Programmation en lambda-calcul pur et typé

Karim Nour

UNIVERSITÉ DE SAVOIE

SYNTHÈSE D'ACTIVITÉS SCIENTIFIQUES

Spécialité : Mathématiques et Informatique Théorique

PROGRAMMATION EN λ-CALCUL PUR ET TYPÉ

présentée par

Karim NOUR

pour obtenir le titre de

HABILITATION À DIRIGER DES RECHERCHES

soutenue le 14 janvier 2000 devant le jury composé de

J.L. Krivine	**(Professeur à l'Université Paris 7)**
S. Ronchi	**(Professeur à Università Degli Studi Di Totino)**
T. Ehrahard	**(Chargé de Recherche au C.N.RS. de Marseille)**
J.Y. Girard	**(Directeur de Recherche au C.N.RS. de Marseille)**
C. Paulin	**(Professeur à l'Université Paris Sud-Orsay)**
R. David	**(Professeur à l'Université de Savoie)**

Gloire à Toi ! Nous n'avons de savoir que ce que tu nous a appris. Certes c'est Toi l'Omniscient, le Sage.

Coran 2, 32

Nous élevons en rang qui Nous voulons. Et au-dessus de tout homme détenant la science, il y a un savant plus docte que lui.

Coran 12, 76

À mon père, image du sacrifice et de la volonté

À ma mère, symbole de la tendresse et du dévouement

À ma femme, pour sa patience et son aide

À mes frères et soeurs qui étaient toujours là pour m'encourager

À mes enfants Maryame et Mouhammad

À toute ma famille

À tous mes amis

Remerciements

Je voudrais tout d'abord remercier le professeur **R. David** pour l'aide et les encouragements constants qu'il m'a apportés lors de la préparation à ce diplôme.

Auteur d'un rapport sur mon travail, le professeur **J.L. Krivine** trouvera ici l'expression de ma gratitude pour l'honneur qu'il me fait d'être président du jury.

Également je tiens à remercier les professeurs **S. Ronchi** et **T. Ehrhard** qui m'ont honoré en acceptant de consacrer une partie de leurs temps pour rapporter mon travail.

Je remercie les professeurs **C. Paulin** et **J.Y. Girard** pour l'honneur qu'il m'ont fait d'accepter d'être membre du jury.

J'aurais également désiré faire figurer tous ceux qui m'ont supporté et soutenu durant les années de préparation à ce diplome. Je ne pourrais les citer tous car la liste serait trop longue. Du fond de mon cœur, je leur dis tous, merci.

Table des matières

Curriculum Vitae **11**

1 Introduction **21**

2 Notions de base en λ-calcul pur et typé **27**
2.1 Quelques résultats du λ-calcul pur 27
2.2 Le système de typage $\mathcal{AF}2$ 28

3 Opérateurs de mise en mémoire **33**
3.1 La définition 33
3.2 Le λ-calcul dirigé 35
3.3 Les S-opérateurs de mise en mémoire 38

4 Opérateurs de mise en mémoire typés **41**
4.1 Le théorème de J.-L. Krivine 41
4.2 Généralisations 42
4.2.1 Les types $\forall^+$ et les transformations de Gödel 42
4.2.2 Le système $\mathcal{TTR}$ 43
4.2.3 Le système $\mathcal{AF}2_\perp$ 46

5 Types de données **49**
5.1 Types de données complets du système $\mathcal{AF}2$ 49
5.2 Types de données syntaxiques du système $\mathcal{F}$ 53
5.3 Les I-types du système $\mathcal{F}$ 56

6 Systèmes numériques **59**
6.1 Systèmes numériques en λ-calcul pur 59
6.2 Systèmes numériques typables 60

7 Logiques classiques **63**
7.1 Le $\lambda\mathcal{C}$-calcul 63
7.2 Le $\lambda\mu$-calcul 66
7.3 Le $\lambda\mu^{++}$-calcul 69

