
	$2+2=4$	

Figure 1 – Deux et deux font quatre

troisième et du quatrième degré (G. Cardan, L. Ferrari, ...), de résolution d'équations linéaires à plusieurs inconnues (C.F. Gauss, ...), de résolutions d'équations diophantiennes (A. Cauchy, ...), de résolution approchées d'équations algébriques (I. Newton, ...) et différentielles (L. Euler, C. Runge et M.W. Kutta, ...), de test de primalité (J. Bertrand, D.H. Lehmer, ...), ... Il y a, en fait, un certain paradoxe de voir coexister un discours qui réduit les mathématiques à la méthode axiomatique et une pratique qui accorde une si grande place au calcul.

Ce paradoxe se retrouve au sein de la logique. La notion de calcul est, par exemple, centrale dans le programme de Hilbert (trouver un algorithme qui décide si une proposition mathématique est un théorème ou non) et dans la théorie de la calculabilité qui permet d'y répondre négativement. Cette notion est également centrale dans l'étude des démonstrations intuitionnistes, par exemple dans la démonstration de terminaison de l'algorithme d'*élimination des coupures* qui transforme chaque démonstration d'existence en un témoin de cette existence. Pourtant, si les logiciens s'intéressent au calcul, la notion de démonstration qu'ils utilisent, qui est celle de la logique du premier ordre, se fonde, quant à elle, uniquement sur le raisonnement.

Depuis l'apparition des ordinateurs, la notion de calcul connaît un certain renouveau. Par delà l'effet de mode, ce regain d'intérêt s'explique, très concrètement, par le fait que la solution de certains problèmes en informatique demande d'utiliser une formalisation des mathématiques qui accorde une place au calcul, contrairement à la formulation traditionnelle de la théorie des ensembles.

Le premier de ces problèmes est celui de la *démonstration automatique*, c'est-à-dire de la construction de programmes qui recherchent des démonstrations pour des propositions mathématiques. L'utilisation de formalisations des mathématiques reposant uniquement sur le raisonnement aboutit au paradoxe suivant : quand on demande à un tel programme de résoudre le problème “Est-ce que $2+2=4$?”, il ne voit pas que ce problème peut se résoudre par un simple calcul et il cherche à démontrer la proposition $2+2=4$ en utilisant potentiellement tous les axiomes des mathématiques. Chercher une telle démonstration est naturellement une opération beaucoup plus coûteuse qu'effectuer une simple addition.

Un autre problème est celui de la *vérification de correction des démonstrations*. Quand on vérifie la correction d'une démonstration qui utilise le fait que $2+2$ est égal à 4, on peut penser qu'il est paradoxal de devoir vérifier une démonstration de ce fait, alors qu'on peut directement effectuer le calcul et vérifier le fait lui-même.

Un autre exemple de problème est celui de la *programmation en langage mathématique* [64, 79, y]. Il est devenu banal de remarquer qu'un programme qui associe une valeur (la valeur de sortie) à une ou plusieurs autres valeurs (les valeurs d'entrée) n'est rien d'autre qu'une fonction, au sens mathématique du terme. Par exemple, le programme qui calcule le montant des mensualités d'un prêt en fonction de la somme empruntée, du nombre de mensualités et du taux d'intérêt n'est rien d'autre que la fonction

$$f = e, n, t \longmapsto \frac{e\ t}{1 - \frac{1}{(1+t)^n}}$$

Le langage mathématique étant bien adapté à l'expression de fonctions, certains proposent de l'utiliser comme langage de programmation. Si on emprunte 5000 € sur 24 mois au taux mensuel de 0.64% (ce qui correspond à un taux annuel de 8%), les mensualités seront de $f(5000, 24, 0.0064)$ €. Hélas, ce n'est pas cette information

qui est utile, mais celle que les mensualités seront de 225 €. Exécuter les programmes écrits en langage mathématique demande donc une formalisation des mathématiques qui distingue un terme quelconque, comme $f(5000, 24, 0.0064)$ d'une *valeur*, comme 225, et qui permette de calculer le terme $f(5000, 24, 0.0064)$ en la valeur 225.

Un dernier exemple est le *calcul formel*. Ici encore, il ne suffit pas de savoir que la dérivée de la fonction $x \longmapsto sin^2 x$ est la fonction $x \longmapsto 2\ sinx\ cosx$ mais il faut que le terme $(D\ (x \longmapsto sin^2\ x))$ se calcule sur le terme $x \longmapsto 2\ sinx\ cosx$.

Des systèmes formels qui intègrent raisonnement et calcul ont été développés pour répondre à ces problèmes : G. Plotkin [80] et P.B. Andrews [3] proposent, dans le cadre de la démonstration automatique, de mêler certains axiomes à l'unification, ce qui permet de les supprimer de l'ensemble des axiomes avec lesquels on raisonne, après le langage du système Automath [26], la théorie des types de P. Martin-Löf [65], le Calcul des constructions [15] et le Calcul des constructions inductives [78, 91] ont une règle de conversion qui permet d'effectuer des calculs dans une proposition sans changer sa démonstration, l'Arithmétique fonctionnelle du second ordre [60, 59] permet d'utiliser des axiomes équationnels sans que cela apparaisse dans les démonstrations. H. Barendregt et E. Barendsen [12] ont proposé d'appeler "principe de Poincaré" ce principe - qu'on retrouve ici et là - qui consiste à calculer dans une proposition sans en changer la démonstration.

L'objectif de l'ensemble d'articles réunis dans ce dossier est de proposer, et d'étudier les propriétés, d'un ensemble de règles de déduction, *la déduction modulo*, qui intègre le calcul et le raisonnement de la manière la plus générale qui soit, indépendamment d'une formalisation particulière des mathématiques (la théorie des ensembles, l'Arithmétique fonctionnelle du second ordre, la théorie des types simples, la théorie des types de Martin-Löf, le Calcul des constructions, le Calcul des constructions inductives, ...) ou d'une application particulière (la démonstration automatique, la vérification de correction des démonstrations, la programmation en langage mathématique, le calcul formel, ...).

Chapitre 1

La déduction modulo

1.1 Les règles

Les notions de langage, de terme et de proposition de la *déduction modulo* sont les mêmes que celles de la logique du premier ordre. En revanche, dans ce formalisme, une théorie est composée non seulement d'un ensemble d'axiomes, mais aussi d'une congruence $\equiv$ définie sur les termes et les propositions. On identifie les propositions congruentes. La règle de *modus ponens*, par exemple, ne s'exprime plus de la manière traditionnelle

$$\frac{A \Rightarrow B \quad A}{B}$$

mais, pour prendre en compte le cas où les deux occurrences de A ne sont pas identiques, mais seulement congruentes, on formule cette règle de la manière suivante

$$\frac{A' \Rightarrow B \quad A}{B} \text{ si } A \equiv A'$$

Une formulation plus générale sera, en fait, utile dans la suite

$$\frac{C \quad A}{B} \text{ si } C \equiv A \Rightarrow B$$

Les règles de déduction naturelle modulo sont données figure 1.1 et celles du calcul des séquents modulo figure 1.2. Nous avons proposé le calcul des séquents modulo dans [g], puis la déduction naturelle modulo dans [h].

À titre d'exemple, donnons une démonstration, en arithmétique, en déduction naturelle modulo de la proposition “4 est pair”, $\exists x\ (2 \times x = 4)$

$$\cfrac{\cfrac{\overline{\forall x\ x = x}}{2 \times 2 = 4}\ (x, x = x, 4)\ \forall\text{-élim}}{\exists x\ (2 \times x = 4)}\ (x, 2 \times x = 4, 2)\ \exists\text{-intro}$$

Substituer la variable x par le terme 2 dans la proposition $2 \times x = 4$ donne la proposition $2 \times 2 = 4$, qui est congruente à $4 = 4$. Le passage d'une proposition à l'autre, qui requiert plusieurs fastidieuses étapes de démonstration dans la formulation traditionnelle de l'arithmétique de Peano, est ici gommée de la démonstration, c'est un simple calcul qui n'a pas besoin d'être écrit, puisque chacun peut le refaire.

On peut remarquer que, dans cette démonstration, on n'utilise pas les axiomes de l'addition et de la multiplication. Cela s'explique par le fait que raisonner modulo une congruence permet de supprimer ces axiomes. En effet, si on pose que les termes $0 + x$ et x sont congruents, alors l'axiome $\forall x\ 0 + x = x$ est congruent à l'axiome de l'égalité $\forall x\ x = x$, il est donc redondant et on peut le supprimer.

$$\frac{}{\Gamma \vdash_{\equiv} B} \text{ axiome si } A \in \Gamma \text{ et } A \equiv B$$

$$\frac{\Gamma, A \vdash_{\equiv} B}{\Gamma \vdash_{\equiv} C} \Rightarrow\text{-intro si } C \equiv (A \Rightarrow B)$$

$$\frac{\Gamma \vdash_{\equiv} C \quad \Gamma \vdash_{\equiv} A}{\Gamma \vdash_{\equiv} B} \Rightarrow\text{-élim si } C \equiv (A \Rightarrow B)$$

$$\frac{\Gamma \vdash_{\equiv} A \quad \Gamma \vdash_{\equiv} B}{\Gamma \vdash_{\equiv} C} \wedge\text{-intro si } C \equiv (A \wedge B)$$

$$\frac{\Gamma \vdash_{\equiv} C}{\Gamma \vdash_{\equiv} A} \wedge\text{-élim si } C \equiv (A \wedge B)$$

$$\frac{\Gamma \vdash_{\equiv} C}{\Gamma \vdash_{\equiv} B} \wedge\text{-élim si } C \equiv (A \wedge B)$$

$$\frac{\Gamma \vdash_{\equiv} A}{\Gamma \vdash_{\equiv} C} \vee\text{-intro si } C \equiv (A \vee B)$$

$$\frac{\Gamma \vdash_{\equiv} B}{\Gamma \vdash_{\equiv} C} \vee\text{-intro si } C \equiv (A \vee B)$$

$$\frac{\Gamma \vdash_{\equiv} D \quad \Gamma, A \vdash_{\equiv} C \quad \Gamma, B \vdash_{\equiv} C}{\Gamma \vdash_{\equiv} C} \vee\text{-élim si } D \equiv (A \vee B)$$

$$\frac{\Gamma, A \vdash_{\equiv} \bot}{\Gamma \vdash_{\equiv} B} \neg\text{-intro si } B \equiv (\neg A)$$

$$\frac{\Gamma \vdash_{\equiv} B \quad \Gamma \vdash_{\equiv} A}{\Gamma \vdash_{\equiv} \bot} \neg\text{-élim si } B \equiv (\neg A)$$

$$\frac{\Gamma \vdash_{\equiv} B}{\Gamma \vdash_{\equiv} A} \bot\text{-élim si } B \equiv \bot$$

$$\frac{\Gamma \vdash_{\equiv} A}{\Gamma \vdash_{\equiv} B} (x, A) \ \forall\text{-intro si } B \equiv (\forall x\ A) \text{ et } x \notin FV(\Gamma)$$

$$\frac{\Gamma \vdash_{\equiv} B}{\Gamma \vdash_{\equiv} C} (x, A, t) \ \forall\text{-élim si } B \equiv (\forall x\ A) \text{ et } C \equiv [t/x]A$$

$$\frac{\Gamma \vdash_{\equiv} C}{\Gamma \vdash_{\equiv} B} (x, A, t) \ \exists\text{-intro si } B \equiv (\exists x\ A) \text{ et } C \equiv [t/x]A$$

$$\frac{\Gamma \vdash_{\equiv} C \quad \Gamma, A \vdash_{\equiv} B}{\Gamma \vdash_{\equiv} B} (x, A) \ \exists\text{-élim si } C \equiv (\exists x\ A) \text{ et } x \notin FV(\Gamma B)$$

$$\frac{}{\Gamma \vdash_{\equiv} A} B \text{ tiers exclu si } A \equiv (B \vee \neg B)$$

FIGURE 1.1 – La déduction naturelle modulo

$$\frac{}{A \vdash_{\equiv} B} \text{ axiome si } A \equiv B$$

$$\frac{\Gamma, A \vdash_{\equiv} \Delta \quad \Gamma \vdash_{\equiv} B, \Delta}{\Gamma \vdash_{\equiv} \Delta} \text{ coupure si } A \equiv B$$

$$\frac{\Gamma, B_1, B_2 \vdash_{\equiv} \Delta}{\Gamma, A \vdash_{\equiv} \Delta} \text{ contr-gauche si } A \equiv B_1 \equiv B_2$$

$$\frac{\Gamma \vdash_{\equiv} B_1, B_2, \Delta}{\Gamma \vdash_{\equiv} A, \Delta} \text{ contr-droite si } A \equiv B_1 \equiv B_2$$

$$\frac{\Gamma \vdash_{\equiv} \Delta}{\Gamma, A \vdash_{\equiv} \Delta} \text{ aff-gauche}$$

$$\frac{\Gamma \vdash_{\equiv} \Delta}{\Gamma \vdash_{\equiv} A, \Delta} \text{ aff-droite}$$

$$\frac{\Gamma \vdash_{\equiv} A, \Delta \quad \Gamma, B \vdash_{\equiv} \Delta}{\Gamma, C \vdash_{\equiv} \Delta} \Rightarrow\text{-gauche si } C \equiv (A \Rightarrow B)$$

$$\frac{A\Gamma \vdash_{\equiv} B, \Delta}{\Gamma \vdash_{\equiv} C, \Delta} \Rightarrow\text{-droite si } C \equiv (A \Rightarrow B)$$

$$\frac{\Gamma, A, B \vdash_{\equiv} \Delta}{\Gamma, C \vdash_{\equiv} \Delta} \wedge\text{-gauche si } C \equiv (A \wedge B)$$

$$\frac{\Gamma \vdash_{\equiv} A, \Delta \quad \Gamma \vdash_{\equiv} B, \Delta}{\Gamma \vdash_{\equiv} C, \Delta} \wedge\text{-droite si } C \equiv (A \wedge B)$$

$$\frac{\Gamma, A \vdash_{\equiv} \Delta \quad \Gamma, B \vdash_{\equiv} \Delta}{\Gamma, C \vdash_{\equiv} \Delta} \vee\text{-gauche si } C \equiv (A \vee B)$$

$$\frac{\Gamma \vdash_{\equiv} A, B, \Delta}{\Gamma \vdash_{\equiv} C, \Delta} \vee\text{-droite si } C \equiv (A \vee B)$$

$$\frac{\Gamma \vdash_{\equiv} A, \Delta}{\Gamma, B \vdash_{\equiv} \Delta} \neg\text{-gauche si } B \equiv (\neg A)$$

$$\frac{\Gamma, A \vdash_{\equiv} \Delta}{\Gamma \vdash_{\equiv} B, \Delta} \neg\text{-droite si } B \equiv (\neg A)$$

$$\frac{}{\Gamma, A \vdash_{\equiv} \Delta} \bot\text{-gauche si } A \equiv \bot$$

$$\frac{\Gamma, [t/x]A \vdash_{\equiv} \Delta}{\Gamma, B \vdash_{\equiv} \Delta} (x, A, t) \ \forall\text{-gauche si } B \equiv (\forall x \ A)$$

$$\frac{\Gamma \vdash_{\equiv} A, \Delta}{\Gamma \vdash_{\equiv} B, \Delta} (x, A) \ \forall\text{-droite si } B \equiv (\forall x \ A) \text{ et } x \notin FV(\Gamma\Delta)$$

$$\frac{\Gamma, A \vdash_{\equiv} \Delta}{\Gamma, B \vdash_{\equiv} \Delta} (x, A) \ \exists\text{-gauche si } B \equiv (\exists x \ A) \text{ et } x \notin FV(\Gamma\Delta)$$

$$\frac{\Gamma \vdash_{\equiv} [t/x]A, \Delta}{\Gamma \vdash_{\equiv} B, \Delta} (x, A, t) \ \exists\text{-droite si } B \equiv (\exists x \ A)$$

FIGURE 1.2 – Le calcul des séquents modulo

Pour que la correction des démonstrations reste décidable, il faut que l'information supprimée des démonstrations puisse être reconstruite. Il faut donc que la congruence $\equiv$ soit décidable. De plus, il faut indiquer le témoin lors de l'application des règles d'élimination du quantificateur universel et d'introduction du quantificateur existentiel. En effet, en l'absence de ces témoins, la décidabilité de la correction des démonstrations demanderait non seulement la décidabilité de la congruence $\equiv$ elle-même, mais aussi celle du *filtrage* modulo cette congruence, puisqu'il faudrait pouvoir décider si une proposition est une instance d'une autre modulo $\equiv$ ou non.

Proposition 1.1.1 *Si la congruence $\equiv$ est décidable, il existe un algorithme qui prend en argument une proposition et une dérivation (en déduction naturelle modulo ou en calcul des séquents modulo) et indique si la dérivation est une démonstration correcte de la proposition ou non.*

Cependant, rien n'empêche de considérer abstraitement des "démonstrations" modulo une congruence indécidable ou dont la décidabilité est un problème ouvert, et d'étudier certaines de leur propriétés. C'est par exemple le cas de la théorie des ensembles que nous discuterons au chapitre 3.

1.2 Les congruences

1.2.1 La réécriture des termes

Une congruence peut se définir par un ensemble de règles de réécriture sur les termes du langage. En arithmétique, par exemple, on pose les règles

$$0 + y \to y$$

$$Succ(x) + y \to Succ(x + y)$$

$$0 \times y \to 0$$

$$Succ(x) \times y \to x \times y + y$$

De même, quand on utilise une opération associative, on pose la règle

$$(x + y) + z \to x + (y + z)$$

Si ce système de réécriture termine et est confluent, la congruence est décidable.

1.2.2 Les équations entre termes

Quand on utilise une opération commutative, on veut parfois pouvoir identifier les propositions $P(x+y)$ et $P(y+x)$, c'est-à-dire considérer que la permutation des termes est du ressort du calcul et non du raisonnement, bien que la règle

$$x + y \to y + x$$

ne termine pas.

On considère donc des congruences définies par des règles de réécriture et des équations entre termes.

1.2.3 La réécriture des propositions

Ces différentes congruences sont définies par des règles de réécriture et des équations sur les termes du langage. La congruence sur les propositions est induite par la congruence sur les termes : c'est parce que les termes $2 + 2$ et 4 sont congruents que les propositions $2 + 2 = 4$ et $4 = 4$ le sont également. Dans de nombreux cas, on veut pouvoir définir des congruences directement sur les propositions. On est amené à poser des règles de réécriture sur les propositions elles-mêmes, par exemple la règle

$$Succ(x) = Succ(y) \to x = y$$

Dans cet exemple, on réécrit une proposition atomique sur une autre proposition atomique. Dans d'autres exemples, on veut pouvoir effectuer des calculs qui introduisent des connecteurs ou des quantificateurs. Par exemple, dans les anneaux intègres, la règle

$$x \times y = 0 \rightarrow x = 0 \vee y = 0$$

transforme une proposition atomique en une disjonction.

Remarquons que le langage des propositions contenant des symboles lieurs (les quantificateurs) ces systèmes de réécriture sont des *systèmes de réduction combinatoires* (*combinatory reduction systems*) ou CRS [58].

La forme des propositions atomiques joue un rôle très mineur dans les démonstrations (seule la règle *axiome* y est sensible). La possibilité d'appliquer ou non une règle logique dépend essentiellement des connecteurs et quantificateurs de la proposition. De ce fait, les congruences qui n'affectent que les termes influent peu sur la possibilité d'appliquer une règle logique ou non. La situation est tout à fait différente avec les règles comme celle ci-dessus : le fait que la proposition $x \times y = 0$ soit congruente à la proposition $x = 0 \vee y = 0$ permet de la démontrer en utilisant une règle d'introduction de la disjonction. Ces règles permettent donc une véritable interaction entre le calcul et le raisonnement.

Dans les articles réunis dans ce dossier, nous nous sommes restreints à des règles qui réécrivent des propositions atomiques en des propositions quelconques. Cela nous prive de la possibilité d'appliquer les règles de simplification propositionnelle proposées par J. Hsiang [56]. L'avantage de se placer dans une telle restriction est que, pour pouvoir appliquer une règle de déduction, il suffit de réduire la proposition considérée. Ce ne serait pas le cas si on avait, par exemple, une règle de la forme

$$A \wedge B \rightarrow C$$

dans ce cas, la proposition C étant congruente à la proposition $A \wedge B$, on pourrait la démontrer à l'aide de la règle d'introduction de la conjonction, mais pour mettre cette proposition C sous la forme d'une conjonction, il faudrait appliquer la règle "à l'envers", ce qui est contraire à l'intuition de la réécriture.

Les congruences que nous considérons sont donc définies par

— un ensemble $\mathcal{E}$ de règles de réécriture et d'équations entre termes ou entre propositions atomiques,
— un ensemble $\mathcal{R}$ de règles de réécriture qui réécrivent des propositions atomiques sur des proposition quelconques.

1.3 Le lemme d'équivalence

La première propriété de ces systèmes de déduction modulo est le lemme d'équivalence démontré dans [g]. Ce lemme établit que du point de vue de la démontrabilité, une théorie modulo est toujours équivalente à une théorie du premier ordre. Du point de vue de la démontrabilité, la déduction modulo n'est donc pas une nouvelle logique, c'est simplement une nouvelle formulation de la logique du premier ordre.

Proposition 1.3.1 *(Équivalence) Soit une congruence $\equiv$, et $\mathcal{T}$ la théorie formée de toutes les clôtures universelles des propositions de la forme $A \Leftrightarrow B$ où $A \equiv B$. On a*

$$\Gamma \vdash_{\equiv} P \textit{ si et seulement si } \mathcal{T}\Gamma \vdash P$$

Naturellement, si les propositions démontrées sont les mêmes, les démonstrations sont très différentes. En particulier elles sont beaucoup plus courtes car les étapes calculatoires ont été supprimées.

1.4 La notion de modèle pour la déduction modulo

Le lemme d'équivalence nous permet de définir simplement la notion de modèle pour la déduction modulo : on définit les modèles d'une théorie modulo $(\equiv, \Gamma)$ comme les modèles de $\mathcal{T}\Gamma$ où $\mathcal{T}$ est la théorie ci-dessus. Cette définition peut également se formuler de la manière suivante.

Définition 1.4.1 Un modèle de $(\equiv, \Gamma)$ est un modèle de Γ tel que si $P \equiv Q$ alors P et Q ont même dénotation.

Du théorème de correction et du théorème de complétude de K. Gödel pour la logique du premier ordre, on déduit les propositions suivantes.

Proposition 1.4.1 *(Correction) Si $\Gamma \vdash_{\equiv} P$ alors P est valide dans tous les modèles de $(\equiv, \Gamma)$.*

Proposition 1.4.2 *(Complétude) Si P est valide dans tous les modèles de $(\equiv, \Gamma)$ alors $\Gamma \vdash_{\equiv} P$.*

Avec cette notion de modèle, deux propositions congruentes ont nécessairement la même dénotation, mais pas nécessairement deux termes congruents. Cependant, les dénotations de deux termes congruents ne peuvent être discernée par aucun prédicat. Comme dans le cas de la théorie de l'égalité, on peut renforcer le lemme de complétude en se restreignant à une classe plus petite de modèles.

Définition 1.4.2 Un *modèle égalitaire* de $(\equiv, \Gamma)$ est un modèle de Γ tel que si $P \equiv Q$ alors P et Q ont même dénotation et si $t \equiv u$ alors t et u ont même dénotation.

Clairement, tous les modèles égalitaires d'une théorie $(\equiv, \Gamma)$ sont des modèles de cette théorie. On montre que pour tout modèle de $(\equiv, \Gamma)$, il existe un modèle égalitaire qui valide les mêmes propositions.

Proposition 1.4.3 *Pour tout modèle $\mathcal{M}$ de $(\equiv, \Gamma)$, il existe un modèle égalitaire $\mathcal{M}'$ qui valide les mêmes propositions.*

Soit $\mathcal{M}$ un modèle de $(\equiv, \Gamma)$. On appelle E la plus petite relation d'équivalence définie sur sur les éléments de M, l'ensemble de base de $\mathcal{M}$, telle que a et b sont mis en relation s'il existe des termes t et u tels que $t \equiv u$, où t dénote a et u dénote b. On pose $M' = M/E$. La relation $\equiv$ étant une congruence, les dénotations des symboles de fonction et de prédicat de $\mathcal{M}$ passent au quotient, ce qui définit un modèle $\mathcal{M}'$. On montre par récurrence structurelle que la dénotation d'un terme dans $\mathcal{M}'$ est la classe de sa dénotation dans $\mathcal{M}$ et la dénotation d'une proposition dans $\mathcal{M}'$ est sa dénotation dans $\mathcal{M}$.

On en déduit les lemmes de correction et de complétude.

Proposition 1.4.4 *(Correction) Si $\Gamma \vdash_{\equiv} P$ alors P est valide dans tous les modèles égalitaires de $(\equiv, \Gamma)$.*

Proposition 1.4.5 *(Complétude) Si P est valide dans tous les modèles égalitaires de $(\equiv, \Gamma)$ alors $\Gamma \vdash_{\equiv} P$.*

1.5 Le cas des congruences présentées par un système de réécriture

Quand la congruence est présentée par un système de réécriture $\mathcal{RE}$, on peut prendre une théorie $\mathcal{T}$ plus petite dans le lemme d'équivalence : pour chaque règle de réécriture $l \to r$ ou équation $l = r$ de $\mathcal{RE}$, on prend la clôture universelle de la proposition $t = u$ ou $t \Leftrightarrow u$ selon que t et u sont des termes ou des propositions.

Cela demande de disposer d'un prédicat d'égalité et des axiomes correspondants. Si la théorie ne contient pas un tel prédicat, on peut l'ajouter de manière conservatrice, comme le montre le lemme suivant.

Proposition 1.5.1 *(Conservativité) Soit $(\equiv, \Gamma)$ une théorie modulo sur un langage $\mathcal{L}$, on obtient une extension conservatrice de cette théorie en ajoutant un symbole d'égalité et les axiomes de l'égalité Eq correspondants.*

On montre que tout modèle égalitaire de la théorie $(\equiv, \Gamma)$ sur le langage $\mathcal{L}$ s'étend en un modèle égalitaire de $(\equiv, \Gamma Eq)$ sur le langage $\mathcal{L} \cup \{=\}$, en interprétant l'égalité par l'égalité.

On peut remarquer que cette démonstration ne marche pas si on ne se restreint pas aux modèles égalitaires. En effet, comme la congruence est définie sur les termes, quand on ajoute un nouveau prédicat elle s'étend automatiquement aux propositions atomiques formées avec ce prédicat. Ainsi, si t et u sont deux termes congruents la proposition $t = u$ est congruente à $t = t$ et elle est donc démontrable.

On obtient également une caractérisation plus simple des modèles égalitaires comme les modèles égalitaires (c'est-à-dire dans lesquels l'égalité dénote l'égalité) de la théorie ci-dessus. De manière équivalente, on peut les caractériser comme les modèles tels que pour toute règle de réécriture $l \to r$ ou équation $l = r$ de $\mathcal{RE}$, l et r ont même dénotation.

Chapitre 2

La théorie des types simples

Deux exemples de théories modulo méritent une attention particulière : la théorie des types simples, aussi appelée logique d'ordre supérieur et la théorie des ensembles. Ce chapitre est consacré à la théorie des types simples. L'intérêt de cette théorie est que beaucoup des propriétés et des algorithmes de la théorie des types simples (en particulier la normalisation des démonstrations et la résolution d'ordre supérieur) seront des cas particuliers de propriétés et d'algorithmes de la déduction modulo.

2.1 La théorie naïve des ensembles

La théorie naïve, incohérente, des ensembles, pose un schéma d'axiome, le *schéma de compréhension*, qui énonce l'existence de tous les ensembles définissables par une proposition P :

$$\forall x_1 \ ... \ \forall x_n \ \exists y \ \forall w \ (w \in y \Leftrightarrow P)$$

où $x_1, ..., x_n$ sont les variables libres de P sauf w.

Pour obtenir un langage de termes on peut skolémiser ce schéma. On obtient alors une infinité de symboles de fonction $f_{x_1,...,x_n,w,P}$ et un axiome

$$\forall x_1 \ ... \ \forall x_n \ \forall w \ (w \in f_{x_1,...,x_n,w,P}(x_1, ..., x_n) \Leftrightarrow P)$$

On peut, si on veut, noter $\{w \mid P\}$ le terme $f_{x_1,...,x_n,w,P}(x_1, ..., x_n)$ (ce point sera discuté en détail aux paragraphes 2.4 et 3.2).

Il est ensuite naturel de transformer ces axiomes en règles de réécriture

$$v \in f_{x_1,...,x_n,w,P}(y_1, ..., y_n) \to [y_1/x_1, ..., y_n/x_n, v/w]P$$

On peut remarquer que ce système de réécriture est trivialement confluent car il est orthogonal, mais qu'il ne termine pas. Un contre-exemple à la terminaison est le paradoxe de Russell : en notant R le terme $\{w \mid \neg w \in w\}$ c'est-à-dire le symbole d'individu $f_{w,\neg w \in w}$, on a la règle de réécriture

$$v \in R \to \neg v \in v$$

et donc en notant A la proposition $R \in R$

$$A \to \neg A$$

ce qui permet de construire une suite infinie de réductions

$$A \to \neg A \to \neg\neg A \to ...$$

De plus cette théorie est incohérente car on peut démontrer la proposition $\bot$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash_{\equiv} \neg A}\text{ axiome} \quad \dfrac{}{A \vdash_{\equiv} A}\text{ axiome}}{A \vdash_{\equiv} \bot}\text{ ¬-élim}}{\vdash_{\equiv} \neg A}\text{ ¬-intro} \quad \dfrac{\dfrac{\dfrac{}{A \vdash_{\equiv} \neg A}\text{ axiome} \quad \dfrac{}{A \vdash_{\equiv} A}\text{ axiome}}{A \vdash_{\equiv} \bot}\text{ ¬-élim}}{\vdash_{\equiv} A}\text{ ¬-intro}}{\vdash_{\equiv} \bot}\text{ ¬-élim}$$

2.2 La théorie des types simples

La théorie des types simples résout le paradoxe de Russell en classant les objets en fonction du nombre et de la nature de leurs arguments : la théorie des types simples est une théorie multisortée.

Comme la théorie naïve des ensembles, la théorie des types simples comporte des termes qui expriment des ensembles, mais elle comporte aussi des termes qui expriment des relations, et également des fonctions. Les ensembles s'identifient alors simplement avec les relations à un argument unique.