8 Logiques mixtes **73**
8.1 Un système de typage basé sur la logique mixte 73
8.2 La logique mixte propositionnelle 75

9 Résultats divers **81**
9.1 Équivalence opérationnelle 81
9.2 Complétudes de la logique du second ordre 82

Bibliographie **84**

Curriculum Vitae

SITUATION PERSONNELLE

Nom et prénom : NOUR Karim

Date et lieu de naissance : 12 / 08 / 1968 à Al Tal (LIBAN)

Nationalité : Libanaise

Situation de famille : Marié et père de deux enfants

Adresse personnelle :
NOUR Karim
10 rue du Mâconnais
73000 Chambéry

Tél (33) 04.79.33.34.78

Situation professionnelle : Maître de conférences à l'université de Savoie
(depuis 01 / 09 / 1993)
1 er échelon des MC 1 er classe

Adresse professionnelle :
NOUR Karim
Université de Savoie
Campus scientifique
Laboratoire de Mathématiques
Équipe de Logique
EP 2067 du CNRS
73376 Le Bourget du Lac

Tél (33) 04.79.75.86.27
Fax (33) 04.79.75.87.42
Email knour@univ-savoie.fr

DIPLÔMES

1985 **Bac. 2ème partie, série Mathématiques**
Tripoli - LIBAN
(Première session / Mention Bien)

1988 **Maîtrise de Mathématiques pures**
Université Libanaise - Faculté des sciences (section III) - Tripoli - LIBAN
(Première session / Mention Très Bien / Rang 1)

Théorie de mesure et intégration - Calcul différentiel - Fonctions analytiques - Modules - Anneaux non commutatifs - Théorie de Galois - Algorithmique numérique - Calcul numérique et programmation - Mécanique analytique

1990 **D.E.A. de Mathématiques pures**
Université Claude Bernard - Lyon I
(Première session / Mention Bien)

1 UV : Algèbre non commutative
1/2 UV : Logique fondamentale
1/2 UV : Lambda calcul pur et typé
Mémoire de DEA : étude d'un article de J.-L. KRIVINE
Sous la direction de René DAVID (professeur à l'université de Savoie)

1993 **Thèse de doctorat en Mathématiques**
Université de Savoie
(Mention Très Honorable)
Directeur de recherche : René DAVID
Titre de la thèse : "Opérateurs de mise en mémoire en λ-calcul pur et typé"
Date de soutenance : 11 janvier 1993
Jury : J.L. KRIVINE, R. DAVID, H. BARENDREGT, J.-Y. GIRARD, S. RONCHI

ACTIVITÉS DE RECHERCHES

A- Exposés (séminaires, groupes de travail,...)

– **Jumelage Européen du lambda calcul typé**
Paris, Février 1991
Titre de l'exposé : *Strong storage operators*