Si R est un terme de relation portant sur des objets de sorte $T_1, ..., T_n$ et $t_1, ..., t_n$ sont des termes de sorte $T_1, ..., T_n$, la proposition exprimant le fait que les termes $t_1, ..., t_n$ sont liés par la relation R s'écrit en général $R(t_1, ..., t_n)$. Comme en théorie naïve des ensembles, si on veut rester dans un langage du premier ordre, il faut introduire un symbole $\in_{T_1,...,T_n}$ et noter cette proposition $\in_{T_1,...,T_n} (R, t_1, ..., t_n)$. De même si f est un terme de fonction qui s'applique à des objets de sorte $T_1, ..., T_n$ pour donner un objet de sorte U et $t_1, ..., t_n$ sont des termes de sorte $T_1, ..., T_n$, le terme exprimant le résultat de l'application de f à $t_1, ..., t_n$ s'écrit en général $f(t_1, ..., t_n)$. Ici encore, si on veut rester dans un langage du premier ordre, il faut introduire un symbole $\alpha_{T_1,...,T_n,U}$ et noter ce terme $\alpha_{T_1,...,T_n,U}(f, t_1, ..., t_n)$.

Quand $n = 0$ le symbole α_U n'a pas grand intérêt, car le terme $\alpha_U(t)$ est redondant avec le terme t. En revanche, le symbole $\in_\phi$, qu'on note aussi ε, est beaucoup plus intéressant, car il permet de distinguer le terme t de la proposition $\varepsilon(t)$. Le terme t véhicule la même information que la proposition $\varepsilon(t)$, mais sous forme d'un terme. On l'appelle *contenu* de la proposition $\varepsilon(t)$. Ce symbole ε et corrélativement la distinction entre les propositions et leur contenu est la principale différence entre cette présentation de la théorie des types et les présentations plus traditionnelles [16, 6], qui insistent moins sur la possibilité d'exprimer cette théorie comme une théorie du premier ordre ou une théorie modulo. Nous reviendrons sur ce symbole au paragraphe 2.3.

En théorie des types, il est d'usage d'identifier la relation R avec la fonction qui à n objets $t_1, ..., t_n$ associe le contenu de la proposition $R(t_1, ..., t_n)$. Il est également d'usage de curryfier les fonctions, c'est-à-dire d'identifier la fonction f avec la fonction qui à x_1 associe la fonction qui à $x_2, ..., x_n$ associe $f(x_1, x_2, ..., x_n)$. On peut, de cette manière, considérer toutes les fonctions comme des fonctions d'un argument unique. Ainsi, les sortes qui permettent de classer les objets peuvent simplement être définies inductivement comme le plus petit ensemble qui contient ι (sorte des objets de base) et o (sorte des contenus propositionnels) et qui est clos par la flèche $\rightarrow$ qui forme la sorte des fonctions d'une sorte vers une autre. Ces sortes sont appelées *types simples*. Suivant l'usage on notera $T_1 \rightarrow ... \rightarrow T_n \rightarrow U$ le type $T_1 \rightarrow (T_2 \rightarrow ... \rightarrow (T_n \rightarrow U)...)$.

Dans cette présentation de la théorie des types, seuls les symboles de fonction $\alpha_{T,U}$, et le symbole de prédicat ε sont utiles. Le terme $\alpha_{T_1,...,T_n,U}(f, t_1, ..., t_n)$ se note $\alpha_{T_n,U}(...\alpha_{T_1,T_2\rightarrow...\rightarrow T_n\rightarrow U}(f, t_1), ..., t_n)$. De même, la proposition $\in_{T_1,...,T_n} (R, t_1, ..., t_n)$ se note $\varepsilon(\alpha_{T_n,o}(...\alpha_{T_1,T_2\rightarrow...\rightarrow T_n\rightarrow o}(R, t_1), ..., t_n))$. Suivant l'usage, dans les exemples, on note $(t\ u)$ le terme $\alpha_{T,U}(t, u)$ et $(t\ u_1\ ...\ u_n)$ le terme $(...(t\ u_1)\ ...\ u_n)$.

Comme en théorie naïve des ensembles on pose un schéma d'axiome de compréhension

$$\exists R\ \forall x_1\ ...\ \forall x_n\ (\varepsilon(R\ x_1\ ...\ x_n) \Leftrightarrow P)$$

où toutes les variables libres de la proposition P sont parmi $x_1, ..., x_n$.

Et on pose également un schéma d'axiome de compréhension fonctionnelle

$$\exists f\ \forall x_1\ ...\ \forall x_n\ (f\ x_1\ ...\ x_n) = t$$

où toutes les variables libres de t sont parmi $x_1, ..., x_n$.

Dans ce dernier schéma, on utilise un symbole d'égalité. On se place donc en logique du premier ordre avec égalité.

Skolémiser ces axiomes amène à introduire un nombre infinis de symboles d'individu, qu'on peut noter $\{x_1, ..., x_n \mid P\}$ et $x_1, ..., x_n \longmapsto t$ et les axiomes de *conversion*

$$\forall x_1 \ ... \ \forall x_n \ (\varepsilon(\{x_1, ..., x_n \mid P\} \ x_1 \ ... \ x_n) \Leftrightarrow P)$$

$$\forall x_1 \ ... \ \forall x_n \ ((x_1, ..., x_n \longmapsto t) \ x_1 \ ... \ x_n) = t$$

On peut ensuite transformer ces axiomes en règles de réécriture

$$\varepsilon(\{x_1, ..., x_n \mid P\} \ y_1 \ ... \ y_n) \to [y_1/x_1, ..., y_n/x_n]P$$

$$((x_1, ..., x_n \longmapsto t) \ y_1 \ ... \ y_n) \to [x_1 \mapsto y_1, ..., x_n \mapsto y_n]t$$ [1]

On peut démontrer qu'on garde une théorie équivalente en se restreignant à certaines instances de ces schémas, ou ce qui est équivalent à certains symboles de Skolem :
— $S_{T,U,V} = x, y, z \longmapsto ((x \ z) \ (y \ z))$, où x est de sorte $T \to U \to V$, y de sorte $T \to U$ et z de sorte T,
— $K_{T,U} = x, y \longmapsto x$ où x est de sorte T et y de sorte U,
— $\dot{\Rightarrow} = \{x, y \mid \varepsilon(x) \Rightarrow \varepsilon(y)\}$,
— $\dot{\wedge} = \{x, y \mid \varepsilon(x) \wedge \varepsilon(y)\}$,
— $\dot{\vee} = \{x, y \mid \varepsilon(x) \vee \varepsilon(y)\}$,
— $\dot{\neg} = \{x \mid \neg\varepsilon(x)\}$,
— $\dot{\bot} = \{\mid \bot\}$,
— $\dot{\forall}_T = \{x \mid \forall y \ \varepsilon(x \ y)\}$,
— $\dot{\exists}_T = \{x \mid \exists y \ \varepsilon(x \ y)\}$,
et aux axiomes ou règles de réécriture correspondants. On aboutit alors à la définition suivante de la théorie des types, comme une théorie multisortée modulo.

Définition 2.2.1 Les *types simples* sont définis inductivement par
— ι et o sont des types simples,
— si T et U sont des types simples alors $T \to U$ est un type simple.

Définition 2.2.2 Le langage de la théorie des types simples est un langage du premier ordre sorté par les types simples comprenant les symboles d'individu suivants.
— $S_{T,U,V}$ de sorte $(T \to U \to V) \to (T \to U) \to T \to V$,
— $K_{T,U}$ de sorte $T \to U \to T$,
— $\dot{\Rightarrow}$, $\dot{\wedge}$ et $\dot{\vee}$ de sorte $o \to o \to o$,
— $\dot{\neg}$ de sorte $o \to o$,
— $\dot{\bot}$ de sorte o,
— $\dot{\forall}_T$ et $\dot{\exists}_T$ de sorte $(T \to o) \to o$,
les symboles de fonction
— $\alpha_{T,U}$ de rang $(T \to U, T, U)$,
et le symbole de prédicat
— ε de rang (o).
Les règles de réécriture de la théorie des types simples sont celles de la figure 2.1.

1. On note $x_1 \mapsto t_1, ..., x_n \mapsto t_n$ la substitution simple, ou le remplacement, des variables $x_1, ..., x_n$ par les termes $t_1, ..., t_n$ par opposition à la notation $t_1/x_1, ..., t_n/x_n$ qui désigne la substitution "avec renommage" utilisée dans les langages avec lieurs.
Cette notation $x_1 \mapsto t_1, ..., x_n \mapsto t_n$ avec une flèche courte $\mapsto$ ne doit pas être confondue avec la notation $x_1, ..., x_n \longmapsto t$ avec une flèche longue.

$$(S\ x\ y\ z) \to ((x\ z)\ (y\ z))$$
$$(K\ x\ y) \to x$$
$$\varepsilon(\dot{\Rightarrow}\ x\ y) \to \varepsilon(x) \Rightarrow \varepsilon(y)$$
$$\varepsilon(\dot{\wedge}\ x\ y) \to \varepsilon(x) \wedge \varepsilon(y)$$
$$\varepsilon(\dot{\vee}\ x\ y) \to \varepsilon(x) \vee \varepsilon(y)$$
$$\varepsilon(\dot{\neg}\ x) \to \neg\varepsilon(x)$$
$$\varepsilon(\dot{\bot}) \to \bot$$
$$\varepsilon(\dot{\forall}\ x) \to \forall y\ \varepsilon(x\ y)$$
$$\varepsilon(\dot{\exists}\ x) \to \exists y\ \varepsilon(x\ y)$$

Figure 2.1 – Les règles de réécriture de la théorie des types simples

Ce système de réécriture est confluent car il est orthogonal. On montre dans [e] qu'il termine fortement en exhibant un plongement dans le langage des combinateurs simplement typés qui termine fortement par le théorème de W. Tait [89]. La congruence est donc décidable.

Dans cette théorie, on utilise deux notions de fonction qu'il ne faut pas confondre. Le symbole d'individu S, par exemple, est un terme et il a, de ce fait, une sorte. Ce terme exprime un objet de la théorie qui se révèle être une fonction. De ce fait, sa sorte est un type fonctionnel. Le symbole de fonction $\alpha_{T,U}$, en revanche n'est pas un terme et n'exprime pas un objet de la théorie mais une fonction associant un objet de la théorie à des objets de la théorie. La sorte des objets antécédents et images de cette fonction est définie par le rang de ce symbole. Cette situation est similaire à celle de la théorie des ensembles où il est d'usage de distinguer les ensembles, en tant qu'objets de la théorie des ensembles d'objets de la théorie.

On peut ensuite étendre cette théorie en ajoutant d'autres axiomes : l'axiome de l'infini, l'axiome de descriptions, l'axiome du choix et les axiomes d'extensionnalité. Les axiomes d'extensionnalité peuvent se formuler ainsi :

$$\forall f\ \forall g\ ((\forall x\ (f\ x) = (g\ x)) \Rightarrow f = g)$$
$$\forall x\ \forall y\ ((\varepsilon(x) \Leftrightarrow \varepsilon(y)) \Rightarrow x = y)$$

2.3 Le symbole ε

2.3.1 Les contenus propositionnels

La principale différence entre cette présentation de la théorie des types et les présentations plus traditionnelles, qui insistent moins sur la possibilité d'exprimer cette théorie comme une théorie du premier ordre ou une théorie modulo, est l'introduction du symbole ε et corrélativement la distinction entre les propositions et leur contenu et entre les connecteurs et les quantificateurs (par exemple, $\Rightarrow$) et leur contenu (par exemple, $\dot{\Rightarrow}$).

Cette distinction est essentielle pour exprimer la théorie des types comme une théorie du premier ordre ou une théorie modulo, puisqu'on ne peut, dans une telle théorie, quantifier que sur des objets, et il faut donc, pour pouvoir quantifier sur les propositions, réifier ces propositions, comme on réifie les prédicats unaires en théorie naïve des ensembles.

On montre dans [b] que quand on pose l'axiome d'extensionnalité

$$\forall x\ \forall y\ ((\varepsilon(x) \Leftrightarrow \varepsilon(y)) \Rightarrow x = y)$$

le contenu d'une proposition se réduit à sa valeur de vérité. Mais en l'absence d'un tel axiome, le contenu d'une proposition peut véhiculer beaucoup plus sur la proposition elle-même. Deux théorèmes qui expriment

des propriétés différentes peuvent avoir des contenus différents même s'ils sont démontrables l'un et l'autre. On vera toutefois au paragraphe 6.3.3 que certains traits syntaxiques de la proposition doivent être gommés de son contenu, faute de quoi, le symbole ε peut se voir comme le prédicat de vérité d'A. Tarski et la théorie obtenue est contradictoire.

2.3.2 Les termes, les propositions et les assertions

Cette distinction entre une proposition $\varepsilon(t)$ et son contenu t, ou *lexis* [61], est traditionnelle en grammaire où on distingue la proposition "Deux et deux sont quatre." du groupe nominal "que deux et deux sont quatre" qui remplit les fonctions d'un groupe nominal, par exemple la fonction d'objet dans la phrase "Je crois que deux et deux sont quatre." Elle permet donc d'utiliser les prédicats d'attitude propositionnelle comme des prédicats ordinaires.

Une terminologie possible pour distinguer ces deux expressions est de nommer l'expression "que deux et deux sont quatre" *proposition* et l'expression "Deux et deux sont quatre." *assertion*, *énoncé* ou *jugement*. Hélas, cette terminologie introduit une confusion. D'une part, l'acte d'asserter, d'énoncer ou de juger n'est pas plus présent dans $\varepsilon(t)$ que dans t, d'autre part, la terminologie de la logique du premier ordre oppose les *termes* qui sont les expressions désignant des objets et les *propositions* qui sont les expressions désignant des faits et il serait maladroit d'appeler le terme t *proposition* et la proposition $\varepsilon(t)$ *énoncé*. Nous préférons donc appeler le terme t *contenu propositionnel* ou *information* et la proposition $\varepsilon(t)$ *proposition*.

Une autre remarque terminologique est que le symbole ε pourrait être noté comme le symbole de Frege "$\vdash$", en suivant, par exemple, Whitehead et Russell qui distinguent la *lexis* p de l'*assertion* $\vdash p$. Cela n'est pas possible, car dans le métalangage (c'est-à-dire dans le langage que nous utilisons pour parler des théories logiques) nous utilisons le symbole $\vdash$ pour affirmer qu'une proposition est démontrable. Nous devons donc distinguer : le terme p, la proposition $\varepsilon(p)$ et la métaproposition $\vdash \varepsilon(p)$ affirmant que la proposition $\varepsilon(p)$ est démontrable. En théorie des ensembles, on distingue, de même, l'ensemble P des nombres pairs, la proposition $4 \in P$ et la métaproposition $\vdash 4 \in P$ affirmant que cette proposition est démontrable.

On pourrait objecter que puisqu'il y a, en théorie des types, une correspondance parfaite entre les propositions et certains termes, on pourrait se passer des propositions, c'est-à-dire considérer les propositions comme des termes comme les autres et n'utiliser que deux sortes d'expressions : le terme t et la métaproposition $\vdash t$ affirmant que t est démontrable. C'est, par exemple, le point de vue de A. Church [16] et de Andrews [6]. L'inconvénient de ce point de vue est que le fait de considérer les propositions comme des termes, c'est-à-dire les faits comme des objets, n'est pas "ontologiquement neutre". En effet, s'il est facile, et "relativement" neutre, de remplacer les symboles de prédicat et les connecteurs par des symboles de fonction et donc de transformer les propositions sans quantificateurs en termes, exprimer les propositions formées avec des quantificateurs comme des termes suppose, en revanche, des choix ontologiques lourds. En effet, les symboles $\dot\forall$ et $\dot\exists$ doivent être appliqués à des fonctions ou des ensembles, il faut donc faire relever la définition de ces notions de la logique et non de la théorie.

La logique du premier ordre (et la déduction modulo), en distinguant les termes (sans lieurs) des propositions (avec quantificateurs) parvient à expliquer la signification des quantificateurs (c'est-à-dire à formuler des règles de déduction) sans utiliser de fonctions ni d'ensembles. La notion de déduction se définit de manière semblable que la théorie contienne des fonctions et des ensembles ou non. Cela permet de séparer clairement la logique, ontologiquement neutre, de la théorie qui exprime les choix concernant les objets du discours. Un autre avantage est que beaucoup de propriétés (complétude, élimination des coupures, ...) et d'algorithmes (démonstration automatique, ...) peuvent être étudiés génériquement indépendamment de la théorie considérée.

Le fait que la théorie considérée, comme la théorie des types simple, permette d'exprimer des fonctions et des ensembles et donc des contenus propositionnels n'impose en rien de se priver de la notion de proposition introduite par la logique du premier ordre et des résultats et algorithmes généraux. Cela demande simplement de distinguer les propositions des contenus propositionnels.

2.4 Le langage des termes et l'intentionnalité

Nous l'avons rapidement dit plus haut : en théorie des types, skolémiser les schéma de compréhension donne un langage pour les objets de la théorie des types dans lequel on peut noter $x \longmapsto x$ la fonction identité et $\{x \mid \exists y\ x = 2 \times y\}$ l'ensemble des nombres pairs. Cependant, ce langage ne comporte pas de symboles lieurs $x \longmapsto ...$ et $\{x \mid ...\}$ dans toute leur généralité. On propose ici une variante plus souple de ce langage.

2.4.1 La substitution des variables de fonction et de prédicat

Certaines formulations anciennes (par exemple, [18]) de la théorie des types comprennent des variables de fonction et de prédicat, mais pas, à proprement parler, de termes de fonction ou de prédicat. Par exemple, le schéma d'axiome de récurrence s'écrit

$$\forall P\ ((P(0) \wedge \forall x\ (P(x) \Rightarrow P(Succ(x)))) \Rightarrow \forall x\ P(x))$$

mais il n'y a pas de terme $\{y \mid y + 0 = y\}$ qu'on puisse substituer à P. On utilise, en revanche, une nouvelle opération de substitution : substituer $P(y_1, ..., y_n)$ par une proposition A consiste à substituer chaque proposition de la forme $P(t_1, ..., t_n)$ par la proposition $[t_1/y_1, ..., t_n/y_n]A$.

Par exemple, en substituant la proposition $P(y)$ par la proposition $y+0 = y$ dans le schéma de récurrence, on obtient la proposition

$$(0 + 0 = 0 \wedge \forall x\ (x + 0 = x \Rightarrow Succ(x) + 0 = Succ(x))) \Rightarrow \forall x\ (x + 0) = x$$

Cette notion de substitution est notoirement difficile à définir correctement, surtout quand on sort du cadre de la logique du second ordre. De plus, on introduit ici une nouvelle opération, qui complique le formalisme et rend difficile l'application de résultats ou d'algorithmes généraux.

2.4.2 Le schéma de compréhension

Dans [46], L. Henkin propose de bannir la règle de substitution des variables de fonction et de prédicat dans la logique du second ordre en ajoutant le schéma d'axiome de compréhension. Il montre l'équivalence de son système avec la formulation "traditionnelle" de la logique du second ordre qui utilise cette règle de substitution.

Henkin démontre cette équivalence sans passer par une théorie intermédiaire, dans laquelle le schéma de compréhension est skolémisé. Il n'introduit donc pas réellement de notation pour les fonctions et les prédicats, mais uniquement des axiomes exprimant l'existence de ces objets.

2.4.3 Les combinateurs

Le schéma de compréhension de Henkin peut se généraliser à la théorie des types. Quand on skolémise ce schéma de compréhension, on introduit pour chaque terme purement applicatif t dont les variables libres sont parmi $x_1, ..., x_n$ un symbole $f_{x_1,...,x_n,t}$ et un axiome

$$\forall x_1\ ...\ \forall x_n\ (f_{x_1,...,x_n,t}\ x_1\ ...\ x_n) = t$$

Par exemple, on introduit un symbole S et l'axiome

$$\forall x\ \forall y\ \forall z\ (S\ x\ y\ z) = ((x\ z)\ (y\ z))$$

De même, on introduit pour chaque proposition A dont les variables libres sont parmi $x_1, ..., x_n$ un symbole $P_{x_1,...,x_n,A}$ et un axiome

$$\forall x_1\ ...\ \forall x_n\ ((P_{x_1,...,x_n,A}\ x_1\ ...\ x_n) \Leftrightarrow A)$$

On distingue alors la proposition $y + 0 = y$ et l'ensemble $P_{y,y+0=y}$ ou $\{y \mid y + 0 = y\}$, et de même le nombre $y + 0$ de la fonction $f_{y,y+0}$ ou $y \longmapsto y + 0$ et on peut utiliser la substitution ordinaire pour les variables de fonction et de prédicat. Ainsi, on introduit une nouvelle proposition

$$(\{y \mid y+0=y\}(0) \wedge \forall x\ (\{y \mid y+0=y\}(x) \Rightarrow \{y \mid y+0=y\}(Succ(x)))) \Rightarrow \forall x\ \{y \mid y+0=y\}(x)$$

qui est équivalente à la proposition

$$(0+0=0 \wedge \forall x\ (x+0=x \Rightarrow Succ(x)+0=Succ(x))) \Rightarrow \forall x\ (x+0)=x$$

Mais, contrairement à la présentation avec une substitution particulière pour les variables de prédicat, il est désormais nécessaire d'introduire des axiomes de compréhension skolémisés ou de *conversion* qui expriment la signification des notations $\{y_1, ..., y_n \mid P\}$ et $y_1, ..., y_n \longmapsto t$ et qui permettent de montrer l'équivalence des deux propositions ci-dessus.

A cause de leur ressemblance avec les supercombinateurs de R.M.J. Hughes [54] nous avons dans [a] appelé ces symboles *hypercombinateurs*. Cette terminologie est, en fait, assez mal choisie, car ces symboles sont précisément ceux que H.B. Curry appelle *combinateurs* dans [22, 23] (avant que ce terme ne serve à désigner des choses assez disparates, comme les termes clos du λ-calcul, ou les symboles de n'importe quel langage du premier ordre servant à exprimer des fonctions).

Dans une présentation utilisant la déduction modulo, les axiomes de conversion peuvent être transformés en règles de réécriture. Substituer une variable de fonction ou de prédicat par un combinateur puis normaliser la proposition obtenue donne le même résultat que la substitution définie au paragraphe 2.4.1. On montre ainsi, au moins pour la logique du second ordre, que la formulation avec des combinateurs est une extension conservatrice de la formulation du paragraphe 2.4.1. Mais c'est précisément cette décomposition entre substitution et réduction qui simplifie le formalisme.

L'équivalence de la présentation de la théorie des types avec le schéma de compréhension et avec des combinateurs est une simple conséquence du théorème de Skolem (de plus, seuls les quantificateurs existentiels les plus extérieurs à la formule sont skolémisés, ce qui permet une démonstration d'équivalence particulièrement simple).

On peut noter $x_1, ..., x_n \longmapsto t$ le combinateur $f_{x_1,...,x_n,t}$. Mais l'impression d'avoir un symbole lieur dans le langage est trompeuse. D'une part toutes les variables de t doivent être liées dans $x_1, ..., x_n \longmapsto t$, on n'a donc pas le terme $x \longmapsto y$, d'autre part le terme t ne doit pas, lui-même, contenir de symboles de Skolem. On n'a donc pas le terme $x \longmapsto (y \longmapsto y)$, ni *a fortiori* le terme $x \longmapsto (y \longmapsto x)$ qui viole les deux conditions à la fois.

Pour construire un langage un peu plus libéral on peut tenter d'utiliser un schéma de compréhension plus général. Le schéma de compréhension fermé

$$\exists R\ \forall x_1\ ...\ \forall x_n\ (\varepsilon(R\ x_1\ ...\ x_n) \Leftrightarrow A)$$

$$\exists f\ \forall x_1\ ...\ \forall x_n\ (f\ x_1\ ...\ x_n) = t$$

dans lequel toutes les variables libres de A (resp. t) sont parmi $x_1, ..., x_n$ est équivalent au schéma de compréhension ouvert dans lequel cette condition est relâchée. Mais, on montre dans [a] que le langage obtenu en skolémisant ces deux schémas est très différent. En skolémisant l'instance du schéma de compréhension ouvert

$$\forall y\ \exists f\ \forall x\ (f\ x) = y$$

on introduit un symbole de fonction unaire f et l'axiome $\forall y\ \forall x\ (f(y)\ x) = y$. On peut noter $x \longmapsto y$, le terme $f(y)$ dans lequel la variable y est libre et qui est la fonction constante identiquement égale à y. On peut donc ainsi avoir des variables libres dans les abstractions. Il faut, cela dit, éviter de noter $x \longmapsto x$ le terme $f(x)$ qui est la fonction constante identiquement égale à x. Ce terme doit se noter $x' \longmapsto x$ dans lequel on a renommé la variable liée pour éviter la confusion avec la variable libre.

On peut aussi appliquer le symbole f à un terme contenant une abstraction, par exemple au terme $y \longmapsto y$ ce qui donne le terme $f(y \longmapsto y)$ qu'on peut noter $x \longmapsto (y \longmapsto y)$. En revanche, il faut éviter de noter

$x \longmapsto (y \longmapsto x)$ le terme $f(y \longmapsto x)$, mais il faut le noter $x' \longmapsto (y \longmapsto x)$. La variable libre dans $y \longmapsto x$ ne peut donc plus être abstraite. Il semble donc impossible de construire une fonction qui à x et y associe x en abstrayant les deux variables l'une après l'autre.

On montre dans [a] qu'il est nécessaire, pour construire cette fonction, d'utiliser une instance du schéma de compréhension binaire

$$\exists f \; \forall x \; \forall y \; (f \; x \; y) = x$$

En effet, cette instance du schéma de compréhension est indépendante du schéma de compréhension unaire, car on peut construire un modèle du schéma de compréhension unaire dans lequel cette proposition n'est pas valide. Le schéma de compréhension unaire est donc plus faible que le schéma de compréhension général. En revanche le schéma de compréhension ternaire est équivalent au schéma général puisque les instances correspondant aux combinateurs S et K suffisent. Nous laissons ouvert le problème de l'équivalence du schéma de compréhension binaire et du schéma de compréhension général.

2.4.4 Le λ-calcul

La λ-calcul introduit par Church [17] et utilisé, également par Church, en théorie des types [16] consiste à généraliser l'utilisation des lieurs de manière à pouvoir former le terme $x \longmapsto (y \longmapsto x)$ qu'on note alors $\lambda x \; \lambda y \; x$. Par rapport aux langages de combinateurs, le λ-calcul introduit une plus grande homogénéité puisqu'il devient possible de lier n'importe quelle variable, y compris celles qui apparaissent libres dans les abstractions.

On pose alors l'axiome (β)

$$(\lambda x \; t) \; u = [u/x]t$$

qui peut se transformer en une règle de réécriture, la règle de *β-réduction*

$$(\lambda x \; t) \; u \to [u/x]t$$

Cependant, le λ-calcul introduit une nouvelle difficulté qui est la nécessité de substituer sous les abstractions. Les formulations de la théorie des types qui utilisent des combinateurs peuvent facilement s'exprimer comme des théories du premier ordre (en introduisant des symboles d'application $\alpha_{T,U}$ et un symbole ε). La substitution dans les termes est donc la substitution simple (le remplacement) de la logique du premier ordre. Ce n'est plus le cas avec la formulation de la théorie des types utilisant le λ-calcul qui demande à nouveau une substitution particulière : quand on substitue la variable x par le terme u dans le terme $\lambda y \; x$, il faut substituer x par u sous le λ, et quand on substitue la variable x par le terme y dans le terme $\lambda y \; x$, il faut, en outre, renommer la variable liée y de manière à éviter les captures.

L'objectif de bannir la règle de substitution des variables de fonction et de prédicat, qui nous avait amené au langage des combinateurs, est donc partiellement abandonné puisqu'il faut à nouveau introduire une substitution particulière. Cette substitution est à l'origine de bien des bizarreries de la formulation de la théorie des types utilisant le λ-calcul, par exemple du traitement assez subtil de la portée des variables dans l'algorithme d'unification d'ordre supérieur de G. Huet [52, 53, 86]. Ce point est discuté dans [c]. Cependant, la simplicité et l'uniformité du λ-calcul font que cette formulation de la théorie des types est souvent préférée à celle utilisant des combinateurs.

L'équivalence entre les formulations utilisant la règle de substitution des variables de prédicats et de fonctions, le schéma de compréhension et des combinateurs est simple à établir. En revanche, l'équivalence de ces formulations avec celles utilisant le λ-calcul est un peu plus difficile. L'idée est de traduire les termes du λ-calcul dans le langage des combinateurs. Cette traduction s'appelle *l'ascension des variables (λ-lifting)* [54, a]. Le seul cas intéressant est celui des abstractions. Pour traduire un terme de la forme $\lambda x_1 \; ... \; \lambda x_n \; t$, on commence par traduire le terme t. On obtient un terme t' qui contient, outre les variables $x_1, ..., x_n$, des variables $y_1, ..., y_p$ et des combinateurs $c_1, ..., c_q$. On remplace ces combinateurs par des variables $z_1, ..., z_q$, ce qui donne un terme t'' et on définit la traduction du terme $\lambda x_1 \; ... \; \lambda x_n \; t$ comme $((y_1, ..., y_p, z_1, ..., z_q, x_1, ..., x_n \longmapsto t'') \; y_1 \; ... \; y_p \; c_1 \; ... \; c_q)$.

Le problème est que pour montrer l'équivalence de ces deux formulations de la théorie des types, on a besoin de l'axiome d'extensionnalité dans les deux formulations de la théorie. Par exemple, dans la formulation utilisant le λ-calcul, la proposition

$$((\lambda x\ \lambda y\ \lambda z\ x)\ w\ w) = ((\lambda x\ \lambda y\ \lambda z\ y)\ w\ w)$$

est démontrable car elle se réduit sur

$$\lambda z\ w = \lambda z\ w$$

En revanche sa traduction

$$((x, y, z \longmapsto x)\ w\ w) = ((x, y, z \longmapsto y)\ w\ w)$$

demande l'axiome d'extensionnalité pour être démontrée. On montre dans [a], en construisant un modèle très simple, qu'en l'absence de cet axiome, cette proposition est indépendante de la théorie des types.

En fait, on peut dire que l'axiome β contient un soupçon d'extensionnalité car il permet la substitution sous les abstractions et identifie donc les termes $((\lambda x\ \lambda y\ \lambda z\ x)\ w\ w)$ et $\lambda z\ w$, qui en toute rigueur ne sont pas intentionnellement identiques. Ce point sera discuté au paragraphe 2.4.7.

De plus, diverses variantes de la traduction du λ-calcul dans le langage des combinateurs donnent des résultats un peu différents. Par exemple, si on traduit les abstractions en bloc, comme ci-dessus, dans la proposition

$$((\lambda x\ \lambda y\ \lambda z\ x)\ w\ w) = ((\lambda y\ \lambda x\ \lambda z\ x)\ w\ w)$$

on obtient une proposition qui ne peut pas se démontrer sans l'axiome d'extensionnalité. En revanche, si on traduit les abstractions une par une, on obtient la proposition

$$((d, e, x \longmapsto (d\, e\, x))\ (x, c, y \longmapsto (c\, y))\ (x, z \longmapsto x)\ w\ w) = ((d, e, y \longmapsto (d\, e))\ (c, x \longmapsto (c\, x))\ (x, z \longmapsto x)\ w\ w)$$

qui est démontrable sans l'axiome d'extensionnalité, car elle se réduit sur

$$((x, z \longmapsto x)\ w) = ((x, z \longmapsto x)\ w)$$

Une autre traduction, qui n'utilise que les combinateurs S et K [10, 48, 59], utilise l'extensionnalité moins souvent encore, mais ne peut pas non plus s'en passer totalement.

On peut alors se demander si le problème n'est pas davantage dans la traduction que dans le λ-calcul lui-même. En faisant l'hypothèse que toutes les traductions conservent les termes purement applicatifs (c'est-à-dire sans abstractions) du λ-calcul, cette question est formulée dans [a] comme le problème, que nous laissons ouvert, de trouver une proposition sans abstractions qui soit démontrable dans la présentation utilisant le λ-calcul, mais pas dans celle utilisant les combinateurs.

2.4.5 Le calcul des substitutions explicites

Si on prend la formulation de la théorie des types utilisant le λ-calcul comme référence, les combinateurs permettent de donner une présentation au premier ordre qui est extensionnellement équivalente, c'est-à-dire que les deux formulations sont équivalentes quand l'axiome d'extensionnalité est posé dans les deux cas. Si on ne pose pas l'axiome d'extensionnalité, les deux théories ne sont plus équivalentes, et on doit poser des axiomes d'extensionnalité faible (les axiomes de Curry [10, 48, 59]) dans la formulation utilisant les combinateurs de manière à obtenir l'équivalence. Ces axiomes sont notoirement compliqués et on ne sait pas bien les transformer en règles de réécriture dans une théorie modulo.

Parallèlement au langage des combinateurs, les indices de N.G. De Bruijn [25], puis les combinateurs catégoriques [19] et le calcul des substitutions explicites ou $\lambda\sigma$-calcul [1, 21] ont fourni d'autres langages du premier ordre pour noter les fonctions et, comme nous allons le voir, d'autres formulations au premier ordre de la théorie des types qui sont intentionnellement équivalentes à la formulation utilisant le λ-calcul.

Les indices de De Bruijn sont, au départ, une notation alternative pour le λ-calcul. S'appuyant sur l'idée que les noms des variables liées servent uniquement à relier une occurrence d'une telle variable à son lieur, De

Bruijn a proposé de remplacer chaque occurrence de variable liée par un nombre entier, qui peut s'assimiler à l'adresse relative du lieur et qui est le nombre de lieurs qu'il faut traverser pour arriver à celui qui correspond à cette variable.

Ainsi, le terme $\lambda x\ \lambda y\ (x\ y\ \lambda z\ (z\ y))$, par exemple, s'écrit $\lambda\lambda(2\ 1\ \lambda(1\ 2))$. Ce terme peut, à son tour, se lire comme un terme d'un langage du premier ordre contenant une infinité de symboles d'individu 1, 2, 3, ..., un symbole de fonction unaire λ et un symbole de fonction binaire α pour l'application.

On peut remarquer que ce langage contient bien davantage que le λ-calcul. Si le terme $\lambda 1$ est naturellement l'expression du terme $\lambda x\ x$, son sous-terme, le terme 1, ne représente aucun λ-terme, à moins de le lire dans un contexte de variables, par exemple, x, y, z, auquel cas, il exprime le terme ouvert x. Les termes ouverts du λ-calcul peuvent donc à la fois s'exprimer comme des termes clos dans un contexte de variables ou comme des termes ouverts du λ-calcul avec des indices de De Bruijn. Par exemple, le terme x peut ou bien s'exprimer comme le terme 1 dans un contexte x, y, z ou bien comme le terme x.

Dans le λ-calcul avec des indices de De Bruijn, l'axiome de β-conversion s'exprime ainsi

$$((\lambda t)\ u) = [u/1]t$$

Cet axiome peut se transformer en une règle de réécriture, la β-reduction

$$((\lambda t)\ u) \to [u/1]t$$

La substitution du $[u/n]t$ désigne maintenant la substitution de l'indice de De Bruijn n par le terme u dans t, il ne s'agit donc pas de l'opération de substitution simple (de remplacement) mais d'une autre opération qui se définit ainsi par récurrence sur la structure de t

— $[u/n]x = x$,
— $[u/n](t_1\ t_2) = ([u/n]t_1\ [u/n]t_2)$,
— $[u/n]\lambda t = \lambda[\mathrm{lft}(u, 0)/n + 1]t$,
— $[u/n]m = m - 1$ si $m > n$,
— $[u/n]m = u$ si $m = n$,
— $[u/n]m = m$ si $m < n$.

où le terme $\mathrm{lft}(u, i)$ (qui se lit “incrémentation (*lift*) de u à partir de i”) est défini par récurrence sur la structure de u

— $\mathrm{lft}(x, i) = x$,
— $\mathrm{lft}((u_1\ u_2), i) = (\mathrm{lft}(u_1, i)\ \mathrm{lft}(u_2, i))$,
— $\mathrm{lft}(\lambda u, i) = \lambda(\mathrm{lft}(u, i + 1))$,
— $\mathrm{lft}(m, i) = m + 1$ si $m > i$,
— $\mathrm{lft}(m, i) = m$ si $m \leq i$.

Par exemple le terme $\lambda x\ \lambda y\ ((\lambda z\ \lambda w\ (x\ z))\ y)$, qui se réduit $\lambda x\ \lambda y\ \lambda w\ (x\ y)$, s'exprime avec des indices de De Bruijn par le terme $\lambda\lambda((\lambda\lambda(4\ 2))\ 1)$. Ce terme se réduit sur $\lambda\lambda([1/1]\lambda(4\ 2)) = \lambda\lambda\lambda([\mathrm{lft}(1, 0)/2](4\ 2)) = \lambda\lambda\lambda([2/2](4\ 2)) = \lambda\lambda\lambda(3\ 2)$ qui qui est bien la traduction du terme $\lambda x\ \lambda y\ \lambda w\ (x\ y)$.

Cette notion de substitution est assez délicate à définir car elle doit prendre en compte deux phénomènes : d'une part la nécessité de décrémenter les indices libres (c'est-à-dire désignant une variable liée au dessus du radical) dans t (par exemple la variable x exprimée par l'indice 4 dans le terme initial est décrémentée et exprimée par l'indice 3 dans le terme final), d'autre part la nécessité d'incrémenter les indices libres dans u quand on traverse une abstraction (par exemple, la variable y représentée par l'indice 1 dans le terme initial est représentée par l'indice 2 dans le terme final).

Ce calcul présente deux inconvénients. D'une part, comme c'était déjà le cas pour le λ-calcul, un algorithme de substitution extérieur aux règles de réécriture est nécessaire à la définition de la réduction. Ce système de réécriture contient, de ce fait, un nombre infini de règles. D'autre part, les variables libres ne sont pas affectées par ces opérations de substitution et d'incrémentation. De ce fait, l'opération de réduction sur les termes ouverts ne commute pas avec la substitution simple (le remplacement) d'une telle variable. Par exemple le terme $(\lambda(x\ 1)\ y)$ se réduit sur $(x\ y)$, mais si on substitue la variable x par le terme $(z\ 1)$ on obtient le terme $(\lambda(z\ 1\ 1)\ y)$ qui se réduit sur $(z\ y\ y)$ alors que si on substitue la variable x par le terme $(z\ 1)$ dans le terme $(x\ y)$, on obtient le terme $(z\ 1\ y)$.

En revanche la réduction commute avec une autre forme de substitution des variables qui correspond à la substitution du λ-calcul et qui demande d'incrémenter les indices libres du terme substitué quand on passe sous un λ. Ainsi, le terme $(\lambda(x\ 1)\ y)$ se réduit sur $(x\ y)$, et si on substitue la variable x par le terme $(z\ 1)$ on obtient le terme $(\lambda(z\ 2\ 1)\ y)$ qui se réduit sur $(z\ 1\ y)$ qui est bien le terme obtenu en substituant la variable x par le terme $(z\ 1)$ dans le terme $(x\ y)$.

Le calcul des combinateurs catégoriques et le calcul des substitutions explicites pallient ces inconvénients du λ-calcul avec des indices de De Bruijn. D'une part, il permettent de ne plus recourir à une opération externe de substitution et ils ont un nombre fini de règles. En corollaire de cela, la substitution des variables libres et la réduction commutent, comme dans tout système de réécriture du premier ordre.

L'idée de ce calcul est d'internaliser la notion de substitution comme un symbole du langage au même titre que λ ou α. On note $t[s]$ le terme t auquel est appliquée la substitution s. Les termes sont à présent de cinq formes : indices de De Bruijn, variables, applications d'un terme à un autre, abstractions et applications d'une substitution à un terme.

Cet opérateur prend en argument un terme et une *substitution explicite* qui est une liste $u_1, ..., u_n$ de termes. Des règles de réécriture permettent ensuite de réduire le terme $t[u_1, ..., u_n]$ en substituant dans t l'indice 1 par u_1, l'indice 2 par u_2, ..., l'indice n par u_n et en décrémentant de n tous les autres indices libres. Pour construire ces listes, on a besoin de la liste vide, notée id car c'est la substitution identité, et du symbole "cons" noté ".". Deux autres symboles sont encore nécessaires : la substitution $\uparrow$ qui incrémente les indices libres d'un terme et le symbole $\circ$ qui permet de composer les substitutions.

Les règles de réécriture du $\lambda\sigma$-calcul sont les suivantes

$$(\lambda a)b \rightarrow a[b.id]$$

$$(a\ b)[s] \rightarrow (a[s]\ b[s])$$
$$1[a.s] \rightarrow a$$
$$a[id] \rightarrow a$$
$$(\lambda a)[s] \rightarrow \lambda(a[1.(s \circ \uparrow)])$$
$$(a[s])[t] \rightarrow a[s \circ t]$$
$$id \circ s \rightarrow s$$
$$\uparrow \circ\ (a.s) \rightarrow s$$
$$(s_1 \circ s_2) \circ s_3 \rightarrow s_1 \circ (s_2 \circ s_3)$$
$$(a.s) \circ t \rightarrow a[t].(s \circ t)$$
$$s \circ id \rightarrow s$$
$$1.\uparrow \rightarrow id$$
$$1[s].(\uparrow \circ\ s) \rightarrow s$$

La première règle est la β-réduction, les autres, appelées règles de σ-réduction, permettent de propager une substitution explicite dans un terme.

Dans ce calcul, comme dans tout système de réécriture du premier ordre, la réduction commute avec la substitution simple (le remplacement) des variables. Ainsi, le terme $(\lambda(x\ 1)\ y)$ se réduit sur $(x[y.id]\ y)$, et si on substitue la variable x par le terme $(z\ 1)$ on obtient le terme $(\lambda(z\ 1\ 1)\ y)$ qui se réduit sur $(z[y.id]\ y\ y)$ qui est le terme obtenu, après normalisation de la substitution, en substituant la variable x par le terme $(z\ 1)$ dans le terme $(x[y.id]\ y)$.

Quand on étend le λ-calcul avec des indices de De Bruijn en ajoutant des substitutions explicites, on peut envisager une autre traduction du λ-calcul dans ce formalisme. Dans la traduction présentée plus haut, quand on traduisait le terme $\lambda x\ (y\ x)$ en conservant y comme une variable libre on obtenait le terme $\lambda(y\ 1)$.

Dans le λ-calcul, si on substitue la variable y par le terme t, il faut éventuellement renommer la variable liée x de manière à éviter les captures. Dans le λ-calcul avec des indices de De Bruijn, il faut de même, comme nous l'avons vu, incrémenter les indices libres de t. Donc, bien que le λ-calcul avec des substitutions explicites soit un langage du premier ordre, la substitution des variables de ce langage du premier ordre qui correspond à celle du λ-calcul n'est pas la substitution simple (le remplacement). Cette opération de substitution peut se décomposer d'une part en une opération d'incrémentation et d'autre part en une substitution simple (un remplacement). Comme le $\lambda\sigma$-calcul contient des opérateurs explicites d'incrémentation, on peut préparer cette incrémentation en appliquant cet opérateur à la variable y, on obtient alors le terme $\lambda(y[\uparrow]\ 1)$ comme traduction de $\lambda x\ (y\ x)$. À présent, si on substitue simplement (on remplace) la variable y par le terme 1 on obtient le terme $\lambda(1[\uparrow]\ 1)$ c'est-à-dire $\lambda(2\ 1)$.

Dans [c], puis dans [i], nous avons appelé "précuisson" ("*pre-cooking*") cette traduction qui anticipe l'incrémentation du terme substitué. Si on note t_F la précuisson du λ-terme t, nous avons montré, dans [c], le résultat clé

$$([u/x]t)_F = [x \mapsto u_F]t_F$$

où t/x la substitution du λ-calcul et $x \mapsto t$ est la substitution simple (le remplacement). Autrement dit, la précuisson est un homomorphisme du λ-calcul dans le $\lambda\sigma$-calcul qui transforme la substitution d'un calcul en la substitution de l'autre.

Pour utiliser le calcul des substitutions explicites comme langage des fonctions de la théorie des types simples, il faut pouvoir associer un type à chaque terme du calcul. Ici, un nouveau problème surgit. Si on sait donner le type $\iota \to \iota$ au terme $\lambda 1$, qui exprime le λ-terme $\lambda x\ x$, il est plus difficile, en revanche, de donner un type au terme 1 qui n'exprime un λ-terme que dans un contexte de variable. On doit donc connaître le type des variables de ce contexte pour pouvoir associer un type à ce terme [1, 20]. Si on considère un contexte de trois variables x, y, z de types respectifs A, B, C alors le terme 1 a le type A, le terme 2 le type B, le terme 3 le type C et le terme 4 n'est pas bien typé.

Les jugements de typage sont donc de la forme $\Gamma \vdash t : T$ où Γ est un *contexte de typage*, c'est-à-dire une liste de types. Pour exprimer le $\lambda\sigma$-calcul typé comme un langage du premier ordre multisorté, nous avons proposé dans [c] puis dans [i], de considérer des sortes de la forme $\Gamma \vdash T$ où T est un type simple et Γ une liste de types simples. Les substitutions qui sont des listes de termes ont naturellement des types qui sont des listes de types simples. Comme les termes, elles doivent être typées dans un contexte. Elles ont donc des sortes de la forme $\Gamma \vdash \Delta$ où Γ et Δ sont des listes de types simples.

On doit ensuite, comme c'était déjà le cas avec les combinateurs, distinguer divers symboles en fonction de leur sorte. Par exemple, on a un nombre infini de constantes de 1^Γ_A, la constante 1^Γ_A étant de sorte $A\Gamma \vdash A$. On aboutit au langage suivant.

1^Γ_A	de sorte	$A\Gamma \vdash A$
$\alpha^\Gamma_{A,B}$	de rang	$(\Gamma \vdash A \to B, \Gamma \vdash A, \Gamma \vdash B)$
$\lambda^\Gamma_{A,B}$	de rang	$(A\Gamma \vdash B, \Gamma \vdash A \to B)$
$[]^{\Gamma,\Gamma'}_A$	de rang	$(\Gamma' \vdash A, \Gamma \vdash \Gamma', \Gamma \vdash A)$
id^Γ	de sorte	$\Gamma \vdash \Gamma$
$\uparrow^\Gamma_A$	de sorte	$A\Gamma \vdash \Gamma$
$\cdot^{\Gamma,\Gamma'}_A$	de rang	$(\Gamma \vdash A, \Gamma \vdash \Gamma', \Gamma \vdash A\Gamma')$
$\circ^{\Gamma,\Gamma',\Gamma''}$	de rang	$(\Gamma \vdash \Gamma'', \Gamma'' \vdash \Gamma', \Gamma \vdash \Gamma')$

Le $\lambda\sigma$-calcul typé avec des variables de termes est confluent [83] et il termine faiblement [74, 75, 76, 41] mais P.-A. Melliès [68] a montré qu'il ne termine pas fortement.

À ces symboles, nous avons ajouté dans [i] les symboles

$\dot{\Rightarrow}$	de sorte	$\vdash o \to o \to o$
$\dot{\wedge}$	de sorte	$\vdash o \to o \to o$
$\dot{\vee}$	de sorte	$\vdash o \to o \to o$
$\dot{\neg}$	de sorte	$\vdash o \to o$
$\dot{\bot}$	de sorte	$\vdash o$
$\dot{\forall}_T$	de sorte	$\vdash (T \to o) \to o$
$\dot{\exists}_T$	de sorte	$\vdash (T \to o) \to o$

et le symbole de prédicat

$$\varepsilon \quad \text{de rang} \quad (\vdash o)$$

Les variables libres dans ce langage ont une sorte de la forme $\Gamma \vdash T$, c'est-à-dire qu'à chaque variable est associé non seulement un type, mais aussi un contexte qui est celui dans lequel les termes qu'on substitue à cette variable doivent être typés. On peut montrer que si un λ-terme t contenant des variables libres $x_1, ..., x_n$ de type $T_1, ..., T_n$ est bien typé alors sa précuisson est un $\lambda\sigma$-terme bien typé contenant les variables libres $x_1, ..., x_n$ de sorte $\vdash T_1, ..., \vdash T_n$.

Sur ce langage, on définit ensuite le système de réécriture de la figure 2.2. La première règle est la règle de β-réduction, le groupe suivant est constitué des règles de σ-réduction, le dernier groupe est constitué des règles de réduction associées aux symboles $\dot{\Rightarrow}$, $\dot{\wedge}$, ...

On peut également ajouter une règle de réduction correspondant à la η-réduction du λ-calcul, qui permet d'incorporer dans la réduction une partie plus importante de l'axiome d'extensionnalité :

$$\lambda(a\ 1) \to b \text{ si } a =_\sigma b[\uparrow]$$

On montre dans [i] que ce système de réécriture est confluent. On montre également qu'il est faiblement normalisable en le plongeant dans le $\lambda\sigma$-calcul typé.

On montre dans [i] que la théorie modulo constituée de ce système de réécriture et de l'ensemble vide d'axiomes est intentionnellement équivalente à la formulation de la théorie des types utilisant le λ-calcul. En effet, une proposition t est démontrable dans la théorie des types utilisant le λ-calcul si et seulement si la proposition $\varepsilon(t_F)$ est démontrable dans cette présentation de la théorie des types. De plus, la structure des démonstrations est conservée : les étapes de calcul correspondent aux étapes de calcul, les étapes de déduction aux étapes de déduction. Contrairement à la formulation de la théorie des types avec des combinateurs, il n'est jamais nécessaire de faire appel aux axiomes d'extensionnalité ou aux axiomes de Curry pour démontrer une proposition qui se démontre sans ces axiomes dans la théorie des types utilisant le λ-calcul.

Cette formulation de la théorie des types est celle que nous utiliserons dans les chapitres suivants.

Nous allons, cependant, conclure ce paragraphe consacré aux langages des termes en théorie des types par la présentation de deux autres langages qui sont plus prospectifs.

2.4.6 Les combinateurs, à nouveau

On construit, dans [i], un modèle de la théorie des types simples utilisant les substitutions explicites. Les sortes de cette théorie étant de la forme $\Gamma \vdash T$ et $\Gamma \vdash \Delta$, on doit donner un ensemble $\mathcal{D}_{\Gamma \vdash T}$ ou $\mathcal{D}_{\Gamma \vdash \Delta}$ pour chacune de ces sortes. Cette construction se fait en deux étapes. On construit, dans un premier temps, une famille d'ensembles $\mathcal{M}_T$ en partant de deux ensembles $\mathcal{M}_\iota$ et $\mathcal{M}_o$ et en définissant $\mathcal{M}_{T \to U}$ comme l'ensemble des fonctions de $\mathcal{M}_T$ dans $\mathcal{M}_U$, puis dans un second temps les ensembles $\mathcal{D}_{\Gamma \vdash T}$ et $\mathcal{D}_{\Gamma \vdash \Delta}$, encore une fois comme des ensembles de fonctions. L'ensemble $\mathcal{D}_{T_1, ..., T_n \vdash U}$ est défini comme ${\mathcal{M}_U}^{\mathcal{M}_{T_1} \times ... \times \mathcal{M}_{T_n}}$ ou mieux, en curryfiant, comme l'ensemble ${\mathcal{M}_U}^{{\mathcal{M}_{T_n}}^{...^{\mathcal{M}_{T_1}}}}$. De même, l'ensemble $\mathcal{D}_{T_1, ..., T_n \vdash U_1, ..., U_p}$ est défini comme $(\mathcal{M}_{U_1} \times ... \times \mathcal{M}_{U_p})^{{\mathcal{M}_{T_n}}^{...^{\mathcal{M}_{T_1}}}}$.

Cette double notion de fonctionnalité (puisque la flèche $\to$ et le symbole $\vdash$ dans les sortes $\Gamma \vdash \Delta$ sont interprétés, l'une comme l'autre comme des constructeurs d'espaces fonctionnels) rappelle l'origine catégorique du calcul des substitutions explicites, puisque les catégories cartésiennes closes comportent deux flèches : celle des morphismes, ici $\vdash$, et l'exponentielle, ici $\to$.

$$(\lambda a)b \to a[b.id]$$

$$(a\ b)[s] \to (a[s]\ b[s])$$
$$1[a.s] \to a$$
$$a[id] \to a$$
$$(\lambda a)[s] \to \lambda(a[1.(s \circ \uparrow)])$$
$$(a[s])[t] \to a[s \circ t]$$
$$id \circ s \to s$$
$$\uparrow \circ (a.s) \to s$$
$$(s_1 \circ s_2) \circ s_3 \to s_1 \circ (s_2 \circ s_3)$$
$$(a.s) \circ t \to a[t].(s \circ t)$$
$$s \circ id \to s$$
$$1. \uparrow \to id$$
$$1[s].(\uparrow \circ s) \to s$$

$$\varepsilon(\dot{\Rightarrow}\ x\ y) \to \varepsilon(x) \Rightarrow \varepsilon(y)$$
$$\varepsilon(\dot{\wedge}\ x\ y) \to \varepsilon(x) \wedge \varepsilon(y)$$
$$\varepsilon(\dot{\vee}\ x\ y) \to \varepsilon(x) \vee \varepsilon(y)$$
$$\varepsilon(\dot{\neg}\ x) \to \neg\varepsilon(x)$$
$$\varepsilon(\dot{\bot}) \to \bot$$
$$\varepsilon(\dot{\forall}_T\ x) \to \forall y\ \varepsilon(x\ y)$$
$$\varepsilon(\dot{\exists}_T\ x) \to \exists y\ \varepsilon(x\ y)$$

Figure 2.2 – Les règles de réécriture de la théorie des types simples avec des substitutions explicites

$$(app^n\ (app^p\ a\ b)\ c) \to (app^{p-1}(app^n\ a\ c)\ (app^n\ b\ c)) \text{ si } n < p \quad (1)$$
$$(app^n\ 1^p\ c) \to c \text{ si } p = n+1 \quad (2)$$
$$(app^n\ 1^p\ c) \to 1^{p-1} \text{ si } n+1 < p \quad (3)$$
$$(app^n\ (\uparrow^p\ a)\ b) \to a \text{ si } p = n \quad (4)$$
$$(app^n\ (\uparrow^p\ a)\ b) \to (\uparrow^{p-1}\ (app^n\ a\ b)) \text{ si } n < p \quad (5)$$
$$(\uparrow^n\ (app^p\ a\ b)) \to (app^{p+1}\ (\uparrow^n\ a)\ (\uparrow^n\ b)) \text{ si } n \le p \quad (6)$$
$$(\uparrow^n\ 1^p) \to 1^{p+1} \text{ si } n < p \quad (7)$$
$$(\uparrow^n\ (\uparrow^p a)) \to (\uparrow^{p+1}\ (\uparrow^n a)) \text{ si } n \le p \quad (8)$$

Figure 2.3 – Règles étendues pour les combinateurs

On interprète ensuite facilement les symboles du langage dans ce modèle. Un point intéressant est que le symbole λ s'interprète par la fonction identité. En effet, le fait d'interpréter les termes ouverts comme des fonctions de leurs variables libres fait que le terme t dans le contexte $A\Gamma$ s'interprète comme le terme λt dans le contexte Γ. Cette remarque n'est pas nouvelle : dans la traduction du λ-calcul dans les combinateurs catégoriques, le symbole λ s'interprète par un combinateur de curryfication, ce qui serait le cas ici aussi si on avait défini $\mathcal{D}_{T_1,\dots,T_n \vdash U}$ comme ${\mathcal{M}_U}^{\mathcal{M}_{T_1} \times \dots \times \mathcal{M}_{T_n}}$ au lieu de le définir comme ${\mathcal{M}_U}^{{\mathcal{M}_{T_n}}^{\dots^{\mathcal{M}_{T_1}}}}$.

Cela amène à se demander si on ne pourrait pas simplifier le formalisme en utilisant le type $T_n \to \dots \to T_1 \to U$ au lieu de la sorte $T_1, \dots, T_n \vdash U$ et en supprimant le symbole λ du langage, car à quoi bon écrire λt si ce terme exprime la même fonction que t ? Le symbole $1_U^{T_1,\dots,T_n}$ de sorte $U, T_1, \dots, T_n \vdash U$ a désormais un type simple $T_n \to \dots \to T_1 \to U \to U$, et il dénote la fonction qui à $a_n, \dots, a_1, b$ associe b. C'est une généralisation du combinateur K. Dans un cadre non typé, on aurait des symboles 1^n simplement paramétrés par leur arité. Le symbole α, se traduit dans ce langage par un combinateur *app* qui est une généralisation du combinateur S.

Pour construire les autres indices de De Bruijn, on a en outre besoin d'un combinateur $\uparrow$ de type $(A_1 \to \dots \to A_n \to B) \to A_1 \to \dots \to A_n \to C \to B$ qui est la fonction qui à $a, b_1, \dots, b_n, c$ associe $(a\ b_1\ \dots\ b_n)$.

Une substitution, en revanche, se traduit par une suite finie de termes. L'application d'une substitution à un terme se traduit également par le combinateur *app*.

Ces symboles sont donc des combinateurs au sens du paragraphe 2.4.3 et les premières règles de réécriture qu'on voudrait poser sont

$$(1^n\ x_1\ \dots\ x_n) \to x_n$$

$$(\uparrow^n\ a\ x_1\ \dots\ x_n\ y\ ) \to (a\ x_1\ \dots\ x_n)$$

$$(app^n\ a\ b\ x_1\ \dots\ x_n) \to (a\ x_1\ \dots\ x_n\ (b\ x_1\ \dots\ x_n))$$

On traduit un λ-terme t exprimé avec des indices de De Bruijn dans un contexte de longueur n comme le terme $|t|_n$.

— $|k|_n = (\uparrow^{n-1}\ (\dots(\uparrow^{n-k+1}\ 1^{n-k+1})\dots))$
— $|(a\ b)|_n = (app^n\ |a|_n\ |b|_n)$
— $|\lambda a|_n = |a|_{n+1}$.

Il est facile de voir que ce système de réécriture ne simule pas la β-réduction. Un β-radical de la forme $((\lambda a)\ b)$ se traduit par le terme $(app^n\ |a|_{n+1}\ |b|_n)$ qui n'est pas, en général, un radical. Un tel β-radical est caractérisé par le fait que l'argument du symbole app^n est traduit dans un contexte plus long que n. Il faut ici distinguer les différents cas en fonction de la forme de a. Si a est par exemple l'indice de De Bruijn 1 on obtient le terme $(app^n\ 1^{n+1}\ |b|_n)$. Ce terme n'est pas réductible par le système ci-dessus et la règle du $\lambda\sigma$-calcul $((\lambda a)\ b) \to a[b.id]$ suggère la règle $(app^n\ 1^{n+1}\ b) \to b$. L'étude des différents cas amène au système de réécriture de la figure 2.3.

On peut montrer, quoi que cela soit fastidieux, que si $a \to_\beta b$ alors $|a|_n \to |b|_n$. La réciproque de ce résultat, en revanche, est fausse car le terme $((\lambda(1\ y))\ z)$ se traduit par $(app^2\ (app^3\ 1^3\ (\uparrow^2\ 1^2))\ (\uparrow^1\ 1^1))$ qui se réduit sur $(app^2\ (app^2\ 1^3\ (\uparrow^1\ 1^1))\ (app^2\ (\uparrow^2\ 1^2)\ (\uparrow^1\ 1^1)))$ qui est la traduction de $(((\lambda 1)\ z)\ ((\lambda y)\ z))$. On paie ici le prix de la confusion entre application d'un terme à un autre et d'une substitution à un terme. Une solution est peut-être d'introduire deux variantes du combinateur *app*.

On peut remarquer que si la règle $(app^n\ 1^{n+1}\ b) \to b$ n'est pas une conséquence du système de réécriture naïf ci-dessus, c'est une conséquence de ce système de réécriture et de l'axiome d'extensionnalité. En effet le terme $(app^n\ 1^{n+1}\ b\ x_1\ ...\ x_n)$ se réduit sur $(1^{n+1}\ x_1\ ...\ x_n\ (b\ x_1\ ...\ x_n))$ puis sur $(b\ x_1\ ...\ x_n)$. On peut montrer qu'il en est de même des autres règles. On retrouve ici cette idée que le λ-calcul contient une forme faible d'extensionnalité qui est absente du langage des combinateurs. La solution traditionnelle à ce problème est d'ajouter les axiomes de Curry à la théorie des combinateurs, mais à la différence des axiomes de Curry, l'extension proposée ici s'exprime sous forme d'un ensemble de règles de réécriture et non d'axiomes équationnels.

Alors que je travaillais sur ce système de réécriture, j'ai appris qu'il avait été proposé indépendamment, à quelques variations près par H. Goguen et J. Goubault-Larrecq [40]. Goguen et Goubault sont allés beaucoup plus loin que moi dans l'étude de ce système. En particulier, ils ont démontré sa confluence, alors que je n'avais montré que la confluence locale, et ils ont montré que le calcul simplement typé n'avait que la propriété de terminaison faible. Il est remarquable aussi que le contre-exemple à la terminaison forte qu'ils proposent ait été découvert par un programme de recherche aléatoire et n'ait qu'un lointain rapport avec le contre-exemple proposé par Melliès à la forte normalisation du $\lambda\sigma$-calcul typé.

2.4.7 Le calcul faible des substitutions explicites

On l'a dit plusieurs fois, le λ-calcul contient une forme d'extensionnalité faible qui n'est pas présente dans le calcul des combinateurs. Cette forme d'extensionnalité est la possibilité de substituer sous un λ, ce qui permet de transformer le terme $((\lambda x\ \lambda y\ \lambda z\ x)\ w\ w)$ ou le terme en $((\lambda x\ \lambda y\ \lambda z\ y)\ w\ w)$ en $\lambda z\ w$ alors que dans le calcul des combinateurs le terme $((x, y, z \longmapsto t)\ w\ w)$ ne peut pas se réduire. Cela est dû au fait qu'en λ-calcul, la possibilité de substituer sous les abstractions permet de n'avoir que des fonctions unaires et d'itérer la construction de fonctions unaires pour construire les fonctions n-aires, alors que cela est impossible dans le calcul des combinateurs qui est essentiellement n-aire.