- **Séminaire de l'université Paris VII**
  Paris, Novembre 1991
  Titre de l'exposé : *Opérateurs de mise en mémoire et types $\forall$-positifs*
- **Séminaire de l'E.N.S. Lyon**
  - Lyon, Novembre 1992
  Titre de l'exposé : *Opérateurs de mise en mémoire et logique classique*
  - Lyon, Novembre 1993
  Titre de l'exposé : *Les entiers en logique classique*
  - Lyon, Février 1995
  Titre de l'exposé : *Opérateurs de mise en mémoire et logique mixte*
- **Journées P.R.C.**
  Aussois, Mars 1993
  Titre de l'exposé : *Une méthode pour simuler "l'appel par valeur" dans le cadre de "l'appel par nom"*
- **Journées "Mathématiques et Informatique"**
  CIRM, Novembre 1993
  Titre de l'exposé : *Opérateurs de mise en mémoire*
- **Journées sur la logique classique**
  "Computational interpretations of classical proofs"
  CIRM, Décembre 1993
  Titre de l'exposé : *Classical natural numbers*
- **Séminaire de l'université de Turin**
  Turin, Mars 1995
  Titre de l'exposé : *Opérateurs de mise en mémoire et logique mixte*
- **Journées P.R.C.**
  "Mélanges de systèmes de réecriture algébriques et de systèmes logiques"
  LRI, Univ. Paris-Sud, Octobre 1996
  Titre de l'exposé : *Logique mixte et opérateurs de mise en mémoire*
- **Séminaire de Mathématiques Discrètes de l'université Claude Bernard Lyon I**
  - Lyon, Octobre 1998
  Titre de l'exposé : *Logique mixte propositionnelle*
  - Lyon, Septembre 1999
  Titre de l'exposé : *Preuves simples des théorèmes de complétude pour les logiques du second ordre classique et intuitionniste*
- **Journées du GDR ALP**
  "Mélanges de systèmes de réecriture algébriques et de systèmes logiques"
  ENS de Lyon, Avril 1999
  Titre de l'exposé : *Les I-types du système $\mathcal{F}$*
- **Journées du GDR ALP**
  "Logique classique et programmation"
  Chambéry, Mai 1999
  Titre de l'exposé : *Une logique classique non détrministe*
- **Rencontre entre les équipes de Chambéry et l'ENS Lyon**
  Aussois, 4-6 Juin 1999
  Titre de l'exposé : *Le $\lambda\mu^{++}$-calcul*

– **Séminaire de l'université de Savoie**
Chambéry, entre 1991 et 1999
Titres des exposés :
- *Simplification de la preuve du théorème de Krivine*
- *Preuve syntaxique du théorème de Krivine*
- *Opérateurs de mise en mémoire et types $\forall$-positifs du système AF2*
- *Opérateurs propres de mise en mémoire*
- *Typage normal et types $\forall$-positifs dans le système TTR*
- *Opérateurs de mise en mémoire et types $\forall$-positifs du système TTR*
- *Lambda-calcul dirigé*
- *Lambda σ-calcul et lambda ν-calcul*
- *Les entiers intuitionnistes et les entiers classiques en λC-calcul*
- *Les entiers classiques en $\lambda\mu$-calcul*
- *Un type général pour les opérateurs de mise en mémoire*
- *Les entiers classiques en λC-calcul*
- *Les systèmes numériques*
- *Les systèmes numériques typés*
- *Séparation d'un ensemble infini de λ-termes*
- *Logique mixte propositionnelle*
- *Représentation des fonctions en λI-calcul typé*
- *Relations entre les systémes F et S*
- *Un bon $\lambda\mu$-calcul*
- *Une preuve simple de la complétude de la logique du second ordre*

B- Participation à des colloques, groupes de travail, ...

– **19ème Ecole de printemps d'informatique théorique de LITP**
"Mathématiques et Informatique, quelques interactions"
Méjannes le clap, 3 - 7 juin 1991
– **CIRM**
"Logique et informatique"
Marseille, 15 - 19 juin 1992
– **Ecole d'été**
"Proof theory and Foundations of programming"
Chambéry, 28 Juin - 9 juillet 1993
– **Ecole d'été**
"Automated Deduction"
Chambéry, 4-15 Juin 1994
– **Journées P.R.C.**
"Mathématiques et Informatique"
CRIN (Nancy), Décembre 1994
– **Rencontre**
"Constructivité et logique de la vision"
ENS, Paris, 31 Mars 1995