Le calcul des substitutions explicites, permet d'avoir la même forme d'extensionnalité que le λ-calcul tout en restant un langage de premier ordre. La possibilité de substituer sous les abstractions dans ce calcul s'exprime par la règle

$$(\lambda a)[s] \to \lambda(a[1.s\circ \uparrow])$$

On peut vouloir garder la notation du λ-calcul qui permet d'abstraire une variable libre dans un terme, où que soit l'occurrence de cette variable, mais rejeter la β-réduction et l'extensionnalité qu'elle contient. En effet, on peut argumenter que du point de vue intentionnel les termes $((\lambda x\ \lambda y\ \lambda z\ x)\ w\ w)$ et $\lambda z\ w$ n'expriment pas le même algorithme et que c'est une erreur du λ-calcul que d'identifier ces termes (argumentation, qui demande, cela dit, d'avoir résolu l'épineux problème de l'égalité intentionnelle de deux fonctions).

J.R. Hindley propose dans [47] une notion de réduction dans le λ-calcul sans substitution sous les abstractions et montre son équivalence avec celle du calcul des combinateurs. Dans le $\lambda\sigma$-calcul, interdire la propagation des substitutions sous les abstractions est particulièrement simple, puisqu'il suffit de supprimer la règle ci-dessus. On aboutit alors au $\lambda\sigma$-calcul faible [21, 66, 63, 44].

Il devient alors naturel de se poser la question de l'équivalence entre le $\lambda\sigma$-calcul faible et le calcul des combinateurs, puisque l'un comme l'autre revendiquent la pureté intentionnelle. Hélas, l'ascension des variables, traduction du $\lambda\sigma$-calcul faible dans le langage des combinateurs, ne respecte pas la réduction. Par exemple, les termes $\lambda y\ x$ et $((\lambda x\ \lambda y\ x)\ x)$ ne sont pas convertibles dans le $\lambda\sigma$-calcul faible, mais leur traduction est la même : $((x, y \longmapsto x)\ x)$. Ici, le problème semble venir de l'ascension des variables elle-même, qui n'est pas intentionnellement neutre.

Cependant, un premier pas vers un théorème d'équivalence entre $\lambda\sigma$-calcul faible et combinateurs a été effectué par T. Strahm [88] qui a montré qu'on pouvait, en revanche, traduire les combinateurs dans le

$\lambda\sigma$-calcul faible en conservant la convertibilité.

2.4.8 En conclusion : quel langage ?

En conclusion de ce long paragraphe consacré aux langage des termes en théorie des types simples, on peut dire qu'on a le choix entre deux langages du premier ordre : celui des combinateurs et celui du calcul des substitutions explicites. Ce dernier est équivalent au λ-calcul ce qui signifie qu'une partie de l'extensionnalité est incluse dans les règles de réduction. On peut voir cela comme un point positif (davantage est transféré du raisonnement vers le calcul) ou comme un point négatif si on cherche la pureté intentionnelle. Le langage des combinateurs, en revanche, est plus intentionnel.

Le langage des substitutions explicites est plus complexe à définir que celui des combinateurs, mais les exemples sont plus simples à écrire du fait de la présence d'un opérateur d'abstraction.

Nous avons également présenté deux autres langages, les combinateurs étendus et le $\lambda\sigma$-calcul faible, qui sont plus prospectifs et demandent à être davantage étudiés.

2.5 La complétude

Formuler la théorie des types comme une théorie du premier ordre (ou une théorie modulo) permet de lui appliquer des théorèmes et des algorithmes généraux valables pour toutes les théories du premier ordre. Le premier exemple auquel on pense est le théorème de complétude.

C'est un résultat du folklore qu'on peut déduire le théorème de complétude de Henkin [45] de celui de Gödel en utilisant une formulation au premier ordre de la théorie des types. Nous avons détaillé la démonstration de ce résultat dans [b].

En appliquant le théorème de complétude de Gödel à une formulation de la théorie des types comme une théorie du premier ordre, on montre qu'une proposition P est démontrable en théorie des types si et seulement si elle est valide dans tous les modèles de cette théorie. La classe des modèles de la théorie des types est, cela dit, plus vaste que la classe des modèles de Henkin. Les modèles de Henkin sont les modèles égalitaires et fonctionnels de la théorie des types. Un premier lemme montre que la classe des modèles égalitaires valide les même propositions que la classe de tous les modèles des axiomes de l'égalité. Un second lemme, homologue du théorème de l'effondrement d'A. Mostovski, montre que la classe des modèles fonctionnels égalitaires valide les même propositions que la classe de tous les modèles égalitaires de l'axiome d'extensionnalité. Ce deuxième lemme n'est vrai que pour les modèles égalitaires et donc cette condition doit être incluse dans la définition des modèles de Henkin (ce que Henkin n'avait pas fait explicitement) comme l'avait déjà remarqué Andrews [5].

Ce théorème de complétude peut aussi s'appliquer à la théorie des types intentionnelle, c'est-à-dire privée de l'axiome d'extensionnalité, et on sait que les modèles de Henkin sont insuffisants pour cela, car ils valident toujours l'axiome d'extensionnalité.

2.6 La skolémisation

Un deuxième exemple de théorème général qui peut s'appliquer aux formulations de la théorie des types comme une théorie du premier ordre est le théorème de Skolem.

2.6.1 Le problème de la skolémisation en théorie des types

Dans la présentation de la théorie des types utilisant le λ-calcul, la règle de skolémisation est relativement complexe. La règle naïve qui consiste à skolémiser l'axiome

$$\forall x\ \exists y\ (P\ x\ y)$$

en introduisant un symbole f de type $T \to U$ où T est le type de x et U celui de y est incorrecte, puisque de la proposition
$$\forall x\ (P\ x\ (f\ x))$$
on peut déduire
$$\exists f\ \forall x\ (P\ x\ (f\ x))$$
qui n'est pas démontrable à partir de la proposition
$$\forall x\ \exists y\ (P\ x\ y)$$
car l'axiome du choix est indépendant de la théorie des types [4].

Une solution naturelle consiste à proposer que pour chaque terme t, on puisse construire un terme $(f\ t)$ tel que $(P\ t\ (f\ t))$ mais sans pour autant disposer de la fonction f dans sa totalité. Cela a amené D.A. Miller [70, 71] à proposer une extension du λ-calcul dans laquelle chaque symbole a un indice, son *indice de Skolem* qui indique le nombre d'arguments auxquels il faut obligatoirement l'appliquer pour former un terme. Ainsi, on skolémise la proposition
$$\forall x\ \exists y\ (P\ x\ y)$$
en introduisant un symbole f^1 d'indice de Skolem 1. Le terme $(f^1\ x)$ est bien formé, mais pas le symbole f^1 tout seul.

Hélas, cette restriction n'est pas suffisante, car si le symbole f^1 tout seul n'est pas un terme bien formé, le terme $\lambda x\ (f^1\ x)$, en revanche, l'est et de la proposition
$$\forall x\ (P\ x\ (f^1\ x))$$
on peut donc déduire
$$\forall x\ (P\ x\ ((\lambda x\ (f^1\ x))\ x))$$
puis
$$\exists f\ \forall x\ (P\ x\ (f\ x))$$

Miller a donc introduit une seconde restriction qui interdit la liaison par un λ des variables libres dans les arguments obligatoires d'un symbole de Skolem. Ainsi, le terme $\lambda x\ (f^1\ x)$ n'est pas non plus bien formé. Miller a montré que si on pose ces deux conditions on retrouve un théorème analogue au théorème de Skolem, c'est-à-dire que les propositions sans occurrences de symboles de Skolem sont conséquences d'une proposition si et seulement si elles sont conséquences de sa forme skolémisée.

Il faut noter, cela dit, que si on considère, comme c'est l'habitude en théorie des types que le λ-terme $\forall x\ P$ est une notation pour le terme $\forall\ (\lambda x\ P)$ où $\forall$ est une constante de type $(T \to o) \to o$ alors la proposition skolémisée elle-même
$$\forall x\ (P\ x\ (f^1\ x))$$
n'est pas bien formée : en effet la seconde condition de Miller interdit de lier par un λ la variable x libre dans un argument obligatoire de f^1.

On doit donc ou bien introduire les quantificateurs comme des symboles lieurs supplémentaires ou bien donner une forme plus restreinte de la règle de skolémisation. Par exemple, si on utilise la skolémisation lors de la mise en forme clausale d'une proposition à réfuter par la méthode de résolution [84], comme le quantificateur universel est lui-même appelé à être supprimé, on peut définir la forme clausale de la proposition
$$\forall x\ \exists y\ (P\ x\ y)$$
comme la proposition (bien formée)
$$(P\ X\ (f^1\ X))$$
et le théorème de Skolem prend la forme de la complétude et correction de cette transformation (c'est-à-dire la forme clausale de la négation d'une proposition P est réfutable par résolution si et seulement si la proposition P est démontrable au sens ordinaire du terme). Dans [70, 71] Miller formule son théorème comme la correction de cette transformation pour la méthode des connections.

2.6.2 La skolémisation dans la présentation de la théorie des types utilisant les combinateurs

La présentation de la théorie des types utilisant des combinateurs étant une théorie du premier ordre, on peut lui appliquer le théorème de Skolem.

L'axiome

$$\forall x\ \exists y\ \varepsilon(\alpha(\alpha(P,x),y))$$

se skolémise en introduisant un symbole de fonction f de rang (T,U) où T est le type de x et U le type de y et l'axiome

$$\forall x\ \varepsilon(\alpha(\alpha(P,x),f(x)))$$

Le symbole f est un symbole de fonction de rang (T,U) et non un symbole d'individu de type $T \rightarrow U$ et on écrit donc $f(x)$ et non $\alpha(f,x)$. De ce fait, il va de soi que le symbole f tout seul n'est pas un terme, car dans un langage du premier ordre, les symboles de fonction d'arité non nulle ne sont pas des termes. On retrouve ainsi la première condition de Miller. La seconde condition n'a pas lieu d'être, puisque dans le langage des combinateurs, il n'y a pas d'opérateur d'abstraction. Enfin, comme dans tout langage du premier ordre on dispose de quantificateurs indépendamment des symboles $\dot{\forall}$ et $\dot{\exists}$ de ce langage particulier. Il n'y a donc pas de problème de formulation de la proposition skolémisée

$$\forall x\ \varepsilon(\alpha(\alpha(P,x),f(x)))$$

et on peut formuler le théorème

Proposition 2.6.1 *Les propositions sans occurrences de symboles de Skolem sont conséquences d'une proposition si et seulement si elles sont conséquences de sa forme skolémisée.*

On peut donner une intuition de la seconde condition de Miller en remarquant qu'on peut étendre l'ascension des variables aux termes qui vérifient cette condition mais pas aux autres.

Pour traduire un terme de la forme $\lambda x_1\ ...\ \lambda x_n\ t$, où t n'est pas une abstraction, on commence par traduire le terme t. On obtient un terme t' qui contient outre les variables $x_1,...,x_n$ des variables $y_1,...,y_p$ et des combinateurs $c_1,...,c_q$ et des sous-termes u_1, ..., u_r de la forme $(f^k\ v_1\ ...\ v_k)$ *dans lesquels les variables* $x_1,...,x_n$ *n'apparaissent pas.* On remplace ces combinateurs par des variables $z_1,...,z_q$, et ces termes par des variables $z'_1,...,z'_r$, ce qui donne un terme t''. On définit ensuite la traduction du terme $\lambda x_1\ ...\ \lambda x_n\ t$ comme $((y_1,...,y_p,z_1,...,z_q,z'_1,...,z'_r,x_1,...,x_n \longmapsto t'')\ y_1\ ...\ y_p\ c_1\ ...\ c_q\ u_1\ ...\ u_r)$.

Naturellement, si les variables $x_1,...,x_n$ apparaissaient dans $u_1,...,u_r$, leur liaison serait perdue.

2.6.3 La skolémisation dans la présentation de la théorie des types utilisant des substitutions explicites

La présentation de la théorie des types utilisant des substitutions explicites étant une théorie du premier ordre, on peut lui appliquer le théorème de Skolem. Ici, le résultat est beaucoup plus intéressant car, comme on le montre dans [i], on retrouve exactement les deux conditions de Miller.

Comme ci-dessus, l'axiome

$$\forall x\ \exists y\ \varepsilon(\alpha(\alpha(P,x),y))$$

se skolémise en introduisant un symbole de fonction f de rang $(\Gamma \vdash T, \Delta \vdash U)$ où $\Gamma \vdash T$ est la sorte de x et $\Delta \vdash U$ la sorte de y et l'axiome

$$\forall x\ \varepsilon(\alpha(\alpha(P,x),f(x)))$$

Ici encore, le symbole f tout seul n'est pas un terme, car c'est un symbole de fonction et non un symbole d'individu. On retrouve donc la première condition de Miller. Mais on retrouve aussi la seconde : si la proposition

$$\forall x\ \exists y\ \varepsilon(\alpha(\alpha(P,x),y))$$

est obtenue par précuisson puis réduction à partir d'un terme de type o du λ-calcul, la variable x a la sorte $\vdash T$ où le contexte est nécessairement vide. L'argument de f doit donc être de sorte $\vdash T$ ce qui lui interdit de contenir des indices de De Bruijn qui désigneraient des lieurs hors du terme lui-même. Le fait que ce contexte soit vide est donc exactement la seconde condition de Miller.

Enfin, ici encore, on dispose de quantificateurs indépendamment des symboles $\dot{\forall}$ et $\dot{\exists}$ de ce langage particulier. Il n'y a donc pas de problème de formulation de la proposition skolémisée

$$\forall x\ \varepsilon(\alpha(\alpha(P, x), f(x)))$$

et on peut formuler le théorème

Proposition 2.6.2 *Les propositions sans occurrences de symboles de Skolem sont conséquences d'une proposition si et seulement si elles sont conséquences de sa forme skolémisée.*

Chapitre 3

La théorie des ensembles

Le second exemple de théorie modulo que nous détaillons est la théorie des ensembles. L'intérêt de cet exemple est dû au fait que c'est un contre-exemple à de nombreuses propriétés des systèmes modulo (terminaison du système de réécriture, normalisation des démonstrations, complétude de la résolution, ...).

Comme le montre la faible longueur de ce chapitre, le travail sur cette théorie modulo est beaucoup moins avancé que celui sur la théorie des types. En particulier, nous n'avons pas encore de langage des termes du premier ordre intentionnellement équivalent au langage avec un lieur.

3.1 La théorie des ensembles de Zermelo

La théorie des ensembles de Zermelo résout le paradoxe de Russell en remplaçant le schéma de compréhension par trois axiomes et un schéma d'axiome : les axiomes de la *paire*, de l'*union*, des *parties* et le *schéma du sous-ensemble* (ou de compréhension restreint). Ces axiomes s'énoncent ainsi

$$\forall x\ \forall y\ \exists z\ \forall w\ (w \in z \Leftrightarrow (w = x \vee w = y))$$

$$\forall x\ \exists y\ \forall w\ (w \in y \Leftrightarrow \exists z\ (w \in z \wedge z \in x))$$

$$\forall x\ \exists y\ \forall w\ (w \in y \Leftrightarrow \forall z\ (z \in w \Rightarrow z \in x))$$

$$\forall x_1 ... \forall x_n\ \forall y\ \exists z\ \forall w\ (w \in z \Leftrightarrow (w \in y \wedge P))$$

où $x_1, ..., x_n$ sont les variables libres de P sauf w.

À ces axiomes s'ajoutent éventuellement l'axiome d'extensionnalité, l'axiome de fondation, et d'autres axiomes d'existence : l'axiome de l'infini, divers axiomes de grands cardinaux, le schéma de remplacement et l'axiome du choix.

Pour obtenir un langage de termes, on peut skolémiser ces axiomes. On obtient alors les axiomes

$$\forall x\ \forall y\ \forall w\ (w \in \{\}(x, y) \Leftrightarrow (w = x \vee w = y))$$

$$\forall x\ \forall w\ (w \in \bigcup(x) \Leftrightarrow \exists z\ (w \in z \wedge z \in x))$$

$$\forall x\ \forall w\ (w \in \mathcal{P}(x) \Leftrightarrow \forall z\ (z \in w \Rightarrow z \in x))$$

$$\forall x_1 ... \forall x_n\ \forall y\ \forall w\ (w \in f_{x_1,...,x_n,w,P}(x_1, ..., x_n, y) \Leftrightarrow (w \in y \wedge P))$$

Ici encore, il est naturel de transformer ces axiomes en règles de réécriture. On aboutit aux règles de la figure 3.1.

$$w \in \{\}(x,y) \to w = x \vee w = y$$
$$w \in \bigcup(x) \to \exists z\ (w \in z \wedge z \in x)$$
$$w \in \mathcal{P}(x) \to \forall z\ (z \in w \Rightarrow z \in x)$$
$$v \in f_{x_1,...,x_n,w,P}(y_1,...,y_n,z) \to v \in z \wedge [y_1/x_1,...,y_n/x_n,v/w]P$$

FIGURE 3.1 – Les règles de réécriture de la théorie des ensembles

Ici encore, le système de réécriture est confluent car il est orthogonal, mais il ne termine pas, un contre-exemple est la proposition de M. Crabbé, qui est une adaptation du paradoxe de Russell. Soit C le terme $\{x \in a \mid \neg x \in x\}$ c'est-à-dire le terme $f_{w,\neg w \in w}(a)$ on a

$$x \in C \to x \in a \wedge \neg x \in x$$

et donc, en notant A la proposition $C \in C$ et B la proposition $C \in a$

$$A \to B \wedge \neg A$$

qui permet de construire la suite de réductions suivante

$$A \to B \wedge \neg A \to B \wedge \neg(B \wedge \neg A) \to ...$$

De ce fait, la décidabilité de la congruence engendrée par les règles de réécriture de la théorie des ensembles ne peut pas s'établir comme une conséquence de la confluence et de la terminaison. Nous laissons ici ouvert le problème de la décidabilité de cette congruence.

3.2 Le langage des termes

Le dernier paragraphe de [a] traite du problème du langage des termes pour la théorie des ensembles. La skolémisation des axiomes de la paire, de la réunion et des parties donne les symboles $\{\}$, $\bigcup$ et $\mathcal{P}$ attendus. Le cas du schéma du sous-ensemble (schéma de compréhension restreint) est plus compliqué. D'une part, comme on ne considère en théorie des ensembles que des ensembles et non des relations, le schéma de compréhension n'est pas trivialement équivalent au schéma de compréhension fermé comme c'était le cas en théorie des types. On doit donc skolémiser le schéma ouvert

$$\forall x_1 ... \forall x_n\ \forall y\ \exists z\ \forall w\ (w \in z \Leftrightarrow (w \in y \wedge P))$$

où $x_1, ..., x_n$ sont les variables libres de P sauf w.

La skolémisation de ce schéma introduit des symboles de fonction $f_{x_1,...,x_n,w,P}$, et en notant $\{x \in z \mid P\}$ le terme $f_{x_1,...,x_n,w,P}(x_1,...,x_n,z)$ on obtient l'axiome

$$\forall x_1\ ...\ \forall x_n\ \forall z\ (w \in \{x \in z \mid P\} \Leftrightarrow (w \in z \wedge P))$$

Comme dans le langage des combinateurs on semble disposer d'un symbole d'abstraction $\{\ |\ \}$, mais avec la restriction que dans le terme $\{x \in A \mid P\}$ la proposition P doit être sans abstractions. On peut introduire certaines abstractions dans P en appliquant le symbole f à des termes qui contiennent des abstractions, mais on n'obtient pas pour autant un symbole d'abstraction général, car, comme en théorie des types, la variable x liée dans $\{x \in A \mid P\}$ ne peut pas être libre dans les abstractions apparaissant dans P.

Par ailleurs ce langage comporte un nombre infini de symboles de fonction produits par la skolémisation du schéma de compréhension, ce qui le rend difficilement utilisable en pratique. Des présentations finies de la théorie des ensembles, comme celle de Von Neumann-Bernays-Gödel [69] seraient donc sans doute utiles ici.

On peut également introduire un symbole d'abstraction général et on obtient alors une présentation de la théorie des ensembles homologue à la présentation de la théorie des types avec le λ-calcul. On montre dans [a] que cette présentation de la théorie des ensembles est équivalente à la présentation avec les symboles de Skolem, mais, ici encore, l'axiome d'extensionnalité est nécessaire pour montrer cette équivalence.

On ne donne pas de présentation au premier ordre de la théorie des ensembles, intentionnellement équivalente à la présentation avec un opérateur lieur général, mais il est probable qu'on puisse construire un tel langage en utilisant des substitutions explicites.

Chapitre 4

L'élimination des coupures

En déduction modulo, comme dans d'autres cadres, l'élimination des coupures s'étudie d'abord pour la logique intuitionniste, puis se généralise ensuite à la logique classique. En déduction modulo, comme en logique du premier ordre, la déduction naturelle intuitionniste est obtenue en supprimant la règle du tiers exclu et le calcul des séquents intuitionniste en restreignant le calcul à des séquents qui ont une conclusion au plus.

4.1 Les coupures en déduction modulo

En déduction naturelle modulo, comme en déduction naturelle du premier ordre, une *coupure* est simplement une règle d'introduction d'un connecteur ou d'un quantificateur suivie d'une règle d'élimination de ce même symbole. Le processus d'élimination des coupures se définit comme en déduction naturelle du premier ordre. En revanche, sa terminaison pose davantage de problèmes. Si on n'impose pas de restriction sur la congruence, il n'est pas difficile d'exhiber une démonstration qui n'a pas de forme normale. Par exemple, avec la règle $A \to \neg A$, la démonstration

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash_\equiv \neg A}\text{axiome} \quad \dfrac{}{A \vdash_\equiv A}\text{axiome}}{A \vdash_\equiv \bot}\neg\text{-élim}}{\vdash_\equiv \neg A}\neg\textbf{-intro} \qquad \dfrac{\dfrac{\dfrac{}{A \vdash_\equiv \neg A}\text{axiome} \quad \dfrac{}{A \vdash_\equiv A}\text{axiome}}{A \vdash_\equiv \bot}\neg\text{-élim}}{\vdash_\equiv A}\neg\text{-intro}}{\vdash_\equiv \bot}\neg\textbf{-élim}$$

contient une coupure unique, et l'élimination de cette coupure donne la même démonstration. Elle n'a donc pas de forme normale.

Un cas similaire, mais plus intéressant utilise la règle $A \to B \wedge \neg A$. La démonstration

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{BA \vdash_\equiv B \wedge \neg A}\text{axiome}}{BA \vdash_\equiv \neg A}\wedge\text{-élim} \quad \dfrac{}{BA \vdash_\equiv A}\text{axiome}}{BA \vdash_\equiv \bot}\neg\text{-élim}}{B \vdash_\equiv \neg A}\neg\textbf{-intro} \qquad \dfrac{\dfrac{}{B \vdash_\equiv B}\text{axiome} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{BA \vdash_\equiv B \wedge \neg A}\text{axiome}}{BA \vdash_\equiv \neg A}\wedge\text{-élim} \quad \dfrac{}{BA \vdash_\equiv A}\text{axiome}}{BA \vdash_\equiv \bot}\neg\text{-élim}}{B \vdash_\equiv \neg A}\neg\text{-intro}}{B \vdash_\equiv A}\wedge\text{-intro}}{B \vdash_\equiv \bot}\neg\textbf{-élim}$$

réduit successivement sur

$$\dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash_{\equiv} B}\text{axiome} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{BA \vdash_{\equiv} B \wedge \neg A}\text{axiome}}{BA \vdash_{\equiv} \neg A}\wedge\text{-élim} \quad \dfrac{}{BA \vdash_{\equiv} A}\text{axiome}}{BA \vdash_{\equiv} \bot}\neg\text{-élim}}{B \vdash_{\equiv} \neg A}\neg\text{-intro}}{B \vdash_{\equiv} B \wedge \neg A}\wedge\textbf{-intro}}{B \vdash_{\equiv} \neg A}\wedge\textbf{-élim} \quad \dfrac{\dfrac{}{B \vdash_{\equiv} B}\text{axiome} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{BA \vdash_{\equiv} B \wedge \neg A}\text{axiome}}{BA \vdash_{\equiv} \neg A}\wedge\text{-élim} \quad \dfrac{}{BA \vdash_{\equiv} A}\text{axiome}}{BA \vdash_{\equiv} \bot}\neg\text{-élim}}{B \vdash_{\equiv} \neg A}\neg\text{-intro}}{B \vdash_{\equiv} A}\wedge\text{-intro}}{B \vdash_{\equiv} \bot}\neg\text{-élim}$$

puis sur

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{BA \vdash_{\equiv} B \wedge \neg A}\text{axiome}}{BA \vdash_{\equiv} \neg A}\wedge\text{-élim} \quad \dfrac{}{BA \vdash_{\equiv} A}\text{axiome}}{BA \vdash_{\equiv} \bot}\neg\text{-élim}}{B \vdash_{\equiv} \neg A}\neg\textbf{-intro} \quad \dfrac{\dfrac{}{B \vdash_{\equiv} B}\text{axiome} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{BA \vdash_{\equiv} B \wedge \neg A}\text{axiome}}{BA \vdash_{\equiv} \neg A}\wedge\text{-élim} \quad \dfrac{}{BA \vdash_{\equiv} A}\text{axiome}}{BA \vdash_{\equiv} \bot}\neg\text{-élim}}{B \vdash_{\equiv} \neg A}\neg\text{-intro}}{B \vdash_{\equiv} A}\wedge\text{-intro}}{B \vdash_{\equiv} \bot}\neg\textbf{-élim}$$

qui est la démonstration initiale.

L'intérêt de ce contre-exemple est que, comme on l'a vu, en théorie des ensembles, la proposition de Crabbé A se réécrit sur une proposition de la forme $B \wedge \neg A$. La théorie des ensembles ne vérifie donc pas la propriété d'élimination des coupures. Ce contre-exemple dû à Crabbé est discuté par L. Hallnäs [42] et J. Ekman [28] dans une présentation de la théorie des ensembles avec des axiomes de conversion.

On conjecture dans [h] que l'élimination des coupures termine modulo toutes les congruences définies par un système de réécriture confluent et qui termine. On montre certains cas particuliers de cette conjecture.

4.2 La notation fonctionnelle des démonstrations

Pour cela, on utilise une notation fonctionnelle pour les démonstrations en suivant la sémantique de Brouwer-Heyting-Kolmogorov et l'isomorphisme de Curry-De Bruijn-Howard. On propose donc un λ-calcul typé dont les types sont des propositions de la déduction modulo. Ici, notre objectif est simplement de disposer d'une notation pour les démonstrations de déduction modulo, pas de proposer une théorie dans laquelle les démonstrations sont des objets (ce sera l'objet du paragraphe 4.6). Autrement dit, les termes de la théorie considérée et les termes de démonstration appartiennent à deux langages distincts. On utilisera des lettres latines $x, y, ...$ pour les variables d'objet et des lettres grecques $(\alpha, \beta, ...)$ pour les variables de démonstrations.

Les termes de ce λ-calcul sont

$$\begin{array}{rl} \pi ::= & \alpha \\ & \mid \lambda\alpha\ \pi \mid (\pi\ \pi') \\ & \mid (\pi, \pi') \mid \mathit{fst}(\pi) \mid \mathit{snd}(\pi) \\ & \mid i(\pi) \mid j(\pi) \mid (\delta\ \pi\ \alpha\pi'\ \beta\pi'') \\ & \mid (\mathit{bot\acute{e}lim}\ \pi) \\ & \mid \lambda x\ \pi \mid (\pi\ t) \\ & \mid (t, \pi) \mid (\mathit{ex\acute{e}lim}\ \pi\ x\alpha\pi') \end{array}$$

La figure 4.1 donne les règles de typage de ce calcul.

Il est trivial qu'une proposition est un type habité de ce calcul si et seulement si elle est démontrable en déduction naturelle modulo intuitionniste.

La démonstration de Crabbé ci-dessus, par exemple, s'exprime par le terme

$$\lambda\alpha\ (\mathit{snd}(\alpha)\ \alpha)\ (\beta, \lambda\alpha\ (\mathit{snd}(\alpha)\ \alpha))$$

On définit ensuite les règles de réduction des démonstrations qui correspondent à l'élimination des coupures.

$$\frac{}{\Gamma \vdash_{\equiv} \alpha : B} \text{ axiome si } \alpha : A \in \Gamma,\ A \equiv B$$

$$\frac{\Gamma, \alpha : A \vdash_{\equiv} \pi : B}{\Gamma \vdash_{\equiv} \lambda\alpha\ \pi : C} \Rightarrow\text{-intro si } C \equiv (A \Rightarrow B)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : C \quad \Gamma \vdash_{\equiv} \pi' : A}{\Gamma \vdash_{\equiv} (\pi\ \pi') : B} \Rightarrow\text{-élim si } C \equiv (A \Rightarrow B)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : A \quad \Gamma \vdash_{\equiv} \pi' : B}{\Gamma \vdash_{\equiv} (\pi, \pi') : C} \wedge\text{-intro si } C \equiv (A \wedge B)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : C}{\Gamma \vdash_{\equiv} \mathit{fst}(\pi) : A} \wedge\text{-élim si } C \equiv (A \wedge B)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : C}{\Gamma \vdash_{\equiv} \mathit{snd}(\pi) : B} \wedge\text{-élim si } C \equiv (A \wedge B)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : A}{\Gamma \vdash_{\equiv} i(\pi) : C} \vee\text{-intro si } C \equiv (A \vee B)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : B}{\Gamma \vdash_{\equiv} j(\pi) : C} \vee\text{-intro si } C \equiv (A \vee B)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : D \quad \Gamma, \alpha : A \vdash_{\equiv} \pi' : C \quad \Gamma, \beta : B \vdash_{\equiv} \pi'' : C}{\Gamma \vdash_{\equiv} (\delta\ \pi\ \alpha\pi'\ \beta\pi'') : C} \vee\text{-élim si } D \equiv (A \vee B)$$

$$\frac{\Gamma, \alpha : A \vdash_{\equiv} \pi : \bot}{\Gamma \vdash_{\equiv} \lambda\alpha\ \pi : B} \neg\text{-intro si } B \equiv (\neg A)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : B \quad \Gamma \vdash_{\equiv} \pi' : A}{\Gamma \vdash_{\equiv} (\pi\ \pi') : \bot} \neg\text{-élim si } B \equiv (\neg A)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : B}{\Gamma \vdash_{\equiv} (\mathit{botélim}\ \pi) : A} \bot\text{-élim si } B \equiv \bot$$

$$\frac{\Gamma \vdash_{\equiv} \pi : A}{\Gamma \vdash_{\equiv} \lambda x\ \pi : B} (x, A)\ \forall\text{-intro si } B \equiv (\forall x\ A),\ x \notin FV(\Gamma)$$

$$\frac{\Gamma \vdash_{\equiv} \pi : B}{\Gamma \vdash_{\equiv} (\pi\ t) : C} (x, A, t)\ \forall\text{-élim si } B \equiv (\forall x\ A),\ C \equiv [t/x]A$$

$$\frac{\Gamma \vdash_{\equiv} \pi : C}{\Gamma \vdash_{\equiv} (t, \pi) : B} (x, A, t)\ \exists\text{-intro si } B \equiv (\exists x\ A),\ C \equiv [t/x]A$$

$$\frac{\Gamma \vdash_{\equiv} \pi : C \quad \Gamma, \alpha : A \vdash_{\equiv} \pi' : B}{\Gamma \vdash_{\equiv} (\mathit{exélim}\ \pi\ x\alpha\pi') : B} (x, A)\ \exists\text{-élim si } C \equiv (\exists x\ A),\ x \notin FV(\Gamma B)$$

FIGURE 4.1 – Les règles de typage des démonstrations

Définition 4.2.1

$$(\lambda\alpha\ \pi_1\ \pi_2) \rightarrow [\pi_2/\alpha]\pi_1$$

$$fst(\pi_1, \pi_2) \rightarrow \pi_1$$

$$snd(\pi_1, \pi_2) \rightarrow \pi_2$$

$$\delta(i(\pi_1), \alpha\pi_2, \beta\pi_3) \rightarrow [\pi_1/\alpha]\pi_2$$

$$\delta(j(\pi_1), \alpha\pi_2, \beta\pi_3) \rightarrow [\pi_1/\beta]\pi_3$$

$$(\lambda x\ \pi\ t) \rightarrow [t/x]\pi$$

$$(\mathit{exélim}\ (t, \pi_1)\ \alpha x\pi_2) \rightarrow [t/x, \pi_1/\alpha]\pi_2$$

On veut maintenant montrer la terminaison du processus d'élimination des coupures, c'est-à-dire la normalisation de ce calcul.