- **BRA-Workshop**
  "Proofs and Types"
  Turin, 5-8 Juin 1995
- **HCM-TYPED LAMBDA CALCULUS project**
  Turin, 26-29 Novembre 1995
- **Journée GDR / PRC AMI**
  "Logique Classique pour le Programmeur"
  ENS, Lyon, 8 Mars 1996
- **CIRM**
  "Logique et modèles de calcul"
  Marseille, 16 - 20 Septembre 1996
- **Journées P.R.C.**
  "Mélanges de systèmes de réecriture algébriques et de systèmes logiques"
  Université de Savoie, Juin 1997
- **Journées P.R.C.**
  "Mélanges de systèmes de réecriture algébriques et de systèmes logiques"
  Université Paris 6, Janvier 1998
- **Séminaire du LIP**
  "Autour de P = NP ?"
  ENS Lyon, 18 février 1998
- **Rencontre entre les équipes de Chambéry et l'ENS Lyon**
  Aussois, 13-15 Mars 1998

C- Contact avec des équipes de recherche

- Equipe de Logique de Paris VII (J.-L. KRIVINE - M. PARIGOT)
- Equipe de Mathématiques discrètes de Marseille (J.-Y. GIRARD)
- Equipe d'Informatique de l'université de Turin (S. BERARDI - P. DEZANI - S. RONCHI)
- Equipe de Mathématiques discrètes de Lyon (L. ITURRIOZ - F. PABION)
- Equipe d'informatique de l'université de dell'Aquila (E. TRONCI)

D- Publications

1. *Opérateurs de mise en mémoire en lambda calcul pur et typé*
   Thèse de doctorat, université de savoie, 1993
2. *Opérateurs propres de mise en mémoire*
   CRAS. Paris, 317, Série I, pp. 1-6, 1993
3. *Preuve syntaxique d'un théorème de J.L. Krivine sur les opérateurs de mise en mémoire*
   CRAS. Paris, 318, Série I, pp. 201-204, 1994

4. *Quelques résultats sur le λC-calcul*
CRAS. Paris, 320, Série I, pp. 259-262, 1995
5. *Strong storage operators and data types*
Archive for Mathematical Logic 34, pp. 65-78, 1995
6. *Entiers intuitionnistes et entiers classiques en λC-calcul*
Informatique Théorique et Application, vol. 29, num 4, pp. 293-313, 1995
7. *A general type for storage operators*
Mathematical Logic Quarterly, 41, pp. 505-514, 1995
8. *Caractérisation opérationnelle des enties classiques en λC-calcul*
CRAS. Paris, 320, Série I, pp. 1431-1434, 1995
9. *Storage operators and directed lambda-calculus*
(avec R. David)
Journal of Symbolic Logic vol 60 num 4, pp. 1054-1086, 1995
10. *Storage operators and $\forall$-positive types of system TTR*
Mathematical Logic Quarterly, 42, pp. 349-368, 1996
11. *Opérateurs de mise en mémoire et types $\forall$-positifs*
Informatique Théorique et Application, vol. 30, num 3, pp. 261-293, 1996
12. *An example of a non adequate numeral system*
CRAS. Paris, 323, Série I, pp. 439-442, 1996
13. *La valeur d'un entier classique en $\lambda\mu$-calcul*
Archive for Mathematical Logic 36, pp. 461-473, 1997
14. *A syntactical proof of the operational equivalence of two λ-terms*
(avec R. David)
Theoretical Computer Science 180, pp. 371-375, 1997
15. *A conjecture on numeral systems*
Notre Dame Journal of Formal Logic, Vol. 38, pp. 270-275,1997
16. *S-storage operators*
Mathematical Logic Quarterly, 44, pp. 99-108, 1998
17. *On storage operators*
Proceedings of a congress "25 Years of Constructive Type Theory" Held in Venice, 19-21 Octobre 1995, pp. 173-190, Oxford University Press 1998
18. *Un résultat de complétude pour les types $\forall^+$ du système F*
(avec S. Farkh)
CRAS. Paris, 326, Série I, pp. 275-279, 1998
19. *Résultats de complétude pour des classes de types du système AF2*
(avec S. Farkh)
Informatique Théorique et Application, vol 31, num 6, pp. 513-537, 1998