4.3 La réductibilité

La première tentative pour démontrer la normalisation des démonstrations, au moins modulo certaines congruences, est de s'inspirer de la démonstration de la normalisation des démonstrations en logique du premier ordre et d'utiliser la méthode de Tait.

4.3.1 L'élimination des coupures en logique du premier ordre

En logique du premier ordre, on ne peut pas montrer que toutes les démonstrations sont normalisables par une simple récurrence sur la structure des démonstrations. En effet, le fait que π et π' soient normalisables ne permet pas de déduire que $(\pi\ \pi')$ l'est, car ce terme peut être lui-même un radical.

On renforce donc l'hypothèse de récurrence en montrant par récurrence que toutes les démonstrations sont *réductibles*. Une démonstration réductible de la proposition $A \Rightarrow B$ est une démonstration fortement normalisable, telle que quand elle se réduit sur une démonstration de la forme $\lambda\alpha\ \pi_1$ alors, pour toute démonstration réductible π' de A, $[\pi'/\alpha]\pi_1$ est une démonstration réductible de B. Le cas où le terme est un radical est ainsi anticipé. On définit de même l'ensemble des démonstrations réductibles d'une proposition de la forme $A \wedge B$, $A \vee B$, $\neg A$, $\bot$, $\forall x\ P$, $\exists x\ P$. Les démonstrations réductibles d'une proposition atomique sont, quant à elles, simplement les démonstrations fortement normalisables. L'ensemble des démonstrations réductibles d'une proposition P se construit donc par récurrence sur la taille de P. En utilisant le vocabulaire du λ-calcul, l'ensemble des termes réductibles de type T se construit par récurrence sur la taille de T.

On montre ensuite, par récurrence sur la structure des démonstrations, que toute démonstration d'une proposition A est réductible et donc fortement normalisable.

Pour cela, on note $\mathcal{R}_A$ l'ensemble des démonstrations réductibles de A et on commence par montrer les lemmes suivants.

- si $\pi \in \mathcal{R}_A$, alors π est fortement normalisable,
- si $\pi \in \mathcal{R}_A$ et $\pi \rightarrow \pi'$ alors $\pi' \in \mathcal{R}_A$,
- si π est neutre (c'est-à-dire une variable ou une élimination) et pour tout π' telle que $\pi \rightarrow^1 \pi'$, $\pi' \in \mathcal{R}_A$ alors $\pi \in \mathcal{R}_A$.

Après [36, 37, 38, 59, 34], on appelle *candidat de réductibilité* un ensemble qui vérifie ces conditions.

4.3.2 L'élimination des coupures en déduction modulo

En déduction modulo, si deux propositions P et Q sont congruentes, toute démonstration de l'une est une démonstration de l'autre. De ce fait, on doit avoir $\mathcal{R}_P = \mathcal{R}_Q$. On ne peut donc plus définir l'ensemble des démonstrations réductibles d'une proposition atomique A comme l'ensemble de toutes les démonstrations fortement normalisables, car cette proposition peut se réduire, par exemple, sur une proposition de la forme $B \Rightarrow C$ et dans ce cas, pour être réductible, une démonstration doit, en outre, vérifier la propriété que si elle se réduit sur une démonstration de la forme $\lambda\alpha\ \pi_1$, alors pour toute démonstration réductible π', $[\pi'/\alpha]\pi_1$ est réductible. De ce fait, la construction de la famille $\mathcal{R}_P$ ne peut plus se faire simplement par récurrence sur la taille de P.

Une tentative consiste à définir les ensembles $\mathcal{R}_P$ par récurrence sur la taille de P pour P normal, puis de définir l'ensemble $\mathcal{R}_P$ pour P quelconque comme $\mathcal{R}_{P\downarrow}$ où $P\downarrow$ est la forme normale de P. Hélas, cette tentative échoue, car cela nous amène à définir les démonstrations réductibles de la proposition normale $\forall x\ A$ comme l'ensemble des démonstrations π de $\forall x\ A$ qui vérifient les propriétés suivantes

- π est fortement normalisable,
- si π se réduit sur une démonstration de la forme $\lambda x\ \pi_1$ alors pour tout terme t, $[t/x]\pi_1$ est une démonstration réductible de $[t/x]A$.

Cette définition est incorrecte, car la proposition $[t/x]A$ peut très bien ne pas être normale. On peut alors tenter de remplacer cette seconde condition par

- si π se réduit sur une démonstration de la forme $\lambda x\ \pi_1$ alors pour terme t, $[t/x]\pi_1$ est une démonstration réductible de $([t/x]A)\downarrow$.

où $([t/x]A)\downarrow$ est la forme normale de la proposition $[t/x]A$. Cette tentative échoue également, car la proposition $([t/x]A)\downarrow$ n'est pas nécessairement plus petite que la proposition $\forall x\ P$.

Cette remarque n'est pas nouvelle : en théorie des types simples la substitution d'une variable de prédicat modifie, similairement, la complexité des propositions. L'élimination des coupures en théorie des types est d'ailleurs un corollaire de cette conjecture. Le contre-exemple traditionnel [36, 37, 38, 59, 34] qui montre qu'on ne peut pas donner une telle définition de la réductibilité en théorie des types, s'adapte ici. On serait amené à poser qu'une démonstration π de la proposition $\forall x\ \varepsilon(x)$ est réductible si elle est fortement normalisable et si quand elle se réduit sur une démonstration de la forme $\lambda x\ \pi_1$, alors pour tout terme t, $[t/x]\pi_1$ est une démonstration réductible de $\varepsilon(t)\downarrow$. Cela demande d'avoir au préalable défini l'ensemble des démonstrations réductibles pour toutes les propositions $\varepsilon(t)\downarrow$ en particulier pour $\varepsilon(\dot\forall\ (\lambda 1))\downarrow = \forall x\ \varepsilon(x)$.

Cependant, cet argument est correct pour tous les systèmes de réécriture qui ne comportent pas de quantificateurs dans leurs membres droits. Dans ce cas, la définition ci-dessus est une définition correcte par récurrence pour l'ordre lexicographique sur le couple $(q(A), c(A))$ où $q(A)$ est le nombre de quantificateurs dans la proposition A et c le nombre de connecteurs.

On en déduit donc la proposition suivante.

Proposition 4.3.1 *Les démonstrations modulo une congruence définie par un système de réécriture qui est confluent, qui termine et qui n'a pas d'occurrences de quantificateurs dans les membres droits de ses règles sont fortement normalisables.*

4.3.3 Les prémodèles

Dans les autres cas, montrer l'élimination des coupures demande de construire une famille d'ensembles $\mathcal{R}_A$ indexée par les propositions qui vérifie les conditions suivantes.

- Les ensembles $\mathcal{R}_A$ sont des candidats de réductibilité.
- Conditions sur les propositions composées :
  - $\mathcal{R}_{A \Rightarrow B}$ est l'ensemble des démonstrations π telles que π est fortement normalisable et si π se réduit sur une démonstration de la forme $\lambda\alpha\ \pi_1$, alors pour toute démonstration π' de $\mathcal{R}_A$, $[\pi'/\alpha]\pi_1$ est un élément de $\mathcal{R}_B$.

- $\mathcal{R}_{A \wedge B}$ est l'ensemble des démonstrations π telles que π est fortement normalisable et si π se réduit sur une démonstration de la forme (π_1, π_2), alors π_1 est un élément de $\mathcal{R}_A$ et π_2 est un élément de $\mathcal{R}_B$.
- $\mathcal{R}_{A \vee B}$ est l'ensemble des démonstrations π telles que π est fortement normalisable et si π se réduit sur une démonstration de la forme $i(\pi_1)$ (resp. $j(\pi_2)$) alors π_1 est un élément de $\mathcal{R}_A$ (resp. π_2 est un élément de $\mathcal{R}_B$).
- $\mathcal{R}_{\neg A}$ est l'ensemble des démonstrations π telles que π est fortement normalisable et si π se réduit sur une démonstration de la forme $\lambda\alpha\pi_1$, alors pour toute démonstration π' de $\mathcal{R}_A$, $[\pi'/\alpha]\pi_1$ est fortement normalisable.
- $\mathcal{R}_{\perp}$ est l'ensemble des démonstrations fortement normalisables.
- $\mathcal{R}_{\forall x\ A}$ est l'ensemble des démonstrations π telles que π est fortement normalisable et si π se réduit sur une démonstration de la forme $\lambda x\ \pi_1$ alors pour tout terme t $[t/x]\pi_1$ est un élément de $\mathcal{R}_{[t/x]A}$.
- $\mathcal{R}_{\exists x\ A}$ est l'ensemble des démonstrations π telles que π est fortement normalisable et si π se réduit sur une démonstration de la forme (t, π_1) alors $[t/x]\pi_1$ est un élément de $\mathcal{R}_{[t/x]A}$.

- Si $P \equiv Q$ alors $\mathcal{R}_P = \mathcal{R}_Q$.

Dans cette définition, nous avons fait le choix de ne pas normaliser les propositions dans l'expression des conditions sur les ensembles $\mathcal{R}_{\forall x\ P}$ et $\mathcal{R}_{\exists x\ P}$. Cela nous impose d'ajouter la dernière condition : les propositions congruentes doivent avoir le même ensemble de démonstrations réductibles.

L'avantage de ce choix est que la famille des ensembles $\mathcal{R}_A$ est entièrement définie par les ensembles $\mathcal{R}_A$ où A est atomique. Autrement dit, pour définir une famille $\mathcal{R}_A$, il suffit de se donner les ensembles $\mathcal{R}_{P(t_1,...,t_n)}$, c'est-à-dire pour chaque symbole de prédicat P une fonction $\hat{P}$ qui à chaque n-uplet de termes associe un candidat de réductibilité. On voit apparaître ici une similarité avec la notion de modèle. Pour se donner un modèle, il faut également, se donner pour chaque symbole de prédicat P une fonction $\hat{P}$ qui à chaque n-uplet de termes associe une valeur de vérité. La comparaison peut se poursuivre. Si deux termes t_1 et t'_1 sont congruents, les ensembles $\hat{P}(t_1, ..., t_n)$ et $\hat{P}(t'_1, ..., t_n)$ doivent être identiques. La fonction $\hat{P}$ peut donc se définir comme une fonction d'un objet abstrait (par exemple, la classe de t_1 et t'_1) que t_1 et t'_1 dénotent. Enfin, la condition selon laquelle deux propositions congruentes doivent avoir le même ensemble de démonstrations réductibles est la même que celle que nous avons vue dans la définition des modèles d'une théorie modulo au paragraphe 1.4.

Dans [h], nous avons donc utilisé le vocabulaire de la théorie des modèles : la donnée d'un ensemble $\mathcal{M}$ (ou d'une famille d'ensembles dans le cas multisorté) de fonctions $\hat{f}$ et $\hat{P}$ est appelée *prémodèle*, et l'ensemble $\mathcal{R}_A$ est appelé *dénotation* de A. La définition d'un prémodèle est la même que celle d'un modèle, sauf que les valeurs de vérité sont remplacées par des candidats de réductibilité et la dénotation des propositions composées est définie comme ci-dessus. Un prémodèle et donc un modèle multivalué dont les valeurs de vérités sont des candidats de réductibilité. Un prémodèle est un prémodèle d'un système de réécriture si deux propositions congruentes ont même dénotation.

Dans [h] on a montré la proposition suivante

Proposition 4.3.2 *Si une congruence a un prémodèle, alors les démonstrations en déduction naturelle modulo cette congruence normalisent fortement.*

On montre également que des variantes de la notion de prémodèle permettent d'obtenir des résultats similaires pour le calcul des séquents intuitionniste et classique. Dans ce cas, il n'y a pas ici de processus d'élimination des coupures dont on montre la terminaison, puisqu'en calcul des séquents, le processus d'élimination d'une coupure n'est pas local. On montre donc que si une congruence a un prémodèle (dans un sens légèrement différent) alors la règle de coupure est redondante dans le calcul des séquents modulo cette congruence.

On montre facilement que pour les systèmes de réécriture qui ne comportent pas de quantificateurs dans leurs membres droits, on peut construire un prémodèle très simple qui rappelle, par certains aspects, le modèle syntaxique construit dans le théorème de complétude.

On montre aussi dans [h] que la démonstration de normalisation pour la théorie des types exprimée avec des combinateurs peut s'exprimer comme la construction d'un prémodèle. Cette construction de prémodèle

ressemble par bien des aspects à la construction du modèle fonctionnel de la théorie des types : dans un cas comme dans l'autre l'ensemble $\mathcal{M}_\iota$ est un singleton et l'ensemble $\mathcal{M}_{T \to U}$ est l'ensemble des fonctions de $\mathcal{M}_T$ dans $\mathcal{M}_U$. On retrouve aussi plusieurs aspects de la démonstration traditionnelle de normalisation des démonstrations en exprimant ces démonstrations dans le λ-calcul polymorphe avec constructeurs de types F_ω [36, 37] : le fait que tous les termes d'un même type construit sans le type o dénotent le même objet correspond à l'élimination des termes du premier ordre dans la démonstration traditionnelle, et la fonctionnalité du modèle fait qu'un symbole de type $o \to o$, par exemple, est interprété comme une fonction des candidats dans les candidats. Dans [i] on montre la normalisation des démonstrations dans la formulation de la théorie des types utilisant des substitutions explicites, encore une fois en construisant un prémodèle.

On montre enfin dans [h] que si un système de réécriture n'a pas d'occurrence négative de proposition atomique dans les membres droits de ses règles, alors on peut construire un prémodèle comme plus petit point fixe.

4.3.4 La conjecture de normalisation

Notre conjecture est que pour tous les systèmes de réécriture confluents et qui terminent, on peut construire un prémodèle et donc les démonstrations normalisent fortement modulo ce système de réécriture. Nous croyons cette conjecture vraie car nous n'avons pas réussi à trouver de contre-exemple.

Même si cette conjecture est vraie, on peut s'interroger sur les outils nécessaires pour la démontrer. Puisqu'elle implique la cohérence de la théorie des types, on sait par le second théorème d'incomplétude de Gödel, qu'on ne peut pas la démontrer dans la théorie des types elle-même. Une question importante, du point de vue méthodologique, est de savoir si cette conjecture implique la cohérence de la théorie des ensembles ou non. Le système de réécriture que nous avons proposé pour la théorie des ensembles ne termine pas, mais on peut le modifier ainsi :

$$w \in \{\}(x, y) \to \forall a \ (w = a \Rightarrow (a = x \vee a = y))$$

$$w \in \bigcup(x) \to \exists z \ (w \in z \wedge z \in x)$$

$$w \in \mathcal{P}(x) \to \forall z \ (z \in w \Rightarrow z \in x)$$

$$v \in f_{x_1,...,x_n,w,P}(y_1, ..., y_n, z)$$
$$\to \forall a \ \forall b_1 \ ... \ \forall b_n \ (a = v \wedge b_1 = y_1 \wedge ... \wedge b_n = y_n) \Rightarrow a \in z \wedge [b_1/x_1, ..., b_n/x_n, a/w]P$$

On peut montrer que ce système termine. Hélas, l'élimination des coupures modulo ce système n'implique pas trivialement la cohérence de la théorie des ensembles car, dans ce système, les axiomes de l'égalité sont nécessaires pour exprimer les démonstrations de la théorie des ensembles. Mais, on peut imaginer qu'il y a là une piste pour montrer que notre conjecture implique la cohérence de la théorie des ensembles et donc qu'elle ne peut se démontrer sans faire appel à des axiomes au delà de la théorie des ensembles, comme par exemple, des axiomes de grands cardinaux.

Plus modestement, on peut voir que le système de réécriture de la figure 3.1 termine pour de grands fragments de la théorie des ensembles (par exemple, vraisemblablement, pour le fragment où on restreint le schéma du sous-ensemble aux propositions stratifiables au sens de W.V.O. Quine) et donc que pour démontrer cette conjecture, il est nécessaire de sortir de ce fragment.

4.3.5 Des prémodèles aux modèles

Si une théorie $(\equiv, \Gamma)$ a un prémodèle, alors le calcul des séquents classiques modulo la congruence $\equiv$ a la propriété d'élimination des coupures. Si, en outre, le candidat de réductibilité dénotation de $\bot$ ne contient pas de démonstration close (c'est, par exemple, toujours le cas quand Γ est vide car il n'y a pas de démonstration sans coupure et sans axiome de la proposition $\bot$), alors la théorie est cohérente et, d'après le théorème de complétude de Gödel, elle a un modèle.

On peut s'interroger sur les relations entre les prémodèles et les modèles de cette théorie, en particulier on peut vouloir transformer les prémodèles en modèles (ce qui justifierait notre terminologie de "prémodèle").

Un prémodèle est un modèle multivalué dont les valeurs de vérité sont des candidats de réductibilité. Une piste est donc de considérer un candidat qui contient une démonstration close comme une valeur de vérité positive et un candidat qui ne contient que des démonstrations ouvertes comme une valeur de vérité négative. Hélas cette idée est trop naïve car une proposition indéterminée de la théorie dénoterait une valeur de vérité négative, ainsi que sa négation. Il faut donc vraisemblablement, comme dans la démonstration du théorème de complétude, commencer par compléter et saturer le prémodèle.

Nous laissons ce projet pour un travail futur.

4.4 Les coupures de conversion

Nous l'avons vu, une théorie en déduction modulo est toujours équivalente à une théorie du premier ordre. La notion de coupure modulo se traduit donc en logique du premier ordre en une notion de coupure relative à certains axiomes (comme les notions de coupure en logique du premier ordre avec égalité, ou dans l'arithmétique sont relatives aux axiomes de l'égalité ou à l'axiome de récurrence).

On a caractérisé dans [e] ces coupures que nous avons appelées *coupures de conversion*. Une coupure de conversion est une séquence formée d'une règle d'introduction et d'une règle d'élimination séparées par une *étape de conversion*, c'est-à-dire une séquence d'utilisation des axiomes équationnels ou d'équivalence de la théorie.

Cela permet, par exemple, de formuler le théorème d'élimination des coupures pour la théorie des types en restant intégralement au premier ordre, sans même utiliser la déduction modulo.

4.5 La normalisation des démonstrations du théorème de Cantor

À titre d'exemple, on présente dans ce paragraphe plusieurs démonstrations du théorème de Cantor dans différentes théories modulo et leur forme normale. Le théorème de Cantor montre qu'il n'y a pas de surjection d'un ensemble dans l'ensemble de ses parties.

4.5.1 En théorie des types avec une fonction

En théorie des types, un ensemble est un objet de type $T \to o$. Ici, nous faisons le choix de nous intéresser uniquement à l'ensemble de tous les objets de type ι. L'ensemble des parties de cet ensemble est l'ensemble de tous les objets de type $\iota \to o$. Cela revient à montrer qu'il n'y a pas de surjection du type ι dans le type $\iota \to o$.

La première possibilité est de représenter cette surjection potentielle par une fonction f de type $\iota \to \iota \to o$. La surjectivité de cette fonction peut s'exprimer par l'existence d'une fonction g inverse à droite de f, c'est-à-dire d'une fonction de type $(\iota \to o) \to \iota$ telle que pour tout x, $(f\ (g\ x)) = x$. En utilisant la définition de G.W. Leibniz de l'égalité, cette hypothèse se traduit par la proposition

$$H : \forall x\ \forall p\ (\varepsilon(p\ (f\ (g\ x))) \Leftrightarrow \varepsilon(p\ x))$$

On considère alors l'ensemble $\lambda x\ \dot\neg((f\ x)\ x)$ des objets x de type ι qui n'appartiennent pas à $(f\ x)$.

$$C = \lambda\dot\neg[\uparrow](f[\uparrow]\ 1\ 1)$$

L'ensemble C est l'image par f de l'objet $(g\ C)$ de type ι. Par définition de C, $(g\ C)$ appartient à C si et seulement si il n'appartient pas à $(f\ (g\ C))$, mais cet ensemble n'est autre que C lui-même donc $(g\ C)$ appartient à C si et seulement si il ne lui appartient pas, ce qui est contradictoire.

On construit le λ-terme démonstration du théorème de Cantor ainsi. La proposition A "$(g\ C)$ appartient à C" s'écrit

$$\varepsilon(\lambda\dot\neg[\uparrow](f[\uparrow]\ 1\ 1)\ (g\ \lambda\dot\neg[\uparrow](f[\uparrow]\ 1\ 1)))$$

Sa forme normale est

$$\neg\varepsilon((f\ (g\ \lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1))\ (g\ \lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1))))$$

En appliquant l'hypothèse H au terme C et au prédicat $P = \lambda(1\ (g[\uparrow]\ \lambda\dot{\neg}[\uparrow^2](f[\uparrow^2]\ 1\ 1)))$ on obtient une démonstration de $(H\ C\ P)$ de la proposition

$$\varepsilon(f\ (g\ \lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1))\ (g\ \lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1))) \Leftrightarrow \varepsilon(\lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1)\ (g\ \lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1)))$$

qui se réduit sur

$$\varepsilon(f\ (g\ \lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1))\ (g\ \lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1))) \Leftrightarrow \neg\varepsilon(f\ (g\ \lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1))\ (g\ \lambda\dot{\neg}[\uparrow](f[\uparrow]\ 1\ 1)))$$

c'est-à-dire sur $A \Leftrightarrow \neg A$. De cette proposition on peut déduire une démonstration de la proposition $\neg A$

$$\lambda\alpha\ (fst(H\ C\ P)\ \alpha\ \alpha)$$

puis une démonstration de $\perp$

$$\lambda\alpha\ (fst(H\ C\ P)\ \alpha\ \alpha)\ (snd(H\ C\ P)\ \lambda\alpha\ (fst(H\ C\ P)\ \alpha\ \alpha))$$

Comme toute démonstration en théorie des types, cette démonstration est fortement normalisable, sa forme normale est

$$(fst(H\ C\ P)\ (snd(H\ C\ P)\ \lambda\alpha\ (fst(H\ C\ P)\ \alpha\ \alpha))\ (snd(H\ C\ P)\ \lambda\alpha\ (fst(H\ C\ P)\ \alpha\ \alpha)))$$

À partir de cette démonstration de $\perp$ sous l'hypothèse H, on peut facilement construire une démonstration de la proposition

$$\neg\exists f\ \exists g\ \forall x\ \forall p\ (\varepsilon(p\ (f\ (g\ x))) \Leftrightarrow \varepsilon(p\ x))$$

4.5.2 En théorie des types avec une relation

Au lieu d'utiliser la notion primitive de fonction de la théorie des types, on peut coder les fonctions comme des relations fonctionnelles. On pose alors un symbole R de type $\iota \to (\iota \to o) \to o$ et deux hypothèses qui expriment que R est surjective

$$E : \forall y\ \exists x\ \varepsilon(R\ x\ y)$$

et fonctionnelle

$$\forall x\ \forall y\ \forall z\ (\varepsilon(R\ x\ y) \Rightarrow \varepsilon(R\ x\ z) \Rightarrow y = z)$$

Encore une fois, on utilise la définition de Leibniz de l'égalité

$$F : \forall x\ \forall y\ \forall z\ (\varepsilon(R\ x\ y) \Rightarrow \varepsilon(R\ x\ z) \Rightarrow \forall p\ (\varepsilon(p\ y) \Leftrightarrow \varepsilon(p\ z)))$$

Pour dire que x n'appartient pas à son image, on est obligé de dire qu'il n'appartient à aucune de ses images car on ne dispose plus d'un terme pour désigner cette image. L'ensemble C est donc $\lambda x\ \dot{\forall}\lambda y\ ((R\ x\ y)\dot{\Rightarrow}\dot{\neg}(y\ x))$, c'est-à-dire

$$C = \lambda\dot{\forall}\lambda((R\ 2\ 1)\dot{\Rightarrow}\dot{\neg}(1\ 2))$$

On commence par démontrer $\perp$ sous l'hypothèse $h : (R\ a\ C)$. Soit A la proposition $\varepsilon(C\ a)$, cette proposition se réduit sur

$$\forall y\ (\varepsilon(R\ a\ y) \Rightarrow \neg\varepsilon(y\ a))$$

Le terme

$$\pi = \lambda\alpha\ (\alpha\ C\ h)$$

est une démonstration de $A \Rightarrow \neg A$ et le terme

$$\pi' = \lambda\alpha\ \lambda y\ \lambda\beta\ (fst(F\ a\ C\ y\ h\ \beta\ (\lambda\dot{\neg}[\uparrow](1\ a[\uparrow])))\ \alpha)$$

une démonstration de $\neg A \Rightarrow A$. On peut comme ci-dessus, en déduire une démonstration de $\bot$

$$(\pi\ (\pi'\ (\lambda\gamma\ (\pi\ \gamma\ \gamma)))\ (\pi'\ (\lambda\gamma\ (\pi\ \gamma\ \gamma))))$$

Comme toute démonstration en théorie des types, cette démonstration a une forme normale

$$\pi_0 = (fst(F\ a\ C\ C\ h\ h\ (\lambda\dot{\neg}[\uparrow](1\ a[\uparrow])))\ (\lambda\gamma\ (\gamma\ C\ h\ \gamma))))(\lambda y\ \lambda\beta\ (fst(F\ a\ C\ y\ h\ \beta\ (\lambda\dot{\neg}[\uparrow](1\ a[\uparrow])))\ (\lambda\gamma\ (\gamma\ C\ h\ \gamma))))$$

On peut ensuite se passer de l'hypothèse h pour obtenir une démonstration normale de $\bot$ sous les hypothèses E et F

$$(ex\acute{e}lim\ (E\ C)\ ah\pi_0)$$

À partir de cette démonstration de $\bot$ sous les hypothèses E et F, on peut facilement construire une démonstration de la proposition

$$\neg\exists R\ ((\forall y\ \exists x\ \varepsilon(R\ x\ y)) \wedge (\forall x\ \forall y\ \forall z\ (\varepsilon(R\ x\ y) \Rightarrow \varepsilon(R\ x\ z) \Rightarrow \forall p\ (\varepsilon(p\ y) \Leftrightarrow \varepsilon(p\ z)))))$$

4.5.3 En théorie des ensembles

La démonstration du théorème de Cantor en théorie des ensembles ressemble beaucoup à celle en théorie des types avec une relation. La démonstration que nous donnons ici est, cela dit, un peu plus générale puisque nous considérons un ensemble B quelconque, alors que dans la démonstration en théorie des types ci-dessus, nous avions fait le choix de nous restreindre à l'ensembles des objets de type ι.

On note $< x, y >$ l'ensemble $\{\{x\}, \{x, y\}\}$ c'est-à-dire $\{\}(\{\}(x, x), \{\}(x, y))$.

On pose un ensemble R et deux axiomes qui expriment que R est une relation surjective

$$E : \forall y\ (y \in \mathcal{P}(B) \Rightarrow \exists x\ (x \in B \wedge < x, y > \in R))$$

et fonctionnelle

$$F : \forall x\ \forall y\ \forall z\ (< x, y > \in R \Rightarrow < x, z > \in R \Rightarrow y = z)$$

On utilise aussi l'axiome de l'égalité

$$L : \forall z\ \forall x\ \forall y\ (x = y \Rightarrow \neg z \in x \Rightarrow \neg z \in y)$$

Pour dire que x n'appartient pas à son image, on dit qu'il n'appartient à aucune de ses images

$$C = \{x \in B \mid \forall y\ (< x, y > \in R \Rightarrow \neg x \in y)\}$$

On a donc une règle de réécriture

$$x \in C \to x \in B \wedge \forall y\ (< x, y > \in R \Rightarrow \neg x \in y)$$ [1]

1. La skolémisation du schéma du sous-ensemble ne nous donne pas un tel terme puisque la proposition

$$\forall r\ \forall b\ \exists e\ \forall x\ (x \in e \Leftrightarrow ((x \in b) \wedge \forall y\ (< x, y > \in r \Rightarrow \neg x \in y)))$$

qui contient des symboles de Skolem, n'est pas une instance du schéma du sous-ensemble. On peut remplacer la proposition $< x, y > \in r$ par

$$\forall u\ ((\forall v\ (v \in u \Leftrightarrow (\forall w\ (w \in v \Leftrightarrow (w = x \vee w = y)) \vee \forall w\ (w \in v \Leftrightarrow w = x)))) \Rightarrow u \in r)$$

mais on obtient une démonstration beaucoup plus compliquée. En revanche, on a un tel terme C dans le système avec lieurs présenté dans [a]. On aurait également un tel terme dans une présentation au premier ordre de ce système. En attendant, on se place dans un système dans lequel le terme C est une constante et où on pose la règle de réécriture ci-dessus.