20. *Une réponse négative à la conjecture de E.Tronci pour les systèmes numériques typés*
Informatique Théorique et Application, vol. 31, num 6, pp. 539-558, 1998
21. *Mixed logic and storage operators*
Accepté : A paraitre dans Archive for Mathematical Logic
22. *Propositional mixed logic : its syntax and semantics*
(avec A. Nour)
Soumis à Notre Dame Journal of Formal Logic, 1998
23. *Les I-types du système F*
Soumis à Informatique Théorique et Application, 1999
24. *Les types de données systaxiques du système F*
(avec S. Farkh)
Soumis à Informatique Théorique et Application, 1999
25. *Types complets dans une extension du système AF2*
(avec S. Farkh)
Soumis à Informatique Théorique et Application, 2000
26. *A non-deterministic classical logic (the $\lambda\mu^{++}$-calculus)*
Soumis à Theoretical Computer Science, 2000
27. *Simple proof of the completeness theorem for second order classical and intuitionictic logic*
(avec C. Raffalli)
En préparation

Livre de logique pour le second cycle
(avec R. David et C. Raffalli)
Titre : *Logique Mathématique : Introduction à la théorie de la démonstration*
Dunod (en préparation)

F- Encadrement doctoral

Encadrement d'un étudiant pour préparer une thèse de doctorat en logique mathématiques.

Nom et Prénom de l'étudiant : FARKH Samir
Titre de la thèse : "Types de données en logique du second ordre"
Date de soutenance : 14 décembre 1998
Jury : J.-L. KRIVINE, C. PAULIN, S. BERARDI, R. DAVID, K. NOUR

E- Activités de referee

– Plusieurs referees pour des journaux internationaux comme "Annals of Pure and Applied Logic" et "Theoretical Computer Science"
– Plusieurs referees pour "Zentralblat fur Mathematik" et "Mathematical Reviews"

F- Remarques

– Je bénéficie depuis septembre 1996 d'un prime d'encadrement doctoral
– J'étais à plusieurs reprises membre dans des comités d'organisation des congrés nationaux et internationaux :
– Ecole d'été : "Automated deduction"
Chambéry, 1994
– Journées P.R.C. : "Mélanges de systèmes de réecriture algébriques et de systèmes logiques"
Université de Savoie, 1997
– Micro Ecole Turin : " A quick introduction to Logical Relations"
(Ugo de'Liguoro) Université de Savoie, 7-8 octobre 1999

RESPONSABILITÉS

1995/98 Responsable de la bibliothèque du laboratoire de Mathématiques
1995/99 Responsable des prépublications du laboratoire de Mathématiques
1996/99 Membre du conseil du laboratoire de Mathématiques

EXPERIENCES D'ENSEIGNEMENT

1988/89 **Professeur de Mathématiques en année préparatoire**
Université JINANE - Tripoli -LIBAN

1990/91 **Vacataire**
Université Claude Bernard - Lyon I
TD en D.E.A de Mathématiques pures (cours de "λ-calcul")

1992/93 **Attaché Temporaire d'Enseignement et de Recherche (A.T.E.R.)**
Ecole d'Ingénieurs de l'Université de Savoie (E.S.I.G.E.C.)
TD d'analyse numérique et de statistique

1993/00 **Maître de conférences**
Université de Savoie

93-94
- TD d'algèbre et d'analyse en DEUG A2
- TD d'algèbre en licence de Mathématiques
- TD de "λ-calcul" en DEA de Mathématiques de l'université Lyon I

94-95
- TD d'algèbre et d'analyse en DEUG A2
- TD d'algèbre en licence de Mathématiques
- TD de logique en maîtrise de Mathématiques
- 1 Projet de maîtrise (logique)
- TD de "λ-calcul" en DEA de Mathématiques de l'université Lyon I