On commence par démontrer la proposition $\bot$ sous l'hypothèse $h : a \in B \wedge < a, C > \in R$. Soit A la proposition $a \in C$, cette proposition se réduit sur

$$a \in B \wedge \forall y\ (< a, y > \in R \Rightarrow \neg a \in y)$$

Le terme

$$\pi = \lambda\alpha\ (snd(\alpha)\ C\ snd(h))$$

est une démonstration de $A \Rightarrow \neg A$ et le terme

$$\pi' = \lambda\alpha\ < fst(h), \lambda y\ \lambda\beta\ (L\ a\ C\ y\ (F\ a\ C\ y\ snd(h)\ \beta)\ \alpha) >$$

une démonstration de $\neg A \Rightarrow A$. On peut, comme ci-dessus, en déduire une démonstration de $\bot$

$$(\pi\ (\pi'\ (\lambda\gamma\ (\pi\ \gamma\ \gamma)))\ (\pi'\ (\lambda\gamma\ (\pi\ \gamma\ \gamma))))$$

Cette démonstration a une forme normale

$$\begin{aligned}\pi_0 = (L\ a\ C\ C\ (F\ a\ C\ C\ snd(h)\ snd(h))\ (\lambda\gamma\ (snd(\gamma)\ C\ snd(h)\ \gamma))\\ < fst(h), \lambda y\ \lambda\beta\ (L\ a\ C\ y\ (F\ a\ C\ y\ snd(h)\ \beta)\ (\lambda\gamma\ (snd(\gamma)\ C\ snd(h)\ \gamma))) >)\end{aligned}$$

On peut ensuite se passer de l'hypothèses h pour obtenir une démonstration normale de $\bot$ sous les hypothèses E et F. La proposition $C \in \mathcal{P}(B)$ se réduit sur $\forall x\ ((x \in B \wedge \forall y\ (< x, y > \in R \Rightarrow \neg x \in y)) \Rightarrow x \in B))$. Le terme $\rho = \lambda x\ \lambda\alpha\ fst(\alpha)$ en est donc une démonstration. On construit alors la démonstration suivante de $\bot$ sous les hypothèses E et F

$$(exélim\ (E\ C\ \rho)\ ah\pi_0)$$

À partir de cette démonstration de $\bot$ sous les hypothèses E et F, on peut facilement construire une démonstration de la proposition

$$\forall B\ \neg\exists R\ ((\forall y\ (y \in \mathcal{P}(B) \Rightarrow \exists x\ (x \in B \wedge < x, y > \in R))) \wedge (\forall x\ \forall y\ \forall z\ (< x, y > \in R \Rightarrow < x, z > \in R \Rightarrow y = z)$$

4.5.4 Dans les théories étendues

Dans la formulation du théorème de Cantor en théorie des types avec une fonction, on peut étendre le système de réécriture avec la règle

$$(f\ (g\ x)) \to x$$

de manière à faire entrer dans la congruence l'égalité entre $(f\ (g\ x))$ et x. L'hypothèse H peut être remplacée par la démonstration $\lambda x\ \lambda p\ (\lambda\beta\ \beta, \lambda\beta\ \beta)$ et la démonstration ci-dessus se réduit sur

$$(\lambda\alpha\ (\alpha\ \alpha)\ \lambda\alpha\ (\alpha\ \alpha))$$

qui n'a pas de forme normale. Remarquons que, dans ce cas, le système de réécriture lui-même ne termine pas car la proposition A se réduit sur sa négation.

Dans les formulations en théorie des types avec une relation et dans la formulation en théorie des ensembles, on utilise l'axiome F pour montrer la proposition $C = C$ puis on applique l'axiome de Leibniz de manière à remplacer C par C dans la proposition $\neg A$. Dans [28], Ekman propose d'étendre la notion de coupure en théorie des ensembles en considérant comme une coupure une application de l'axiome de Leibniz à une égalité de la forme $a = b$ si a et b sont liées par une certaine relation d'équivalence. En suivant cette idée, on peut considérer comme une coupure une application de l'axiome de Leibniz à une égalité de la forme $a = a$ que cette proposition soit démontrée par l'axiome d'identité *ou de n'importe quel autre manière.* Une telle coupure s'élimine naturellement en supprimant la démonstration de la proposition $a = a$ et l'application de l'axiome de Leibniz.

Cela amène à ajouter une règle de réduction des démonstrations liée à l'axiome F

$$(fst(F\ x\ y\ y\ \alpha\ \alpha'\ P)\ \beta) \to \beta$$

en théorie des types et

$$(L\ x\ y\ y\ (F\ x'\ y\ y\ \alpha\ \alpha')\ \beta) \to \beta$$

en théorie des ensembles.

Pour cette notion de coupure étendue la démonstration π_0 ci-dessus (en théorie des types, comme en théorie des ensembles) n'est plus normale, et de plus elle n'est pas normalisable.

Les deux démonstrations du théorème de Cantor en théorie des types ont une forme normale (comme toutes les démonstrations en théorie des types). La démonstration en théorie des ensembles également. En revanche si dans une théorie ou dans l'autre on étend la congruence, on obtient des démonstrations qui n'ont plus de formes normales. La normalisation du théorème de Cantor dépend donc essentiellement de la congruence qu'on utilise dans la théorie dans laquelle on formalise ce théorème.

4.6 Les démonstrations objets de la théorie

4.6.1 Les motivations

Dans ce chapitre, nous avons proposé un λ-calcul qui permet d'exprimer les démonstrations d'une théorie modulo. Le langage des termes de cette théorie et le langage des démonstrations sont distincts. Autrement dit, les démonstrations ne sont pas des objets de la théorie elle-même. Ce type de calcul est traditionnel en théorie de la démonstration, où le but est d'étudier les démonstrations d'une théorie donnée *a priori* (par exemple, le système F [36, 37, 38, 59, 34] permet d'exprimer les démonstrations de la logique du second ordre, mais les termes du systeme F ne sont pas ceux de la logique du second ordre).

L'interprétation fonctionnelle des démonstrations peut avoir un autre objectif qui est d'enrichir une théorie (par exemple la théorie des types simples) de manière à ce que les démonstrations deviennent des objets à part entière de la théorie, à l'instar des nombres, des fonctions ou des ensembles.

Un objectif dans la construction d'une telle théorie est d'avoir des règles de réduction pour l'opérateur de descriptions (et de pouvoir donc programmer en langage mathématique [64, 79, y]). Les mathématiques informelles permettent, en effet, de définir des fonctions non seulement explicitement (par exemple, "$\lambda x\ x$") mais également implicitement (par exemple, "la fonction f telle que $\forall x\ f(x, 0) = 1 \wedge \forall x\ \forall n\ f(x, n+1) = x \times f(x, n)$"). Quand une fonction est ainsi définie avec l'opérateur de descriptions ("l'objet tel que ..."), il n'est pas absolument évident de proposer des règles de réduction qui permettent de réduire, par exemple, le terme $f(2, 3)$ sur le terme 8. Une solution consiste à inclure dans la définition de la fonction f une démonstration intuitionniste d'existence d'un objet vérifiant la propriété définissante. La réduction d'un terme de la forme $f(2, 3)$ consiste alors à éliminer les coupures dans cette démonstration. Pour pouvoir inclure une démonstration d'existence dans la définition de f (c'est-à-dire, formellement, que l'opérateur de descriptions prenne en argument non seulement une propriété définissante, mais aussi une démonstration d'existence d'un objet vérifiant cette propriété), il est nécessaire que les démonstrations soient des objets à part entière de la théorie.

On trouvera dans ce paragraphe quelques pistes pour la construction d'un λ-calcul avec des types dépendants modulo, dans lequel on exprime à la fois les objets, les propositions et les démonstrations d'une théorie modulo. Ce calcul se situe dans la tradition de la théorie des types de Martin-Löf, du Calcul des constructions et du Calcul des constructions inductives.

Ces pistes sont encore largement à explorer et l'étude mathématique du calcul proposé ici reste à faire.

4.6.2 Le plongement des théories du premier ordre dans le λ-calcul

Commençons par le cas d'une théorie du premier ordre. On construit un λ-calcul typé avec des types dépendants, un produit cartésien dépendant, une union disjointe et un type vide.

Ce calcul comporte une constante *Type* qui est à la fois le type des sortes de la théorie et celui de ses propositions. Les types fonctionnels servent à la fois à exprimer la fonctionnalité des symboles de fonction, celle des symboles de prédicat et les propositions formées avec l'implication, la négation et le quantificateur

universel. Le type produit cartésien sert à exprimer les propositions formées avec la conjonction et le quantificateur existentiel, l'union disjointe sert à former les propositions formées avec la disjonction, et le type vide les propositions formées avec la négation et la contradiction.

Pour que les propositions atomiques aient le type $Type$ il est nécessaire de plonger les symboles de prédicat de rang $(s_1, ..., s_n)$ comme des symboles de type $s_1 \to ... \to s_n \to Type$. Cela demande de pouvoir former des termes tels que $s_1 \to ... \to s_n \to Type$. On introduit pour cela une nouvelle constante $Kind$ et on donne le type $Kind$ à ces termes. On aboutit au calcul de la figure 4.2. Remarquons que contrairement aux présentations traditionnelles du λ-calcul avec des types dépendants [43, 11], ce calcul ne comporte ni règle de conversion, ni règle d'abstraction pour former des objets dont le type est de type $Kind$:

$$\frac{\Gamma \vdash T : Type \quad \Gamma, x : T \vdash T' : Kind \quad \Gamma, x : T \vdash t : T'}{\Gamma \vdash \lambda x : T\ t : \Pi x : T\ T'}$$

Suivant l'usage, on note $T \to U$ le terme $\Pi x : T\ U$ et $T \times U$ le terme $\Sigma x : T\ U$ quand la variable x n'apparaît pas dans U.

On pose, pour chaque sorte s de la théorie, une variable s' de type $Type$, pour chaque variable x de sorte s une variable x' de type s', pour chaque symbole de fonction f de rang $(s_1, ..., s_n, t)$ une variable f' de type $s'_1 \to ... \to s'_n \to t'$, pour chaque symbole de prédicat P de rang $(s_1, ..., s_n)$ une variable P' de type $s'_1 \to ... \to s'_n \to Type$. Les propositions se plongent alors comme des termes de type $Type$ dans ce contexte :
- $(f(t_1, ..., t_n))' = (f'\ t'_1\ ...\ t'_n)$,
- $(P(t_1, ..., t_n))' = (P'\ t'_1\ ...\ t'_n)$,
- $(A \Rightarrow B)' = A' \to B'$,
- $(A \wedge B)' = A' \times B'$,
- $(A \vee B)' = A' + B'$,
- $(\neg A)' = A' \to \emptyset$,
- $\bot' = \emptyset$,
- $(\forall x\ A)' = \Pi x : T\ A'$,
- $(\exists x\ A)' = \Sigma x : T\ A'$.

Pour chaque axiome A, on pose une variable x_A de type A'. Les démonstrations d'une proposition A peuvent alors s'exprimer comme des termes de type A'.

4.6.3 Le plongement des théories modulo dans le λ-calcul

On peut, de même, plonger une théorie modulo, en transformant les règles de ce calcul de manière à autoriser une conversion à chaque étape de déduction (on remplace alors le symbole $\vdash$ par le symbole $\vdash_\equiv$). De manière équivalente, on peut ajouter une règle de conversion :

$$\frac{\Gamma \vdash_\equiv t : T \quad \Gamma \vdash_\equiv T : Type \quad \Gamma \vdash_\equiv T' : Type \quad T \equiv T'}{\Gamma \vdash_\equiv t : T'}$$

Dans ce cas, cela dit, les étapes de conversions apparaissent explicitement dans la dérivation bien qu'elles n'apparaissent pas dans le λ-terme.

4.6.4 Le cas de la théorie des types simples

Comme toute théorie modulo, la théorie des types simples avec des combinateurs peut se plonger dans le λ-calcul avec des types dépendants. Il y a, malgré tout, une petite difficulté due au fait que cette théorie a un nombre infini de sortes : ι, o, $\iota \to o$, ... Pour ne pas avoir à introduire un nombre infini de variables de type $Type$, on peut introduire deux variables ι et o, un nouveau symbole $Arrow$ et une nouvelle règle

$$\frac{\Gamma \vdash_\equiv A : Type \quad \Gamma \vdash_\equiv B : Type}{\Gamma \vdash_\equiv Arrow(A, B) : Type}$$

Déclaration et utilisation des variables :

$$\frac{}{[\,]\text{ bien formé}}$$

$$\frac{\Gamma \vdash T : Kind}{\Gamma, x : T \text{ bien formé}}$$

$$\frac{\Gamma \vdash T : Type}{\Gamma, x : T \text{ bien formé}}$$

$$\frac{\Gamma \text{ bien formé} \quad x : T \in \Gamma}{\Gamma \vdash x : T}$$

Construction des objets de type $Kind$:

$$\frac{\Gamma \text{ bien formé}}{\Gamma \vdash Type : Kind}$$

$$\frac{\Gamma \vdash T : Type \quad \Gamma, x : T \vdash T' : Kind}{\Gamma \vdash \Pi x : T\ T' : Kind}$$

Construction des objets de type $Type$:

$$\frac{\Gamma \vdash T : Type \quad \Gamma, x : T \vdash T' : Type}{\Gamma \vdash \Pi x : T\ T' : Type}$$

$$\frac{\Gamma \vdash T : Type \quad \Gamma, x : T \vdash T' : Type}{\Gamma \vdash \Sigma x : T\ T' : Type}$$

$$\frac{\Gamma \vdash T : Type \quad \Gamma \vdash T' : Type}{\Gamma \vdash T + T' : Type}$$

$$\frac{\Gamma \text{ bien formé}}{\Gamma \vdash \emptyset : Type}$$

Construction et utilisation des fonctions :

$$\frac{\Gamma \vdash T : Type \quad \Gamma, x : T \vdash T' : Type \quad \Gamma, x : T \vdash t : T'}{\Gamma \vdash \lambda x : T\ t : \Pi x : T\ T'}$$

$$\frac{\Gamma \vdash t : \Pi x : T\ T' \quad \Gamma \vdash t' : T}{\Gamma \vdash (t\ t') : [t'/x]T'}$$

Construction et utilisation des couples :

$$\frac{\Gamma \vdash T : Type \quad \Gamma, x : T \vdash T' : Type \quad \Gamma \vdash a : T \quad \Gamma \vdash b : [a/x]T'}{\Gamma \vdash (p^{\Sigma x : T\ T'}\ a\ b) : \Sigma x : T\ T'}$$

$$\frac{\Gamma \vdash U : Type \quad \Gamma \vdash a : \Sigma x : T\ T' \quad \Gamma, x : T, y : T' \vdash b : U}{\Gamma \vdash (E\ a\ xyb) : U}$$

Construction et utilisation des éléments des unions disjointes :

$$\frac{\Gamma \vdash T : Type \quad \Gamma \vdash T' : Type \quad \Gamma \vdash a : T}{\Gamma \vdash (i^{T,T'}\ a) : T + T'}$$

$$\frac{\Gamma \vdash T : Type \quad \Gamma \vdash T' : Type \quad \Gamma \vdash b : T'}{\Gamma \vdash (j^{T,T'}\ b) : T + T'}$$

$$\frac{\Gamma \vdash a : T + T' \quad \Gamma \vdash U : Type \quad \Gamma, x : T \vdash b : U \quad \Gamma, y : T' \vdash c : U}{\Gamma \vdash (\delta^{T,T',U}\ a\ xb\ yc) : U}$$

Utilisation des éléments du type vide :

$$\frac{\Gamma \vdash t : \emptyset}{\Gamma \vdash (R^T\ t) : T}$$

FIGURE 4.2 – Le λ-calcul avec des types dépendants

Mais ce plongement n'est pas satisfaisant. En effet, la théorie des types simples comportant déjà une notion de fonction le calcul obtenu comporte deux notions, redondantes, de fonction : celle propre au λ-calcul et celle importée de la théorie des types. Au type $A \rightarrow B$ que permet de former le λ-calcul avec des types dépendants, s'ajoute le type $Arrow(A, B)$ qui est une sorte de la théorie des types simples. De même, au terme $\lambda x\ \lambda y\ x$ s'ajoute le terme K de la théorie des types simples.

Autrement dit, dans le cas de la théorie des types simples, parce que cette théorie comporte déjà une notion de fonction, on peut optimiser le plongement naïf ci-dessus : on ne pose que deux variables ι et o de type $Type$, et on plonge la sorte $T \rightarrow U$ (flèche de la théorie des types simples) comme le type $T \rightarrow U$ (flèche du λ-calcul). On plonge le combinateur K comme le terme $\lambda x\ \lambda y\ x$, et on plonge de manière similaire les autres combinateurs et les termes formés par application. Il faut alors ajouter la β-conversion des termes du λ-calcul avec des types dépendants dans la congruence $\equiv$ utilisée dans la règle de conversion pour remplacer les règles de réduction des combinateurs.

On peut s'étonner du fait que nous parvenions ici à exprimer les démonstrations de la théorie des types simples dans le λ-calcul avec des types dépendants (non polymorphe) alors que le polymorphisme a précisément été inventé pour exprimer les démonstrations de cette théorie [36, 37, 38, 59, 34]. Cela est dû au fait que, si nous avons identifié l'implication $\Rightarrow$ avec la flèche $\rightarrow$ du λ-calcul typé, et également la flèche $\rightarrow$ de construction des sortes de la théorie des types simples avec la flèche $\rightarrow$ du λ-calcul typé, nous avons, en revanche, gardé la distinction entre l'implication $\Rightarrow$ et son contenu $\dot{\Rightarrow}$ qui a été introduite dans la présentation au premier ordre de la théorie des types simples.

Certes, nous ne pouvons pas typer le terme $\lambda X : Type\ \lambda x : X\ x$ de type $\Pi X : Type\ (X \rightarrow X)$, mais nous pouvons typer le terme $\lambda X : o\ \lambda x : \varepsilon(X)\ x$ de type $\Pi X : o\ (\varepsilon(X) \rightarrow \varepsilon(X))$. Certes, nous ne pouvons pas appliquer le terme $\lambda X : Type\ \lambda x : X\ x$ au type $T \rightarrow T$ pour obtenir le terme $\lambda x : T \rightarrow T\ x$ de type $(T \rightarrow T) \rightarrow (T \rightarrow T)$, mais nous pouvons appliquer le terme $\lambda X : o\ \lambda x : \varepsilon(X)\ x$ au terme $t \dot{\Rightarrow} t$ et obtenir le terme $\lambda x : \varepsilon(t \dot{\Rightarrow} t)\ x$ de type $\varepsilon(t \dot{\Rightarrow} t) \rightarrow \varepsilon(t \dot{\Rightarrow} t)$ qui est identique, modulo la congruence, au terme $\lambda x : \varepsilon(t) \rightarrow \varepsilon(t)\ x$ de type $(\varepsilon(t) \rightarrow \varepsilon(t)) \rightarrow (\varepsilon(t) \rightarrow \varepsilon(t))$.

Cet exemple nous montre que le polymorphisme peut être obtenu en λ-calcul avec des types dépendants modulo, non seulement par l'ajout de règles de typage, comme c'est l'usage, mais aussi par l'ajout de règles de réécriture. On obtient ainsi une présentation du Calcul des constructions qui si elle n'est pas la plus courante, n'est pas, pour autant, nouvelle. Elle a, par exemple, été étudiée en détails par T. Altenkirch [2] qui note pr notre symbole ε et qui distingue le terme P de type o du type $pr(P)$ des démonstrations de P.

On retombe sur la présentation traditionnelle du Calcul des constructions en optimisant encore le plongement : au lieu de poser deux variables ι et o on ne pose que la variable ι et on plonge la sorte o comme le symbole $Type$. De même, on plonge le symbole $\dot{\Rightarrow}$ comme la flèche du λ-calcul, et on donne un plongement similaire pour les symboles $\dot{\wedge}$, $\dot{\vee}$, ... Il est alors nécessaire d'ajouter la règle des types polymorphes pour exprimer la quantification sur les objets de type o qui sont désormais de type $Type$, la règle des constructeurs de types pour exprimer la sorte $o \rightarrow o$, et la règle de construction d'abstractions dont le type est de type $Kind$ pour former le terme $\lambda x : o\ \lambda y : \iota\ x$ qui est désormais de type $Type \rightarrow \iota \rightarrow Type$.

Ces différentes présentations ne sont pas nécessairement équivalentes. En particulier, il reste à déterminer celles dans lesquelles on peut exprimer une démonstration de l'axiome du choix ou le paradoxe de la somme forte.

Il y a, cela dit, un avantage à garder la distinction entre les connecteurs et leur contenus. En effet, les extensions du λ-calcul avec des types dépendants sont traditionnellement classées en deux familles : celles obtenues par l'ajout de règles de typage (qui sont élégamment présentées dans le cube de H. Barendregt [11]) : le polymorphisme et les constructeurs de types, et celles qui sont obtenues par l'ajout de règles de réécriture : les types inductifs. Dans le λ-calcul avec des types dépendants modulo, toutes ces extensions sont obtenues de manière uniforme par l'ajout de règles de réécriture.

Notons enfin qu'on peut aussi plonger la théorie des types simples présentée avec des substitutions explicites dans le λ-calcul avec des types dépendants. Mais, pour optimiser le plongement comme ci-dessus, il est nécessaire de présenter le λ-calcul avec des types dépendants lui-même avec des substitutions explicites, comme l'ont proposé P. Martin-Löf [66], L. Magnusson [62] et C. Muñoz [77, 74, 75].

Chapitre 5

La démonstration automatique

5.1 La résolution équationnelle

Dans une théorie modulo dont la congruence est définie par des règles de réécriture et des équations portant sur les termes uniquement, les méthodes de démonstration automatique sont assez similaires à celles pour la logique du premier ordre. On peut utiliser la résolution, ou la méthode des tableaux, en remplaçant simplement l'algorithme d'unification du premier ordre par un algorithme d'unification équationnelle modulo cette congruence. Cette idée n'est pas nouvelle, Plotkin propose dans [80] d'abandonner l'axiome d'associativité en remplaçant l'unification par l'unification associative (ce qui revient, dans notre cadre, à raisonner modulo l'associativité) et M. Stickel [87] donne un cadre général à la résolution équationnelle qu'il appelle *theory resolution*.

Un problème d'unification équationnelle est une équation $a = b$ (ou un ensemble d'équations) et une solution en est une substitution σ telle que les termes σa et σb soient congruents (et non plus nécessairement identiques).

L'algorithme général d'unification équationnelle est appelé la *surréduction* [30, 55, 35, 57]. L'idée de base est d'ajouter aux règles de l'unification du premier ordre, une règle qui permet de surréduire (c'est-à-dire d'instancier puis de réduire) un sous-terme d'une équation qui n'est pas réductible mais dont une instance l'est.

Par exemple, le problème d'unification

$$z + 5 = 7$$

n'a pas de solution pour l'unification du premier ordre, mais il a une solution $z \mapsto 2$, modulo les règles de réécriture

$$0 + y \to y$$

$$Succ(x) + y \to Succ(x + y)$$

En effet, le terme $z + 5$ n'est pas réductible, c'est-à-dire que ce n'est pas une instance d'un membre gauche d'une des règles, ou encore qu'il n'est pas filtré (au sens du filtrage du premier ordre) par le membre gauche de l'une des règles, mais il est surréductible c'est-à-dire qu'une de ses instance est réductible ou encore qu'il est unifiable (au sens de l'unification du premier ordre) avec le membre gauche de l'une des règles. En unifiant, au sens ordinaire, le terme $z + 5$ avec le terme $Succ(x) + y$ on instancie z en $Succ(z')$. Le terme $Succ(z') + 5$ se réduit alors en $Succ(z' + 5)$. En procédant de même on instancie z' en $Succ(z'')$, puis z'' en 0 ce qui donne la solution $z \mapsto 2$.

5.2 La résolution modulo

5.2.1 La règle de surréduction étendue

Dans le cas général, où on a des règles de réécriture qui réécrivent des propositions atomiques en des propositions non atomiques, la résolution équationnelle et la méthode des tableaux équationnelle, ne sont plus complètes. Par exemple, modulo la règle

$$x \times y = 0 \to x = 0 \vee y = 0$$

on peut démontrer la proposition

$$a \times a = 0 \Rightarrow a = 0$$

et donc

$$\exists y \ (a \times a = y \Rightarrow a = y)$$

Pourtant, la mise en forme clausale de la négation de cette proposition donne les deux clauses

$$a \times a = Y$$

$$\neg a = Y$$

et les propositions $a \times a = Y$ et $a = Y$ n'étant pas unifiables, on ne peut pas appliquer la règle de résolution avec succès.

Ce problème a déjà été rencontré dans le cadre de la méthode de résolution d'ordre supérieur de Huet [50, 51]. En logique d'ordre supérieur, en effet, la la proposition

$$\exists x \ x$$

est démontrable car la proposition $A \vee \neg A$ l'est, mais on ne peut rien déduire de la forme clausale de sa négation. Dans notre présentation, cela s'exprime par le fait que la proposition

$$\exists x \ \varepsilon(x)$$

est démontrable car la proposition $\varepsilon(A \dot{\vee} \dot{\neg} A)$ l'est, mais on ne peut rien déduire de la forme clausale de sa négation $\neg\varepsilon(X)$.

En résolution d'ordre supérieur, cela a amené à proposer une nouvelle règle de déduction, la règle de *scission* (*splitting*), qui consiste à instancier arbitrairement les littéraux dont le symbole de tête est une variable en des propositions formées avec des connecteurs et des quantificateurs. Par exemple, en partant de la clause $\neg X$, on instancie la variable X par le terme $Y \vee Z$ ce qui donne après mise en forme clausale de la proposition $\neg(Y \vee Z)$ les deux clauses

$$\neg Y$$

$$\neg Z$$

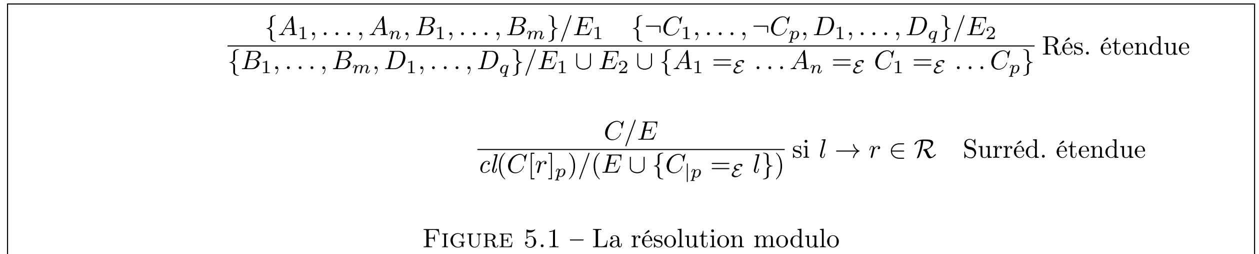
on instancie ensuite la variable Z par le terme $\neg Z'$ ce qui donne les deux clauses

$$\neg Y$$

$$Z'$$

on applique ensuite la règle de résolution pour obtenir la clause vide. La règle de scission et la règle de résolution d'ordre supérieur forment ensemble un système complet pour la logique d'ordre supérieur [50, 51].

Dans le cadre de la déduction modulo, il faut ajouter à la règle de résolution une règle qui anticipe l'instanciation d'un littéral, permettant de le réduire sur une proposition composée. Par exemple, le littéral



$$\frac{\{A_1,\ldots,A_n,B_1,\ldots,B_m\}/E_1 \quad \{\neg C_1,\ldots,\neg C_p,D_1,\ldots,D_q\}/E_2}{\{B_1,\ldots,B_m,D_1,\ldots,D_q\}/E_1 \cup E_2 \cup \{A_1 =_{\mathcal{E}} \ldots A_n =_{\mathcal{E}} C_1 =_{\mathcal{E}} \ldots C_p\}} \text{ Rés. étendue}$$

$$\frac{C/E}{cl(C[r]_p)/(E \cup \{C_{|p} =_{\mathcal{E}} l\})} \text{ si } l \to r \in \mathcal{R} \quad \text{Surréd. étendue}$$

FIGURE 5.1 – La résolution modulo

$a \times a = Y$ s'instancie en $a \times a = 0$ qui se réduit en la proposition $a = 0 \vee a = 0$ dont la forme clausale est $a = 0$. Le fait qu'une instance du littéral $a \times a = Y$ soit réductible par la règle

$$x \times y = 0 \to x = 0 \vee y = 0$$

c'est-à-dire que le littéral $a \times a = Y$ soit surréductible par cette règle, peut s'exprimer par le fait que ce littéral et le membre gauche de la règle sont unifiables. La règle qu'il est nécessaire d'ajouter à la règle de résolution ressemble donc beaucoup à la règle de surréduction de l'algorithme d'unification équationnelle. Mais à la différence de cette règle, elle s'applique directement aux clauses du problème de recherche et non aux équations du problème d'unification. Elle consiste à unifier un littéral avec le membre gauche d'une règle de réduction des propositions, à réduire ce littéral et à remettre la proposition obtenue en forme clausale. Nous l'avons appelée *surréduction étendue*. C'est l'homologue de la scission pour la déduction modulo. Cela nous amène à partitionner les règles de réécriture et les équations en deux ensembles : l'ensemble $\mathcal{E}$ des équations et règles entre termes (et éventuellement entre propositions atomiques) et l'ensemble $\mathcal{R}$ des règles qui transforment des propositions atomiques en des propositions composées. Les premières sont utilisées par l'algorithme d'unification équationnelle et les secondes par la règle de surréduction étendue.

5.2.2 Les contraintes

Comme d'habitude, quand les problèmes d'unification ne sont pas décidables et unitaires, la bonne formulation de la résolution est celle de la résolution contrainte [50, 51]. Les problèmes d'unification ne sont pas toujours immédiatement résolus au moment de l'application des règles, mais leur résolution peut être retardée. On les garde alors comme des contraintes. Nous aboutissons alors au système de la figure 5.1 appelé *Résolution et surréduction étendues* ou, plus simplement, *Résolution modulo*.

5.2.3 La complétude

La complétude de cette méthode s'exprime comme le fait que si une proposition A est démontrable en déduction modulo, alors de la forme clausale de sa négation, on peut déduire par les règles ci-dessus une clause contrainte vide, c'est-à-dire une clause de la forme $\square/E$ où E est un ensemble unifiable de contraintes.

Dans [g], on montre que cette complétude est atteinte pour toutes les théories modulo qui ont la propriété d'élimination des coupures, faute de quoi il faut se restreindre aux propositions A qui ont une démonstration sans coupures.

5.2.4 Un exemple simple

Pour comprendre pourquoi il est plus efficace de transformer un axiome $A \Leftrightarrow B$ en une règle de réécriture de l'ensemble $\mathcal{R}$ et d'utiliser la règle de surréduction étendue plutôt que de garder cet axiome tel quel, on peut considérer l'exemple suivant.

Si on veut réfuter la théorie $P_1 \Leftrightarrow (Q_2 \vee P_2)$, P_1, $Q_2 \Leftrightarrow \bot$, $P_2 \Leftrightarrow \bot$, la solution naïve donne les clauses

$$\neg P_1, Q_2, P_2$$

$$\neg Q_2, P_1$$
$$\neg P_2, P_1$$
$$P_1$$
$$\neg Q_2$$
$$\neg P_2$$

En revanche, si on transforme la proposition $P_1 \Leftrightarrow (Q_2 \vee P_2)$ en une règle de réécriture

$$P_1 \rightarrow Q_2 \vee P_2$$

et les propositions $Q_2 \Leftrightarrow \bot$ et $P_2 \Leftrightarrow \bot$, en les règles

$$Q_2 \rightarrow \bot$$
$$P_2 \rightarrow \bot$$

il ne reste plus que la proposition P_1 a mettre en forme clausale. Cette proposition se réduit sur $\bot \vee \bot$. Sa forme clausale est la clause vide.

Plus généralement, si on veut réfuter la théorie $P_1 \Leftrightarrow (Q_2 \vee P_2)$, ..., $P_i \Leftrightarrow (Q_{i+1} \vee P_{i+1})$, ..., $P_n \Leftrightarrow (Q_{n+1} \vee P_{n+1})$, P_1, $Q_2 \Leftrightarrow \bot$, ..., $Q_{n+1} \Leftrightarrow \bot$, $P_{n+1} \Leftrightarrow \bot$, la solution naïve donne les $4n+2$ clauses

$$\neg P_1, Q_2, P_2$$
$$\neg Q_2, P_1$$
$$\neg P_2, P_1$$
$$...$$
$$\neg P_i, Q_{i+1}, P_{i+1}$$
$$\neg Q_{i+1}, P_i$$
$$\neg P_{i+1}, P_i$$
$$...$$
$$\neg P_n, Q_{n+1}, P_{n+1}$$
$$\neg Q_{n+1}, P_{n+1}$$
$$\neg P_{n+1}, P_{n+1}$$
$$P_1$$
$$\neg Q_2$$
$$...$$
$$\neg Q_{n+1}$$
$$\neg P_{n+1}$$

En revanche, si on transforme les propositions $P_i \Leftrightarrow (Q_{i+1} \vee P_{i+1})$ en des règles de réécriture

$$P_i \rightarrow Q_{i+1} \vee P_{i+1}$$

les propositions $Q_i \Leftrightarrow \bot$ en des règles de réécriture

$$Q_i \rightarrow \bot$$

et la proposition $P_n \Leftrightarrow \bot$ en une règle de réécriture

$$P_n \rightarrow \bot$$

il ne reste plus que la proposition P_1 a mettre en forme clausale. Cette proposition se réduit sur $\bot \vee ... \vee \bot$ et sa forme clausale est la clause vide.

Certes, réduire la proposition P_1 a un certain coût, mais ce coût est sans comparaison avec celui de la recherche non déterministe d'une réfutation par résolution de l'ensemble de clauses ci-dessus. Ce processus de réduction est, en effet, déterministe car le système de réécriture est confluent.

5.3 La démonstration automatique en théorie des types simples

5.3.1 La surréduction et la scission

Dans le cas de la théorie des types simples, on partitionne les règles de réécriture en deux ensembles. Les règles de l'ensemble $\mathcal{E}$ qui sont exactement les règles du $\lambda\sigma$-calcul (avec η-réduction)

$$(\lambda a)b \to a[b.id]$$

$$\lambda(a\ 1) \to b \text{ si } a =_\sigma b[\uparrow]$$

$$(a\ b)[s] \to (a[s]\ b[s])$$
$$1[a.s] \to a$$
$$a[id] \to a$$
$$(\lambda a)[s] \to \lambda(a[1.(s \circ \uparrow)])$$
$$(a[s])[t] \to a[s \circ t]$$
$$id \circ s \to s$$
$$\uparrow \circ\ (a.s) \to s$$
$$(s_1 \circ s_2) \circ s_3 \to s_1 \circ (s_2 \circ s_3)$$
$$(a.s) \circ t \to a[t].(s \circ t)$$
$$s \circ id \to s$$
$$1. \uparrow \to id$$
$$1[s].(\uparrow \circ\ s) \to s$$

et les règles de l'ensemble $\mathcal{R}$ qui sont les autres règles

$$\varepsilon(\dot{\wedge}\ x\ y) \to \varepsilon(x) \wedge \varepsilon(y)$$

$$\varepsilon(\dot{\vee}\ x\ y) \to \varepsilon(x) \vee \varepsilon(y)$$
$$\varepsilon(\dot{\Rightarrow}\ x\ y) \to \varepsilon(x) \Rightarrow \varepsilon(y)$$
$$\varepsilon(\dot{\neg}\ x) \to \neg\varepsilon(x)$$
$$\varepsilon(\dot{\bot}) \to \bot$$
$$\varepsilon(\dot{\forall}_T\ x) \to \forall y\ \varepsilon(x\ y)$$
$$\varepsilon(\dot{\exists}_T\ x) \to \exists y\ \varepsilon(x\ y)$$

Remarquons qu'en logique classique, on peut se restreindre à n'utiliser que les connecteurs $\vee$ et $\neg$ et le quantificateur $\forall$, les autres pouvant se définir à l'aide des règles de De Morgan. De même, on peut se restreindre à n'utiliser que les symboles $\dot{\vee}$, $\dot{\neg}$ et $\dot{\forall}$. Dans ce cas le système $\mathcal{R}$ se réduit aux règles

$$\varepsilon(\dot{\vee}\ x\ y) \to \varepsilon(x) \vee \varepsilon(y)$$

$$\varepsilon(\dot{\neg}\ x) \to \neg\varepsilon(x)$$
$$\varepsilon(\dot{\forall}_T\ x) \to \forall y\ \varepsilon(x\ y)$$

La règle de surréduction étendue se spécialise alors exactement en la règle de scission de [50, 51]. En effet, un littéral réduit est surréductible s'il n'a pas la forme $\varepsilon(\dot{\vee}\ x\ y)$, $\varepsilon(\dot{\neg}\ x)$, $\varepsilon(\dot{\forall}\ x)$, mais s'il peut s'unifier avec

une telle proposition. Il est facile de montrer que cette condition est équivalente au fait que le symbole de tête d'un tel littéral est une variable. La règle de surréduction étendue consiste à remplacer un tel littéral L, ou bien par $\varepsilon(x) \vee \varepsilon(y)$ en ajoutant la contrainte $L = \varepsilon(\dot{\vee}\ x\ y)$, ou bien par $\neg\varepsilon(x)$ en ajoutant la contrainte $L = \varepsilon(\dot{\neg}\ x)$, ou bien par $\forall y\ \varepsilon(x\ y)$ en ajoutant la contrainte $L = \varepsilon(\dot{\forall}\ x)$. C'est exactement la règle de scission.

La règle de scission n'est donc pas propre à la théorie des types. La nécessité d'utiliser une telle règle s'explique par le fait que la théorie des types contient des règles qui réécrivent des propositions atomiques en des propositions non atomiques. La forme de la règle de scission se déduit simplement de la forme de ces règles de réécriture.

5.3.2 L'unification équationnelle et l'unification d'ordre supérieur

L'unification équationnelle qui est utilisée dans cette méthode est l'unification équationnelle modulo le système $\mathcal{E}$ ci-dessus, c'est-à-dire modulo le $\lambda\sigma$-calcul avec η-réduction.

Nous avons donné dans [c] un algorithme d'unification équationnelle dans le $\lambda\sigma$-calcul qui peut être vu comme un algorithme optimisé de surréduction. Les optimisations portent sur trois points. D'une part, on utilise le théorème de terminaison de la réduction pour ne considérer que des équations en forme normale [90] et pour réduire l'espace de recherche en spéculant sur la forme normale des solutions (en particulier sur leur symbole de tête). Ensuite, on reconnaît certaines équations qui n'ont pas de solution et on en simplifie d'autres en gardant le même ensemble de solutions. Enfin, on reconnaît certaines équations (entre termes flexibles) qui ont toujours des solutions et qu'il n'est donc pas nécessaire de résoudre puisque l'utilisation des contraintes permet de se limiter au test d'unifiabilité et ne demande jamais l'énumération des solutions [50, 51].

Nous avons établi dans [c] un théorème d'équivalence entre l'unification équationnelle modulo la théorie $\lambda\sigma$ et l'unification d'ordre supérieur [52, 53, 86]. Ce théorème exprime qu'un problème d'unification d'ordre supérieur

$$a = b$$

a une solution si et seulement si c'est également le cas du problème d'unification équationnelle dans $\lambda\sigma$

$$a_F = b_F$$

où $._F$ est l'injection du λ-calcul dans le $\lambda\sigma$-calcul appelée précuisson (*pre-cooking*) et décrite au paragraphe 2.4.5 (mais qui a été introduite pour la première fois dans le cadre de cet algorithme d'unification [c]).

Le sens direct du théorème est facile : si le premier problème a une solution $c_1/x_1, ..., c_n/x_n$ alors le second a la solution $x_1 \mapsto c_{1F}, ..., x_n \mapsto c_{nF}$. Pour montrer la réciproque, on doit montrer que si le problème d'unification $a_F = b_F$ a une solution alors il a aussi une solution dans l'image de la précuisson.

Nous avons montré dans [c] que l'algorithme d'unification dans le $\lambda\sigma$-calcul reproduisait fidèlement l'algorithme d'unification d'ordre supérieur, c'est-à-dire que chaque étape de cet algorithme pouvait être simulée par un nombre fini d'étapes de l'algorithme d'unification dans le $\lambda\sigma$-calcul.

Nous avons enfin montré dans [c] que l'algorithme d'unification dans le $\lambda\sigma$-calcul était plus simple, par certains aspects que l'algorithme d'unification d'ordre supérieur. En particulier l'utilisation de la substitution simple (du remplacement), et non de la substitution particulière du λ-calcul qui demande un renommage des variables, évite le codage fonctionnel de la portée des variables et permet une expression plus atomique des règles. Par exemple, une équation de la forme $\lambda a = \lambda b$ se simplifie en $a = b$ dans le $\lambda\sigma$-calcul alors que dans le λ-calcul une telle simplification est incorrecte puisque la variable liée par le λ ne peut pas être substituée aux variables libres de a et b dans la première équation, mais le peut dans la seconde. Autre exemple, une variable X de sorte $\Gamma \vdash T \rightarrow U$ peut être instanciée par un terme de la forme λY où Y est une variable de sorte $T\Gamma \vdash U$ alors qu'une telle instanciation est incomplète dans le λ-calcul, car il n'est plus possible ensuite d'instancier la variable Y par la variable liée par le symbole λ. Cette simplicité est due au fait que les contraintes de portée des variables sont incorporées dans les termes eux-mêmes par la précuisson.

Nous avons enfin montré dans [d] que la restriction de l'unification d'ordre supérieur aux *patterns* de Miller [72, 73] pouvait également se définir dans le $\lambda\sigma$-calcul. La décidabilité et le caractère unitaire de l'unification est alors une conséquence de l'inversibilité de certaines substitutions explicites.

5.3.3 De la résolution d'ordre supérieur à la résolution modulo

Par rapport à la résolution du premier ordre, la résolution d'ordre supérieur apporte trois nouveautés : l'unification d'ordre supérieur, la règle de scission et la skolémisation d'ordre supérieur (qui n'a, en fait, été formulée correctement que dix ans après l'apparition de la résolution d'ordre supérieur dans [70, 71]).

Plusieurs auteurs, en particulier M. Davis [24], ont insisté assez tôt sur le fait que la théorie des types n'était qu'une théorie du premier ordre comme les autres. On pouvait en tirer l'intuition que les outils introduits pour la résolution d'ordre supérieur avaient une portée plus générale et pouvaient s'appliquer à d'autres théories, en particulier à la théorie des ensembles. Cela suggérait le programme de définir une méthode de démonstration automatique pour la logique du premier ordre qui se spécialise exactement en la résolution d'ordre supérieur quand on l'applique à la formulation au premier ordre de la théorie des types.

Ce programme présentait deux difficultés : d'une part les présentations traditionnelles de la théorie des types comme une théorie du premier ordre utilisaient le schéma de compréhension ou des combinateurs, alors que l'unification d'ordre supérieur utilisait la notation plus élégante et plus compacte du λ-calcul. D'autre part, il semblait bien que la séparation entre calcul (β-réduction) et déduction que permettait la théorie des types était un élément essentiel pour le fonctionnement de l'unification d'ordre supérieur et de la scission.

Cette première difficulté a été surmontée par l'introduction des combinateurs catégoriques puis du calcul des substitutions explicites que nous avons utilisé dans [c] pour réduire l'unification d'ordre supérieur à l'unification équationnelle du premier ordre, puis dans [i] pour donner une formulation au premier ordre de la théorie des types intentionnellement équivalente à la formulation utilisant le λ-calcul. La seconde difficulté a été surmontée par l'introduction dans [g] d'un cadre général, *la déduction modulo*, qui permet de séparer calcul et déduction. Cela nous a permis de formuler une méthode de démonstration automatique pour la déduction modulo qui s'instancie exactement en la résolution d'ordre supérieur quand on l'applique à la formulation de la théorie des types donnée dans [i] et de réaliser ce programme.

Les particularités de la skolémisation d'ordre supérieur sont donc avant tout une conséquence du système de sortes du $\lambda\sigma$-calcul. L'unification d'ordre supérieur en revanche est une forme particulière d'unification équationnelle et la scission une forme particulière de surréduction étendue.

5.4 La démonstration automatique en théorie des ensembles

En théorie des ensembles, toutes les règles de réécriture transforment des propositions atomiques en des propositions composées, l'ensemble $\mathcal{E}$ est donc vide et l'ensemble $\mathcal{R}$ contient toutes les règles de la figure 3.1. L'unification est donc simplement l'unification du premier ordre. La résolution modulo est incomplète car elle ne permet de démontrer que les propositions qui ont une démonstration sans coupures. Elle permet de démontrer le théorème de Cantor, mais pas le théorème de Crabbé

$$\neg\{x \in A \mid \neg x \in x\} \in A$$

en effet notons f le symbole de Skolem introduit par la skolémisation de l'axiome

$$\forall A\ \exists B\ \forall x\ ((x \in B) \Leftrightarrow (x \in A \wedge \neg x \in x))$$

on a la règle de réécriture

$$x \in f(y) \to x \in y \wedge \neg x \in x$$

la proposition de Crabbé s'écrit

$$\neg f(a) \in a$$

la forme clausale de sa négation est formée de l'unique clause

$$f(a) \in a$$

Avec cette clause on ne peut pas appliquer la règle de résolution, on ne peut pas non plus appliquer la règle de surréduction étendue car on devrait alors unifier la proposition $f(a) \in a$ avec une proposition de la forme

$w \in \{\}(x, y)$, $w \in \mathcal{P}(x)$, $w \in \bigcup(x)$ ou $w \in g(x_1, ..., x_n)$. La contrainte se simplifierait en une contrainte de la forme

$$a = \{\}(x, y)$$

$$a = \mathcal{P}(x)$$

$$a = \bigcup(x)$$

ou

$$a = g(x_1, ..., x_n)$$

et aucune de ces équations n'a de solution puisque a est une constante.

Ici, il faudrait que la méthode de démonstration automatique devine la coupure qui se trouve dans la démonstration de Crabbé.

Les différences entre la résolution modulo en théorie des types et en théorie des ensembles sont discutées dans [j]. Cette discussion est illustrée par la recherche de démonstrations du théorème de Cantor dans les différentes formulations présentées au paragraphe 4.5.

Chapitre 6

Les types et la terminaison

6.1 Les types et les ensembles

Dans les chapitres précédents nous avons donné deux exemples de théories modulo, la théorie des types simples et la théorie des ensembles, qui se présentent, l'une et l'autre, comme des axiomatisations possibles des mathématiques. L'une et l'autre peuvent être vues comme des restrictions de la théorie naïve (incohérente) des ensembles. La théorie des ensembles restreint le schéma de compréhension à quatre instances et la théorie des types le restreint en le plaçant dans un langage multisorté.

La déduction modulo permet de jeter un regard nouveau sur les différences entre ces deux théories. Le système de réécriture associé à la théorie des ensembles ne termine pas, de ce fait la théorie des ensembles n'a pas la propriété d'élimination des coupures et la résolution modulo n'est pas complète. En revanche, le système de réduction de la théorie des types simples termine, cette théorie a la propriété d'élimination des coupures et la résolution modulo est complète.

Par ailleurs, la théorie des types dispose d'une notion primitive de fonction et de la possibilité de former un terme $\alpha(f, x)$ pour désigner l'image de l'objet x par la fonction f. Cela permet de définir des objets comme l'ensemble de Cantor plus simplement (en particulier avec moins de quantificateurs et de connecteurs) qu'en théorie des ensembles et la recherche de démonstrations fait moins souvent appel à la règle de surréduction qu'en théorie des ensembles.

Cependant, on ne peut pas ignorer certaines critiques souvent opposées à la théorie des types. D'abord, en utilisant un langage multisorté, c'est-à-dire restreint syntaxiquement, la théorie des types élude, plus qu'elle ne résout, le paradoxe de Russell. En caricaturant, on peut dire que la théorie des types résout le paradoxe du menteur en interdisant aux Crétois de prononcer la phrase "Tous les crétois sont menteurs". D'autre part il semble y avoir, dans cette théorie, une certaine redondance entre la notion de type et la notion d'ensemble, par exemple entre le type ι et l'ensemble $\lambda x \top$ de type $\iota \to o$ de tous les objets de type ι. Plus concrètement, on peut se demander comment choisir les ensembles qui sont destinés à être des types de base. Certains choisissent par exemple un type de base pour les entiers, alors que pour d'autres, les entiers ne sont que des objets de type $(\iota \to o) \to o$ qui se révèlent être des cardinaux finis. Autrement dit, la théorie des types renouvelle l'épineuse distinction entre propriété essentielle et propriété accidentelle.

Dans [82], Quine a résolu le premier problème en donnant une formulation de la théorie des types comme une théorie du premier ordre ordinaire (c'est-à-dire monosortée). Pour cela, il suffit de relativiser [29, 33] la présentation de la théorie des types comme une théorie du premier ordre multisortée. On peut, en effet, toujours traduire une théorie du premier ordre multisortée en une théorie monosortée en introduisant pour chaque symbole de fonction ou de prédicat un symbole de même arité et pour chaque sorte s un prédicat unaire $\mathcal{T}_s$ et en traduisant les propositions comme suit :

— $x' = x$, $(f(t_1, ..., t_n))' = f'(t'_1, ..., t'_n)$,
— $(P(t_1, ..., t_n))' = P'(t'_1, ..., t'_n)$,
— $(A \Rightarrow B)' = A' \Rightarrow B'$, $(A \wedge B)' = A' \wedge B'$, $(A \vee B)' = A' \vee B'$, $(\neg A)' = \neg A'$, $\bot' = \bot$,

— $(\forall x\ A)' = \forall x\ (\mathcal{T}_s(x) \Rightarrow A')$, $(\exists x\ A)' = \exists x\ (\mathcal{T}_s(x) \wedge A')$.

La traduction d'une théorie Γ s'effectue en traduisant chacune des propositions et en ajoutant pour chaque symbole de fonction f de rang $(s_1, ..., s_n, t)$ un axiome

$$\forall x_1\ ...\ \forall x_n((\mathcal{T}_{s_1}(x_1) \wedge ... \wedge \mathcal{T}_{s_n}(x_n)) \Rightarrow \mathcal{T}_t(f(x_1, ..., x_n)))$$

et pour chaque sorte s un axiome

$$\exists x\ \mathcal{T}_s(x)$$

On peut alors démontrer que $\Gamma \vdash P$ dans la théorie initiale si et seulement si $\Gamma' \vdash P'$ dans la théorie monosortée.

En exprimant ainsi la théorie des types comme une théorie du premier ordre monosortée, on résout le premier problème puisqu'on n'utilise plus d'artifice syntaxique pour éviter le paradoxe de Russell.

Comme la théorie des ensembles, la théorie des types apparaît comme une restriction du schéma de compréhension à certaines instances : celles qui sont stratifiables et dont les quantificateurs sont bornés. L'instance du schéma de compréhension utilisée dans le paradoxe de Russell

$$\exists R\ \forall x\ (R\ x) \Leftrightarrow \neg(x\ x)$$

n'est pas un axiome de la théorie car la proposition $\neg(x\ x)$ n'est la relativisation d'aucune proposition du langage multisorté. La question soulevée par Quine de savoir si on peut relâcher la condition des quantifications bornées et accepter toutes les instances stratifiables du schéma de compréhension, c'est-à-dire la cohérence de ses *Fondations nouvelles* (*New foundations* ou NF) est, à notre connaissance, toujours ouverte.

Si cette formulation de la théorie des types résout le premier problème, le second demeure car les types sont remplacés par les prédicats $\mathcal{T}_\iota, \mathcal{T}_o, \mathcal{T}_{\iota \to o}, ...$ qui sont toujours redondants avec certains ensembles. Dans [b], nous avons proposé une formulation de la théorie des types dans laquelle les types ne sont que des ensembles particuliers. Pour cela, on introduit dans le langage lui-même des symboles I, O et $\to$, on supprime les symboles $\mathcal{T}_s$ et on traduit les propositions de la forme $\mathcal{T}_s(t)$ par $\varepsilon(s'\ t)$, où s' est défini comme suit :

— $\iota' = I$, $o' = O$,

— $(T \to U)' = T' \to U'$.

L'expression $I \to I$, à la différence du type $\iota \to \iota$, est un terme du langage.

Nous montrons dans [b], en utilisant des techniques de construction de modèles, qu'une proposition est démontrable en théorie des types si et seulement si sa traduction l'est dans cette théorie.

6.2 La réduction dans un cadre monosorté

On peut, à présent, s'interroger sur la possibilité de présenter cette théorie comme une théorie modulo. Dans cette théorie, les axiomes de conversion sont des axiomes gardés. Par exemple, l'un de ces axiomes se formule

$$(\varepsilon((I \to I \to I)\ a) \wedge \varepsilon((I \to I)\ b) \wedge \varepsilon(I\ c)) \Rightarrow (S_{I,I,I}\ a\ b\ c) = (a\ c\ (b\ c))$$

Les règles de réécriture correspondantes doivent donc également être gardées. Pour réduire $(S_{I,I,I}\ a\ b\ c)$ en $(a\ c\ (b\ c))$, il faut au préalable "vérifier" que a appartient à l'ensemble $I \to I \to I$, b à l'ensemble $I \to I$ et c à l'ensemble I.

Autrement dit, rien dans ce langage n'interdit de former une proposition comme $\varepsilon(\lambda x\ \dot\neg(x\ x)\ \lambda x\ \dot\neg(x\ x))$, mais cette proposition n'est pas prouvablement équivalente à sa négation et *a fortiori* ne se réduit pas sur elle, car l'argument auquel la fonction est appliquée n'est pas dans son domaine de définition et donc l'image obtenue n'est pas spécifiée par la théorie.

Dans ce cadre restreint où les ensembles de définition des fonctions ne sont que quelques ensembles particuliers (les *types*, c'est-à-dire les ensembles formés à partir des ensembles I et O avec le constructeur d'espace fonctionnel $\to$), on peut espérer que ces gardes soient décidables. Mais nous proposons dans le dernier paragraphe de [b] une extension de cette théorie dans laquelle, comme dans les mathématiques

informelles, n'importe quel ensemble peut être ensemble de définition d'une fonction. Il est alors illusoire d'espérer décider les gardes des règles de réécriture.

Pour avoir à la fois un système de réécriture dont les gardes sont décidables et la possibilité de prendre n'importe quel ensemble comme ensemble de définition d'une fonction, il faut demander, à chaque fois qu'on applique une fonction à un objet, une démonstration que cet objet est dans l'ensemble de définition de cette fonction. Cela demande de faire des démonstrations des objets à part entière de la théorie.

6.3 Les démonstrations objets de la théorie

6.3.1 Les motivations

Si on demande, à chaque fois qu'on applique une fonction à un objet, une démonstration que cet objet est dans l'ensemble de définition de cette fonction, l'application n'est plus une opération binaire qui s'applique à une fonction et son argument, mais une opération ternaire qui s'applique à une fonction, son argument et une démonstration que cet argument est dans l'ensemble de définition de la fonction. Cette méthode est la manière traditionnelle de traiter des "fonctions partielles" (c'est-à-dire les fonctions dont le domaine de définition est une partie d'un type et non un type entier) en théorie des types de Martin-Löf, dans le Calcul des constructions ou dans le Calcul des constructions inductives.

Ici, puisque nous sommes dans un cadre non typé, rien n'empêche de former le terme $\alpha(f, a, p)$ où a est un objet qui n'est pas dans le domaine de définition de f et p est un terme quelconque. Mais dans ce cas, la vérification que p est une démonstration que a est dans l'ensemble de définition de f échoue et les règles de réduction ne s'appliquent pas.

Une seconde motivation pour construire une théorie où les démonstrations sont des objets à part entière est, comme nous l'avons déjà évoqué au paragraphe 4.6, de donner des règles de calcul pour l'opérateur de descriptions.

6.3.2 Une formulation sans types des mathématiques où les démonstrations sont des objets

Plusieurs formalisations des mathématiques dans lesquelles les démonstrations sont des objets ont été proposées, parmi lesquelles la théorie des types de Martin-Löf, le Calcul des constructions et le Calcul des constructions inductives. Toutes ces théories sont des théories typées. Cela s'explique par le fait que leurs auteurs ont toujours cherché un formalisme dans lequel la β-réduction termine et par le fait que l'isomorphisme de Curry-De Bruijn-Howard entre propositions et types demande un langage typé.

Nous défendons le point de vue qu'il n'est pas nécessaire d'interdire l'application d'une fonction à un argument qui est hors de son domaine de définition par des règles syntaxiques, mais qu'il est suffisant d'interdire la β-réduction quand l'argument d'une fonction n'est pas dans cet ensemble. (Simplement parce que la règle de β-réduction est fausse dans ce cas, si f est la fonction identité définie sur l'ensemble des nombres pairs $f = x \in P \longmapsto x$, alors $f(1)$ n'est pas nécessairement égal à 1. La valeur de ce terme n'est pas spécifiée par les axiomes.) En suivant ce point de vue, seule la terminaison de la β-réduction gardée importe.

Par ailleurs, l'isomorphisme de Curry-De Bruijn-Howard nous paraît un outil utile en théorie de la démonstration (c'est-à-dire quand on cherche à construire un langage $\mathcal{L}'$ pour exprimer les démonstrations d'une théorie fondée sur un langage $\mathcal{L}$, sans confusion entre ces langages (voir, par exemple, le chapitre 4)) mais pas nécessairement quand on cherche à construire une théorie fondée sur un langage $\mathcal{L}$ telle que les démonstrations de cette théorie soient exprimables dans ce même langage $\mathcal{L}$.

Dans [f] nous avons proposé une première tentative de langage non typé dans lequel les démonstrations sont des objets. Pour cela, nous avons introduit un symbole pr tel que si P est un contenu propositionnel, alors $pr(P)$ est l'ensemble des démonstrations de la proposition $\varepsilon(P)$. La sémantique de Brouwer-Heyting-Kolmogorov s'exprime alors par certains axiomes de la théorie, par exemple

$$\varepsilon(pr(P \dot{\Rightarrow} Q) = pr(P) \rightarrow pr(Q))$$

On retrouve ici plusieurs outils que nous avons introduits dans les versions multisortée et monosortée de la théorie des types simples : la distinction entre proposition et contenus propositionnels, la notion primitive de fonction, le constructeur d'espace fonctionnel $\rightarrow$, ...

Les propositions $p \in pr(P)$ ne sont pas toujours décidables, mais on donne dans [f] un algorithme qui permet d'en décider certaines : on montre qu'à chaque fois que la proposition $\varepsilon(P)$ est démontrable, il existe un terme p tel que la proposition $p \in pr(P)$ soit non seulement démontrable, mais de plus reconnue par l'algorithme. On appelle ces démonstrations reconnues par l'algorithme *démonstrations directes*. On montre aussi que si p une démonstration indirecte de P, et que q est une démonstration directe de $p \in pr(P)$, alors on peut construire à partir de p et q une démonstration directe de P.

6.3.3 Le paradoxe de Tarski

On montre enfin qu'on peut poser l'axiome "vérité = démontrabilité"

$$\varepsilon(P \dot{\Leftrightarrow} \dot{\exists} x \in pr(P))$$

sans qu'on puisse pour autant appliquer le paradoxe de Tarski pour montrer l'incohérence de cette théorie. En effet, réfléchir les propositions comme des termes (c'est-à-dire avoir une fonction "." qui associe un terme à chaque proposition), un prédicat de vérité $\mathcal{T}$ et un schéma

$$P \Leftrightarrow \mathcal{T}(\text{“}P\text{”})$$

n'est pas contradictoire en soi. Si Γ est une théorie qui a un modèle $\mathcal{M}$ de plus de deux éléments, on peut montrer facilement qu'ajouter ce schéma à Γ n'est pas contradictoire, il suffit d'appeler 0 et 1 deux éléments distincts de $\mathcal{M}$, de prendre 1 comme dénotation de "P" si la proposition P est valide dans $\mathcal{M}$ et 0 sinon, et d'interpréter $\mathcal{T}$ par l'ensemble $\{1\}$.

La contradiction, dans le paradoxe de Tarski, vient de l'existence d'un prédicat de substitution, c'est-à-dire d'un prédicat R tel qu'on puisse démontrer la proposition

$$\forall x \ \forall y \ \exists z \ R(x, y, z)$$

et pour toute proposition P et terme t, la proposition (schéma de substitution)

$$R(\text{“}P\text{”}, t, q) \Leftrightarrow q = \text{“}P[t/x]\text{”}$$

Le théorème de Tarski montre que dès qu'on a un prédicat de vérité, ce schéma de substitution est une forme du schéma de compréhension de la théorie naïve des ensembles. En effet, en skolémisant la proposition

$$\forall x \ \forall y \ \exists z \ R(x, y, z)$$

on obtient

$$\forall x \ \forall y \ R(x, y, \alpha(x, y))$$

et donc pour toute proposition P et terme t, on a

$$R(\text{“}P\text{”}, t, \alpha(\text{“}P\text{”}, t))$$

soit

$$\alpha(\text{“}P\text{”}, t) = \text{“}P[t/x]\text{”}$$

En utilisant les axiomes de l'égalité, on obtient

$$\mathcal{T}(\alpha(\text{“}P\text{”}, t)) \Leftrightarrow \mathcal{T}(\text{“}P[t/x]\text{”})$$

et comme $\mathcal{T}$ est un prédicat de vérité

$$\mathcal{T}(\alpha(\text{“}P\text{”}, t)) \Leftrightarrow P[t/x]$$

En écrivant $a \in b$ pour $\mathcal{T}(\alpha(b,a))$ et $\{x \mid P\}$ pour "P", cela se lit

$$t \in \{x \mid P\} \Leftrightarrow P[t/x]$$

qui est le schéma de compréhension. Le paradoxe de Russell donne alors une contradiction.

Si on réfléchit une proposition comme son contenu propositionnel, on n'a naturellement pas le schéma de substitution. Pour obtenir ce schéma, il faudrait par exemple, réfléchir une proposition P par le terme λx_1 ... λx_n p où p est le contenu propositionnel de P et $x_1, ..., x_n$ sont ses variables libres. Cela demanderait de construire une fonctions définie sur l'univers entier, ce qui est déjà, en soi, contradictoire.