95-96
- TD d'algèbre et d'analyse en DEUG A2
- 3 Projets Math-Info.
- TD d'algèbre en licence de Mathématiques
- 1 Projet de licence (algèbre)
- Cours et TD de logique en maîtrise de Mathématiques
- Cours de "logique pour l'informatique"
en DEA de Mathématiques de l'université Lyon I
- Cours de "logique et programmation fonctionnelle"
en DEA d'informatique de l'université de Savoie

96-97
- TD d'algèbre et d'analyse en DEUG A2
- 3 Projets Math-Info
- TD d'algèbre en licence de Mathématiques
- Cours et TD de logique en maîtrise de Mathématiques
- 3 Projets de maîtrise (logique)
- 1 Projet de maîtrise (Algèbre)

97-98
- TD d'algèbre et d'analyse en DEUG MIAS 2
- 3 Projets Math-Info
- TD d'algèbre en licence de Mathématiques
- Cours et TD de logique en maîtrise de Mathématiques
- 1 Projet de maîtrise (logique)

98-99
- TD d'algèbre et d'analyse en DEUG MIAS 2
- TD d'algèbre en DEUG MIAS 1
- 3 Projets Math-Info
- TD d'algèbre en licence de Mathématiques
- Cours et TD de logique en maîtrise de Mathématiques
- 1 Projet de maîtrise (logique)
- 1 Projet de magistère de Mathématiques de l'ENS Lyon (logique)
- Cours de "logique pour l'informatique"

en DEA de Mathématiques de l'université Lyon I

99-00
- TD d'algèbre et d'analyse en DEUG MIAS SM 2
- TD d'algèbre en DEUG MIAS 1
- 3 Projets Math-Info
- TD d'algèbre en licence de Mathématiques
- Cours et TD de logique en maîtrise de Mathématiques
- 1 Projet de maîtrise (logique)
- 1 Projet de magistère de Mathématiques de l'ENS Lyon (logique)
- Cours de "Théorie de la démonstration et λ-calcul"

en DEA de Mathématiques de l'université Lyon I

Chapitre 1

Introduction

Le λ-calcul pur a été inventé vers 1930, et a connu un développement considérable pour ses rapports étroits avec les langages de programmation fonctionnelle. L'intérêt principal de ce calcul provient essentiellement de la simplicité de sa syntaxe et de sa capacité à programmer toutes les fonctions calculables. Le λ-calcul typé suscite depuis quelques années un grand intérêt à cause du lien qu'il établit entre les notions de programme et de preuve en logique, c'est ce qu'on appelle la "correspondance de Curry-Howard". Parmi les systèmes de typage qui ont un grand pouvoir d'expression, on trouve le système $\mathcal{AF}2$ des types du second ordre de J.-L. Krivine. L'intérêt de ce système est que les programmes qu'il produit sont prouvés.

Mes travaux de recherche tournent autour des théorèmes de programmation en λ-calcul pur et typé. Ils recouvrent essentiellement les thèmes suivants :

1- Opérateurs de mise en mémoire

Le λ-calcul ne peut être considéré comme un vrai langage de programmation que si une stratégie de réduction est définie pour exécuter les programmes. Parmi les stratégies les plus simples à implémenter, on trouve la stratégie de la réduction gauche (appel par nom) qui consiste à réduire toujours le redex le plus à gauche dans le terme. Cette stratégie a de bonnes propriétés ; elle se termine toujours si on l'applique à des termes normalisables. Mais, avec elle, l'argument d'une fonction est recalculé à chaque utilisation. J.-L. Krivine a introduit en 1990 la notion d'opérateurs de mise en mémoire pour remédier à ce défaut. Il s'agit de λ-termes clos qui, pour un type de données fixé, permettent de simuler "l'appel par valeur" dans le cadre d'un "appel par nom". Les opérateurs de mise en mémoire ont été utilisés d'une manière implicite dans la preuve du théorème de programmation en λ-calcul pur. En effet, pour représenter la composition sur les fonctions partielles, on commence par réduire les arguments avant de donner leurs résultats (s'ils existent) à la fonction. Les opérateurs de mise en mémoire permettent de réaliser ce qu'on vient d'expliquer.

Pour étudier les propriétés des opérateurs de mise en mémoire en λ-calcul pur, nous avons introduit (R. David et moi même) une nouvelle structure : le λ-calcul dirigé (un langage de contexte pour le λ-calcul), et nous avons démontré que ce calcul garde les principales propriétés du λ-calcul ordinaire. Le λ-calcul dirigé permet de trouver une définition équivalente pour les opérateurs de mise en mémoire plus pratique pour établir leurs propriétés. Citons-en quelques unes : la stabilité de l'ensemble des opérateurs de mise en mémoire par la β-équivalence, la non et la semi-décidabilité du problème de l'existence des opérateurs pour un λ-terme normal clos, et une inégalité contrôlant le temps calcul d'un opérateur. J'ai étudié ensuite le problème de la mise en mémoire sous une forme particulière ; λ-terme normal (opérateurs forts) ou λ-terme clos (opérateurs propres). J'ai donné des conditions nécessaires et suffisantes pour que les représentations récursives et itératives des données en λ-calcul possèdent des opérateurs forts, et une condition suffisante pour obtenir des opérateurs propres. J'ai prouvé que chaque ensemble fini de λ-termes normaux clos possède des opérateurs propres de mise en mémoire qui sont forts dans le cas où les termes ont des formes η-normales distinctes. J'ai associé à chaque λ-terme S qui réalise la fonction successeur sur les entiers de Church une notion de S-opérateur de mise en mémoire. J'ai démontré que chaque S-opérateur est un opérateur de mise en mémoire et que la réciproque n'est pas toujours vraie.

Concernant le λ-calcul typé, J.-L. Krivine a montré qu'en utilisant la traduction de Gödel de la logique classique en logique intuitionniste, on peut trouver, pour chaque type de données courant, un type très simple du système $\mathcal{AF}2$ pour les opérateurs de mise en mémoire. Ceci donne un moyen de les obtenir. Historiquement, le type proposé par J.-L. Krivine a été trouvé avant la notion d'opérateurs de mise en mémoire. J.-L. Krivine a démontré que les logiques classique et intuitionniste prouvent la totalité de la même classe de fonction. Dans sa démonstration, il a introduit les notions suivantes : $A[x]$ est un "type entrée" (resp. "type sortie") si la formule $A[x] \to A^g[x]$ (resp. $A^g[x] \to \neg\neg A[x]$) est prouvable en logique intuitionniste où ^{g} est la traduction de Gödel de la logique classique dans la logique intuitionniste. Alors si $A[x]$ est un type entrée et $B[x]$ est un type sortie, une preuve classique de $A[x] \to B[x]$ peut se transformer en une preuve intuitionniste. La notion d'opérateurs de mise en mémoire a été découverte en cherchant la propriété de tous les λ-termes de type $N^g[x] \to \neg\neg N[x]$ où $N[x]$ est le type des entiers dans $\mathcal{AF}2$.

Dans mes travaux, j'ai trouvé une version plus générale du théorème de J.-L. Krivine[1] pour les types $\forall^+$ (tous les quantificateurs du second ordre sont en position positive dans ces types), et pour les transformations de Gödel (une généralisation de la traduction de Gödel classique) dans le système $\mathcal{AF}2$ et le système $\mathcal{TTR}$ des types récursifs de M. Parigot. Les types que j'ai proposés ne permettent de typer qu'une petite classe des opérateurs de mise en mémoire.

[1] J.-L. Krivine a trouvé indépendament une partie de ces résultats.

Par contre, j'ai trouvé une légère extension du système $\mathcal{AF}2$ dans laquelle on peut en typer une grande classe.