Pour employer une autre terminologie, associer un terme à chaque proposition est un plongement du langage dans lui-même, le paradoxe de Tarski montre qu'un plongement profond [14] est contradictoire, mais un plongement superficiel ne l'est pas nécessairement.

6.3.4 Le problème de la cohérence

Cependant, le problème de la cohérence de la théorie proposée dans [f] est laissé ouvert. Pour construire un modèle, la première idée est d'interpréter les contenus propositionnels par l'ensemble de leurs démonstrations (et corrélativement le symbole *pr* par l'identité). L'ensemble O des contenus propositionnels s'interprète alors comme un ensemble d'ensembles de démonstrations.

Le problème est qu'il est possible de quantifier sur les contenus propositionnels, pour former un contenu propositionnel

$$\dot{\forall} p \in O \ (p \dot{\Rightarrow} p)$$

l'ensemble $pr(\dot{\forall} p \in O \ (p \dot{\Rightarrow} p))$ contient une fonction f dont le domaine de définition est l'ensemble $\hat{O}$, interprétation de O.

Cet objet $\hat{O}$ contient donc un élément qui contient un élément qui est une fonction dont le domaine de définition est l'ensemble $\hat{O}$ lui-même. La construction d'un tel objet imprédicatif est la principale difficulté qui m'a arrêtée. Mais, il est vraisemblable que les techniques utilisées pour construire des modèles du λ-calcul polymorphe puissent s'appliquer ici.

6.4 Une restriction de la théorie des ensembles

Un projet moins ambitieux consiste à chercher une axiomatisation des mathématiques non typée et dont le système de réécriture termine mais en abandonnant l'objectif d'avoir des fonctions comme des objets primitifs.

Nous avons vu qu'on ne peut pas définir une fonction sur l'univers entier, mais qu'il faut lui donner un domaine de définition. Similairement, en théorie des ensembles, on ne peut pas définir un ensemble en compréhension sur l'univers entier, mais il faut donner un ensemble englobant qu'on peut assimiler à un domaine de définition de l'ensemble.

Cependant, les notions de fonction et d'ensemble ont une différence essentielle : si f est une fonction de domaine de définition A et que t n'est pas un objet de A alors le terme $f(t)$ désigne un objet non spécifié. En revanche si B est un ensemble, d'ensemble englobant A ($B = \{x \in A \mid P\}$), et t un objet hors de A, alors la valeur de vérité de la proposition $t \in B$ est très bien spécifiée : cette proposition est fausse. De ce fait, au lieu d'avoir une règle de réécriture conditionnelle

$$\in (a, \{x \in A \mid P\}, p) \to [a/x]P \text{ si } p \text{ est une démonstration directe de } a \in A$$

on a, en théorie des ensembles, une règle

$$\in (a, \{x \in A \mid P\}) \to a \in A \wedge [a/x]P$$

C'est la raison principale de la non terminaison du système de réécriture de la théorie des ensembles, et de ce fait de l'absence de l'élimination des coupures.

On peut s'interroger sur l'existence de restrictions de la théorie des ensembles pour lesquelles le système de réécriture termine et la propriété d'élimination des coupures est vérifiée. Une solution est sans doute de restreindre le schéma de compréhension aux propositions P stratifiables au sens de Quine. On aurait alors un système intermédiaire entre NF et Z. On peut remarquer que ce système n'est pas typé, comme d'usage en théorie des ensembles, il restreint la formation des objets en agissant sur les axiomes et non sur la syntaxe du langage, et il est sans doute suffisant pour exprimer une bonne partie des mathématiques informelles. Un résultat qui semble indiquer que ce programme est réalisable est la démonstration par S.C. Bailin [8] de l'élimination des coupures pour une variante de la théorie des ensembles dans laquelle une règle de déduction

$$\frac{t \in t}{\perp}$$

est ajoutée, ainsi qu'un mécanisme de priorité qui interdit d'appliquer une règle autre que celle-ci à une proposition de cette forme.

Conclusion

La principale contribution de ce dossier d'habilitation est d'avoir proposé un ensemble de règles de déduction qui permet d'intégrer le calcul et le raisonnement. Certes, cette idée n'est pas nouvelle : Plotkin et Andrews l'avaient déjà implicitement proposé dans le cadre de la démonstration automatique, le langage du système Automath, la théorie des types de Martin-Löf, le Calcul des constructions et le Calcul des constructions inductives comportaient déjà une règle de conversion, l'Arithmétique fonctionnelle du second ordre permettait déjà d'utiliser les axiomes équationnels sans laisser de trace dans la démonstration. Mais nous soutenons que cette distinction qui est centrale en mathématique doit être étudiée dans le cadre le plus général possible : la logique du premier ordre. Les applications en démonstration automatique et en λ-calcul s'en déduisent alors naturellement.

Par ailleurs, nous avons insisté sur la possibilité d'utiliser des règles de calcul qui transforment des propositions atomiques en des propositions non atomiques. C'est dans ce cadre qu'on peut exprimer des systèmes puissants comme la théorie des types simples et la théorie des ensembles. Ce sont ces règles de calcul qui sont responsables de la difficulté de démontrer l'élimination des coupures et ce sont elles qui, en démonstration automatique, rendent nécessaire la règle de surréduction étendue.

Cette habilitation propose deux autres contributions plus techniques qui consistent à replacer le théorème d'élimination des coupures et les algorithmes de démonstration automatique développés pour la théorie des types simples dans le cadre plus général de la déduction modulo. Nous défendons le point de vue que les outils qui ont été développés dans ces deux cas ne sont en rien propres à cette théorie, et qu'ils ont une portée beaucoup plus générale.

Contrairement à ce que nous avions fait dans les articles qui constituent ce dossier, nous avons accordé une large place, dans ce mémoire, à des conjectures et à la discussion de problèmes encore mal dégrossis. On peut donc à présent tracer un programme de travail pour les années à venir : en premier lieu, nous avons conjecturé qu'on pouvait toujours construire un prémodèle pour une congruence présentée par un système de réécriture confluent et qui termine. Si nous croyons cette conjecture vraie, parce que nous n'avons pas réussi à construire de contre-exemple, nous pouvons cependant nous demander si elle peut être démontrée en théorie des ensembles, ou si, au contraire, elle implique la cohérence de cette théorie. Nous avons également uniquement esquissé les applications de la notion de déduction modulo dans le cadre du λ-calcul typé avec des types dépendants. Nous laissons également ouvert le problème de trouver une axiomatisation des mathématiques non typée dont le système de réécriture termine et dans laquelle les fonctions sont des objets primitifs. Si la notion de β-réduction gardée nous semble ici essentielle, il est probable que cela demande de développer une théorie non typée dans laquelle les démonstrations sont des objets, et nous avons évoqué les difficultés, notamment de cohérence, que cela pose.

Enfin, nous voudrions conclure par une perspective qui concerne la déduction modulo elle-même. Dans le système de déduction naturelle modulo que nous avons présenté au chapitre 1, les règles de déduction peuvent modifier les axiomes de la théorie (c'est, par exemple, le cas de la règle d'introduction de l'implication) mais jamais la congruence. De ce fait, la congruence est fixée une fois pour toutes au début de la démonstration et ne peut pas s'enrichir. On peut imaginer un cadre plus général où il serait possible de transformer dynamiquement un théorème en une règle de réécriture. Il faudrait alors donner à ce moment un métaargument qui assurerait que le système de réécriture reste confluent et qu'il continue à terminer. Éventuellement, cela peut demander d'utiliser un algorithme de complétion pour compléter le système de

réécriture enrichi.

Cela nous semble particulièrement important pour les démonstrations par récurrence où on veut souvent transformer l'hypothèse de récurrence en une règle de réécriture pour résoudre la conclusion.

Une telle possibilité correspondrait également davantage à la pratique mathématique : au sein des mathématiques, nous développons des algorithmes, nous montrons leur terminaison puis nous les incorporons peu à peu à nos automatismes de calcul qui, ainsi, s'enrichissent sans cesse.

Bibliographie

[a] G. Dowek, Lambda-calculus, combinators and the comprehension scheme, M. Dezani-Ciancaglini and G. Plotkin (Eds.), *Typed lambda calculi and applications*, Lecture notes in computer science 902, Springer-Verlag (1995), pp. 154-170. *Rapport de Recherche* 2565, INRIA (1995).

[b] G. Dowek, Collections, sets and types, *Mathematical structures in computer science*, 9 (1999), pp. 1-15.

[c] G. Dowek, Th. Hardin, and C. Kirchner, Higher-order unification via explicit substitutions, *Logic in computer science* (1995), pp. 366-374. *Information and Computation* (to appear).

[d] G. Dowek, Th. Hardin, C. Kirchner, and F. Pfenning, Higher-order unification via explicit substitutions : the case of higher-order patterns, M. Maher (Ed.), *Joint International Conference and Symposium on Logic Programming* (1996), pp. 259-273. *Rapport de Recherche* 3591, INRIA (1998).

[e] G. Dowek, Proof normalization for a first-order formulation of higher-order logic, E.L. Gunter and A. Felty (Eds.), *Theorem Proving in Higher-order Logics*, Lecture notes in computer science 1275, Springer-Verlag (1997), pp. 105-119. *Rapport de Recherche* 3383, INRIA (1998).

[f] G. Dowek, A Type-free formalization of mathematics where proofs are objects, E. Giménez and Ch. Paulin-Mohring (Eds.), *Types for proofs and programs*, Lecture notes in computer science 1512, Springer-Verlag (1998), pp. 88-111. *Rapport de Recherche* 2915, INRIA (1996).

[g] G. Dowek, Th. Hardin, and C. Kirchner, Theorem proving modulo, *Rapport de Recherche* 3400, INRIA (1998).

[h] G. Dowek and B. Werner, Proof normalization modulo, *Types for proofs and programs* (1998) (to appear). *Rapport de Recherche* 3542, INRIA (1998).

[i] G. Dowek, Th. Hardin, and C. Kirchner, HOL-$\lambda\sigma$: an intentional first-order expression of higher-order logic. *Rewriting Techniques and Applications* (1999) (to appear). *Rapport de Recherche* 3556, INRIA (1998).

[j] G. Dowek, Automated theorem proving in first-order logic modulo : on the difference between type theory and set theory. *First-order theorem proving* (1998) (to appear).

[x] G. Dowek, Le sens du calcul, manuscript.

[y] G. Dowek, Le langage mathématique et les langages de programmation, manuscript.

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy, Explicit substitutions, *Journal of Functional Programming*, 4, 1 (1991), pp. 375-416.

[2] T. Altenkirch, Contructions, inductive types and strong normalization, *Ph.D.*, University of Edinburgh (1993).

[3] P.B. Andrews, Resolution in type theory, *The Journal of Symbolic Logic*, 36 (1971), pp. 414-432.

[4] P.B. Andrews, General models, descriptions and choice in type theory, *The Journal of Symbolic Logic*, 37, 2 (1972), pp. 385-394.

[5] P.B. Andrews, General models and extensionality, *The Journal of Symbolic Logic*, 37, 2 (1972), pp. 395-397.

[6] P.B. Andrews, An introduction to mathematical logic and type theory : to truth through proof, *Academic Press* (1986).

[7] P.B. Andrews, D.A. Miller, E. Longini Cohen, and F. Pfenning, Automating higher-order logic, W.W. Bledsoe and D.W. Loveland (Eds.), *Automated theorem proving : after 25 years*, Contemporary Mathematics Series 29, American Mathematical Society (1984), pp. 169-192.

[8] S.C. Bailin, A normalization theorem for set theory, *The Journal of Symbolic Logic*, 53, 3 (1988), pp. 673-695.

[9] S.C. Bailin, A λ-unifiability test for set theory, *Journal of Automated Reasoning*, 4 (1988), pp. 269-286.

[10] H.P. Barendregt, The lambda-calculus, its syntax and semantics, second edition, *North-Holland* (1984).

[11] H.P. Barendregt, Lambda calculi with types, S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum (Eds.), *Handbook of Logic in Computer Science*, Vol. 2, Clarendon Press (1992), pp. 118-309.

[12] H. Barendregt and E. Barendsen, Autarkic computations in formal proofs, manuscript.

[13] P. Borovanský, Implementation of higher-order unification based on calculus of explicit substitution, M. Bartošek, J. Staudek, and J. Wiedermann (Eds.) *SOFSEM'95 : Theory and Practice of Informatics*, Lecture notes in computer science 1012, Springer-Verlag (1995), pp. 363-368.

[14] R. Boulton, A. Gordon, M. Gordon, J. Harrison, J. Herbert and J. Van Tassel, Experience with embedding hardware description languages in HOL, V. Stavridou, T.F. Melham and R. T. Boute (Eds.), *IFIP TC10/WG 10.2, International Conference on Theorem Provers in Circuit Design : Theory, Practice and Experience*, IFIP Transactions volume A-10, North-Holland/Elsevier (1992), pp. 129-156.

[15] Th. Coquand and G. Huet, The calculus of constructions, *Information and Computation*, 76 (1988), pp. 95-120.

[16] A. Church, A formulation of the simple theory of types, *The Journal of Symbolic Logic*, 5 (1940), pp. 56-68.

[17] A. Church, The calculi of lambda-conversion (1941).

[18] A. Church, Introduction to mathematical logic, *Princeton University Press* (1956).

[19] P.-L. Curien, Categorical combinators, sequential algorithms and functional programming, second edition, *Birkhäuser* (1993).

[20] P.-L. Curien and A. Rios, Un résultat de complétude pour les substitutions explicites, *Compte-rendus à l'Académie des Sciences de Paris*, I, 312 (1991), pp. 471–476.

[21] P.-L. Curien, Th. Hardin, and J.-J. Lévy, Confluence properties of weak and strong calculi of explicit substitutions, *Journal of the Association for Computing Machinery*, 43, 2 (1996), pp. 362-397.

[22] H.B. Curry, The combinatory foundations of mathematical logic, *The Journal of Symbolic Logic*, 7, 2 (1942), pp. 49-64.

[23] H.B. Curry and R. Feys, Combinatory logic, Vol. 1, *North-Holland* (1958).

[24] M. Davis, Invited commentary to [85], *International Federation for Information Processing Congress*, North-Holland (1968), pp. 67-68.

[25] N.G. De Bruijn, Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem, *Indagationes Mathematicae*, 5, 34 (1972), pp. 381-392.

[26] N.G. De Bruijn, A Survey of the project Automath, *To H.B. Curry : essays on combinatory logic, lambda calculus and formalism*, Academic Press (1980) pp. 579-606.

[27] D.J. Dougherty, Higher-order unification via combinators, *Theoretical Computer Science*, 114 (1993), pp. 273-298.

[28] J. Ekman, Normal proofs in set theory, *Doctoral thesis*, Chalmers University of Technology and University of Göteborg (1994).

[29] H.B. Enderton, A mathematical introduction to logic, *Academic Press* (1972).

[30] M. Fay, First-order unification in an equational theory, *Fourth Workshop on Automated Deduction* (1979), pp. 161-167.

[31] S. Feferman, Finitary inductively presented logics, *Logic Colloquium*, R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo (Eds.), North-Holland (1989).

[32] S. Feferman, Reflecting on incompleteness, *The Journal of Symbolic Logic*, 56, 1, (1991), pp. 1-49.

[33] J. Gallier, Logic in computer science, *Harper and Row* (1986).

[34] J. Gallier, On Girard's *candidats de réductibilité*, P. Odifreddi (Ed.), *Logic and Computer Science*, Academic Press (1990), pp. 123-203.

[35] J. Gallier and W. Snyder, Complete set of transformations for general E-unification, *Theoretical Computer Science*, 67, 2-3 (1989), pp. 203-260.

[36] J.-Y. Girard, Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types, J.E. Fenstad (Ed.), *Second Scandinavian Logic Symposium*, North-Holland (1970).

[37] J.-Y. Girard, Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur, *Thèse d'État*, Université de Paris VII (1972).

[38] J.-Y. Girard, Y. Lafont, and P. Taylor, Proofs and types, *Cambridge University Press* (1989).

[39] P. Gochet et P. Gribomont, Logique, *Hermès* (1990, 1994).

[40] H. Goguen and J. Goubault-Larrecq, Sequent combinators : a Hilbert system for the lambda calculus, manuscript.

[41] J. Goubault-Larrecq, A proof of weak termination of the simply typed $\lambda\sigma$-calculus, *Rapport de Recherche* 3090, INRIA (1997).

[42] L. Hallnäs, On normalization of proofs in set theory, *Doctoral thesis*, University of Stockholm (1983).

[43] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40, 1 (1993), pp. 143-184.

[44] Th. Hardin, L. Maranget, and B. Pagano, Functional back-ends within the lambda-sigma calculus, *Journal of Functional Programming*, 8, 2 (1998), pp. 131-176.

[45] L. Henkin, Completeness in the theory of types, *The Journal of Symbolic Logic*, 15 (1950), pp. 81-91.

[46] L. Henkin, Banishing the rule of substitution for functional variables, *The Journal of Symbolic Logic*, 18, 3 (1953), pp. 201-208.

[47] J.R. Hindley, Combinatory reductions and lambda reductions compared, *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 23 (1977), pp. 169-180.

[48] J.R. Hindley and J.P. Seldin, Introduction to combinators and λ-calculus, *Cambridge University Press* (1986).

[49] W.A. Howard, The Formulæ-as-type notion of construction, 1969, J.R. Hindley and J.P. Seldin (Eds.), *To H.B. Curry : essays on combinatory logic, lambda calculus and formalism*, Academic Press (1980) pp. 479-490.

[50] G. Huet, Constrained resolution : a complete method for higher order logic, *Ph.D.*, Case Western Reserve University (1972).

[51] G. Huet, A mechanization of type theory, *International Joint Conference on Artificial Intelligence* (1973), pp. 139-146.

[52] G. Huet, A unification algorithm for typed lambda calculus, *Theoretical Computer Science*, 1, 1 (1975), pp. 27–57.

[53] G. Huet, Résolution d'équations dans les Langages d'Ordre 1,2, ..., ω, *Thèse d'État*, Université de Paris VII (1976).

[54] R.J.M. Hughes, Super-combinators, a new implementation method for applicative languages, *Lisp and Functional Programming* (1982), pp. 1-20.

[55] J.-M. Hullot, Canonical forms and unification, W. Bibel and R. Kowalski (Eds.) *Conference on Automated Deduction*, Lecture notes in computer science 87, Springer-Verlag (1980), pp. 318-334.

[56] J. Hsiang, Refutational theorem proving using term-rewriting systems, *Artificial Intelligence*, 25 (1985), pp. 255-300.

[57] J.-P. Jouannaud and C. Kirchner, Solving equations in abstract algebras : a rule-based survey of unification, J.-L. Lassez and G. Plotkin (Eds.) *Computational logic. Essays in honor of Alan Robinson*, MIT press (1991), pp. 257–321.

[58] J.W. Klop, V. van Oostrom, and F. van Raamsdonk, Combinatory reduction systems : introduction and survey, *Theoretical Computer Science*, 121 (1993), pp. 279-308.

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

[60] J.L. Krivine and M. Parigot, Programming with proofs, *J. Inf. Process. Cybern. EIK* 26 (1990), pp. 149-167.

[61] A. Lalande, entrée *lexis*, Vocabulaire technique et critique de la philosophie, *Presses Universitaires de France* (1951).

[62] L. Magnusson, The implementation of ALF, a proof editor based on Martin-Löf monomorphic type theory with explicit substitution, *Doctoral thesis*, Chalmers University of Technology and University of Göteborg (1994).

[63] L. Maranget, Optimal derivations in weak lambda-calculi and in orthogonal term rewriting systems, *Principle of Programming Languages* (1991), pp. 255-269.

[64] P. Martin-Löf, Constructive mathematics and computer programming, *Logic, Methodology and Philosophy of Science*, L.J. Cohen, J. Łoś, H. Pfeiffer, and K.-P. Podewski (Eds.), North-Holland (1982), pp. 153-175.

[65] P. Martin-Löf, Intuitionistic type theory, *Bibliopolis* (1984).

[66] P. Martin-Löf, Personal communication.

[67] V. McGee, How truthlike can a predicate be ? A negative result, *Journal of Philosophical Logic*, 14 (1985), pp. 399-410.

[68] P.-A. Melliès, Typed λ-calculi with explicit substitutions may not terminate, M. Dezani-Ciancaglini and G. Plotkin (Eds.), *Typed Lambda Calculi and Applications* (1995).

[69] E. Mendelson, Introduction to mathematical logic, *Van Nostrand* (1963).

[70] D.A. Miller, Proofs in higher order logic, *Ph.D.*, Carnegie Mellon University (1983).

[71] D.A. Miller, A compact representation of proofs, *Studia Logica*, 46, 4 (1987), pp. 347-370.

[72] D.A. Miller, Unification under a mixed prefix, *Journal of Symbolic Computation*, 4, 14 (1992), pp. 321-358.

[73] D.A. Miller, A logic programming language with lambda-abstraction, function variables, and simple unification, *Journal of Logic and Computation*, 1, 4 (1991), pp. 497-536.

[74] C.A. Muñoz-Hurtado, Un calcul de substitutions pour la représentation des preuves partielles en théorie des types, *Thèse de Doctorat*, Université de Paris VII (1997).

[75] C.A. Muñoz-Hurtado, A calculus of substitutions for incomplete-proof representation in type theory, *Rapport de Recherche* 3309, INRIA (1997).

[76] C.A. Muñoz-Hurtado, A left-linear variant of $\lambda\sigma$, *International Conference PLILP/ALP/HOA* 1997, Lecture notes in computer science 1298, Springer-Verlag (1997) pp. 224-239.

[77] C.A. Muñoz-Hurtado, Dependent types with explicit substitutions : a meta-theoretical development, E. Giménez and Ch. Paulin-Mohring (Eds.), *Types for proofs and programs*, Lecture notes in computer science 1512, Springer-Verlag (1998), pp. 294-316.

[78] Ch. Paulin-Mohring, Inductive definitions in the system COQ, Rules and properties, *Typed Lambda Calculi and Applications*, Lecture notes in computer science 664, Springer-Verlag (1993), pp. 328-345.

[79] Ch. Paulin-Mohring and B. Werner, Synthesis of ML programs in the system Coq. *Journal of Symbolic Computation*, 15 (1993), pp. 607-640.

[80] G. Plotkin, Building-in equational theories, *Machine Intelligence*, 7 (1972), pp. 73-90.

[81] H. Poincaré, La science et l'hypothèse, 1902, Flammarion (1968).

[82] W.V.O. Quine, Set theory and its logic, *Belknap press* (1969).

[83] A. Ríos, Contributions à l'étude des λ-calculs avec des substitutions explicites, *Thèse de Doctorat*, Université de Paris VII (1993),

[84] J. A. Robinson, A Machine-oriented logic based on the resolution principle, *Journal of the Association for Computing Machinery*, 12, 1 (1965), pp. 23-41.

[85] J.A. Robinson, New directions in mechanical theorem proving, *International Federation for Information Processing Congress*, North-Holland (1968), pp. 63-67.

[86] W. Snyder and J. Gallier, Higher-order unification revisited : complete sets of transformations, *Journal of Symbolic Computation*, 8 (1989), pp. 101-140.

[87] M. Stickel, Automated deduction by theory resolution, *Journal of Automated Reasoning*, 4, 1 (1985), pp. 285-289.

[88] T. Strahm, Partial applicative theories and explicit substitutions, *Journal of Logic and Computation* 6, 1 (1996), pp. 55-77.

[89] W.W. Tait, Intensional interpretation of functionals of finite type I, *The Journal of Symbolic Logic*, 32, 2 (1967), pp. 198-212.

[90] A. Werner, Normalizing narrowing for weakly terminating and confluent systems, U. Montanari and F. Rossi, (Eds.), *First International Conference on Principles and Practice of Constraint Programming*, Lecture notes in computer science 976, Springer-Verlag (1995), pp. 415-430.

[91] B. Werner, Une théorie des constructions inductives, *Thèse de Doctorat*, Université de Paris VII (1994).

[92] A.N. Whitehead and B. Russell, Principia mathematica, *Cambridge University Press* (1910-1913, 1925-1927).

Table des matières

1 La déduction modulo **11**
1.1 Les règles 11
1.2 Les congruences 14
1.2.1 La réécriture des termes 14
1.2.2 Les équations entre termes 14
1.2.3 La réécriture des propositions 14
1.3 Le lemme d'équivalence 15
1.4 La notion de modèle pour la déduction modulo 15
1.5 Le cas des congruences présentées par un système de réécriture 16

2 La théorie des types simples **19**
2.1 La théorie naïve des ensembles 19
2.2 La théorie des types simples 20
2.3 Le symbole ε 22
2.3.1 Les contenus propositionnels 22
2.3.2 Les termes, les propositions et les assertions 23
2.4 Le langage des termes et l'intentionnalité 24
2.4.1 La substitution des variables de fonction et de prédicat 24
2.4.2 Le schéma de compréhension 24
2.4.3 Les combinateurs 24
2.4.4 Le λ-calcul 26
2.4.5 Le calcul des substitutions explicites 27
2.4.6 Les combinateurs, à nouveau 31
2.4.7 Le calcul faible des substitutions explicites 34
2.4.8 En conclusion : quel langage ? 35
2.5 La complétude 35
2.6 La skolémisation 35
2.6.1 Le problème de la skolémisation en théorie des types 35
2.6.2 La skolémisation dans la présentation de la théorie des types utilisant les combinateurs 37
2.6.3 La skolémisation dans la présentation de la théorie des types utilisant des substitutions explicites 37

3 La théorie des ensembles **39**
3.1 La théorie des ensembles de Zermelo 39
3.2 Le langage des termes 40

4 L'élimination des coupures **43**
4.1 Les coupures en déduction modulo . . . 43
4.2 La notation fonctionnelle des démonstrations . . . 44
4.3 La réductibilité . . . 46
4.3.1 L'élimination des coupures en logique du premier ordre . . . 46
4.3.2 L'élimination des coupures en déduction modulo . . . 47
4.3.3 Les prémodèles . . . 47
4.3.4 La conjecture de normalisation . . . 49
4.3.5 Des prémodèles aux modèles . . . 49
4.4 Les coupures de conversion . . . 50
4.5 La normalisation des démonstrations du théorème de Cantor . . . 50
4.5.1 En théorie des types avec une fonction . . . 50
4.5.2 En théorie des types avec une relation . . . 51
4.5.3 En théorie des ensembles . . . 52
4.5.4 Dans les théories étendues . . . 53
4.6 Les démonstrations objets de la théorie . . . 54
4.6.1 Les motivations . . . 54
4.6.2 Le plongement des théories du premier ordre dans le λ-calcul . . . 54
4.6.3 Le plongement des théories modulo dans le λ-calcul . . . 55
4.6.4 Le cas de la théorie des types simples . . . 55

5 La démonstration automatique **59**
5.1 La résolution équationnelle . . . 59
5.2 La résolution modulo . . . 60
5.2.1 La règle de surréduction étendue . . . 60
5.2.2 Les contraintes . . . 61
5.2.3 La complétude . . . 61
5.2.4 Un exemple simple . . . 61
5.3 La démonstration automatique en théorie des types simples . . . 63
5.3.1 La surréduction et la scission . . . 63
5.3.2 L'unification équationnelle et l'unification d'ordre supérieur . . . 64
5.3.3 De la résolution d'ordre supérieur à la résolution modulo . . . 65
5.4 La démonstration automatique en théorie des ensembles . . . 65

6 Les types et la terminaison **67**
6.1 Les types et les ensembles . . . 67
6.2 La réduction dans un cadre monosorté . . . 68
6.3 Les démonstrations objets de la théorie . . . 69
6.3.1 Les motivations . . . 69
6.3.2 Une formulation sans types des mathématiques où les démonstrations sont des objets 69
6.3.3 Le paradoxe de Tarski . . . 70
6.3.4 Le problème de la cohérence . . . 71
6.4 Une restriction de la théorie des ensembles . . . 71

Index

ascension des variables (λ-*lifting*), 26
axiome
  de conversion, 21, 25
  de Curry, 27

β-réduction, 26

calcul, 7
  des séquents, 11
candidat de réductibilité, 46
combinateur, 24
  catégorique, 27
congruence, 11
conjecture de normalisation, 49
contenu propositionnel, 20
contrainte, 61
coupure, 43
  de conversion, 50
cube de Barendregt, 57

décidabilité de la congruence, 14
déduction
  modulo, 11
  naturelle, 11
démonstration automatique, 8, 59

élimination des coupures, 8, 43
ε, 20
équation, 14
η-réduction, 31

fonction partielle, 69
Fondations nouvelles (*New foundations*), 68

imprédicativité, 71
indice
  de De Bruijn, 27
  de Skolem, 36
isomorphisme de Curry-De Bruijn-Howard, 44, 69

λ-calcul, 26
  avec des constructeurs de types, 49, 57
  avec des substitutions explicites, 27
  avec des types dépendants, 54
  avec des types dépendants modulo, 54
  avec des types inductifs, 57
  faible avec des substitutions explicites, 34
  polymorphe, 49, 57
$\lambda\sigma$-calcul, 27
lemme
  d'équivalence, 15
  de conservativité, 16
lexis, 23

méthode de Tait, 46
modèle, 15, 49

opérateur de descriptions, 54, 69

paradoxe
  de Russell, 19
  de Tarski, 23, 70
plongement
  d'une théorie dans le λ-calcul, 54
  d'une théorie modulo dans le λ-calcul, 55
précuisson (*pre-cooking*), 30, 64
prémodèle, 48
programmation en langage mathématique, 8, 54
proposition de Crabbé, 40, 44, 65

raisonnement, 7
réductible, 46
règle
  de réduction des démonstrations, 44
  de réécriture, 14
    de la théorie des ensembles, 40
    de la théorie des types simples, 22
  de typage des démonstrations, 45
résolution
  équationnelle, 59
  modulo, 60

schéma de compréhension, 19, 24
  fermé, 25
  ouvert, 25
scission, 60

sémantique de Brouwer-Heyting-Kolmogorov, 44, 69
skolémisation
 de la théorie des ensembles, 39
 de la théorie des types, 21
 de la théorie naïve des ensembles, 19
 en théorie des types, 35
stratifiable, 49, 68, 72
substitution
 des indices de De Bruijn, 28
 des variables de fonction et de prédicat, 24
 du λ-calcul, 26
surréduction, 59
 étendue, 60
système de réduction combinatoire (*combinatory reduction systems*), 15

théorème
 d'incomplétude (second), 49
 de Cantor, 50
 de complétude, 16, 35
 de complétude de la résolution modulo, 61
 de correction, 16
 de l'effondrement, 35
 de Tait, 22
théorie
 des ensembles, 39
 des ensembles de Von Neumann-Bernays-Gödel, 40
 des types simples, 20
 naïve des ensembles, 19

unification
 d'ordre supérieur, 64
 équationnelle, 59

valeur, 9
vérification de correction des démonstrations, 8