2- Types de données dans les systèmes $\mathcal{F}$ et $\mathcal{AF}2$ [2]

La classe $\mathcal{A}$ des types de données du système $\mathcal{AF}2$ définie par J.-L. Krivine a la particularité suivante : un λ-terme normal est typable d'un type $D \in \mathcal{A}$ ssi il est dans l'interprétation de D pour la sémantique de la réalisabilité. Nous nous sommes intéressés à construire des classes plus larges de types ayant cette particularité que nous avons appelés "types complets" puisque la sémantique considérée est complète pour ces types. Nous avons montré que les types $\forall^+$ sont des types complets pour la sémantique des parties saturées par $\beta\eta$-équivalence. Comme la clôture par expansion faible de tête est la notion la plus faible pour la sémantique de réalisabilité, nous nous sommes intéressés à étendre le résultat précédent à ces modèles. Ceci nous a amené à construire une classe plus restreinte de types (les bons types positifs) pour lesquels le type est prśervé par η-réduction. Nous avons démontré que cette classe de types est complet pour la sémantique des parties saturées par expansion faible de tête. Une deuxième façon pour avoir la préservation de type par η-réduction est d'étendre $\mathcal{AF}2$ à un système de typage paramétré par une relation de sous-typage à la Mitchell. Nous avons démontré que, dans le nouveau système, les types $\forall^+$ sont complets pour la sémantique des parties stables par expansion faible de tête.

Nous avons donné une définition d'un type de données du système $\mathcal{F}$ par des types entrée-sortie qui servent à caractériser les domaines et codomaines des fonctions que l'on cherche à programmer. Nous avons démontré que les types $\forall^+$ sont des types entrée-sortie, et que tout type entrée est un type sortie. La réciproque est démontrée dans des cas particuliers où on impose des restrictions sur la règle d'élimination des quantificateurs du second ordre.

Les formules du système $\mathcal{F}$ qui représentent les types de données courants sont tous habités par au moins un λK-terme. Une question se pose : Peut-on représenter les types de données courants par des types du système $\mathcal{F}$ habités uniquement par des λI-termes (ces types sont appelés "I-types") ? J'ai démontré que les I-types sont $\forall^+$. Ce qui m'a permis de présenter la liste de tous les I-types contenant au plus deux quantificateurs. On ne connaît pas encore s'il existe des I-types habités par plus d'un λI-terme $\beta\eta$-normal.

3- Programmation en logique classique et logique mixte

M. Parigot et J.-L. Krivine ont trouvé des moyens de donner un contenu algorithmique à des preuves faites en logique classique. On sait qu'un "entier intuitionniste" (i.e. provient d'une preuve en logique intuitionniste) est β-équivalent

[2] Les principaux résultats de cette partie ont été obtenus avec S. Farkh dans le cadre de sa thèse de doctorat.

à un entier de Church. Ceci n'est plus valable pour les "entiers classiques" (i.e. provient d'une preuve en logique classique). Mais ils ont démontré que les opérateurs de mise en mémoire typés dans le système intuitionniste $\mathcal{AF}2$ d'un type particulier peuvent servir aussi pour trouver la valeur d'un entier classique.

J'ai obtenu quelques résultats dans le $\lambda\mathcal{C}$-calcul de J.-L. Krivine et le $\lambda\mu$-calcul de M. Parigot. En $\lambda\mathcal{C}$-calcul, nous avons caractérisé, par des méthodes purement syntaxiques[3] le comportement des $\lambda\mathcal{C}$-termes de type "les raisonnements par l'absurde intuitionniste et classique". On obtient ainsi les opérateurs de contrôles des langages de programmation. J'ai caractérisé aussi le comportement des entiers classiques du λC-calcul et j'ai donné quelques restrictions sur le système de typage afin d'obtenir uniquement les entiers intuitionnistes. En $\lambda\mu$-calcul, j'ai trouvé des algorithmes plus simples que ceux de M. Parigot qui permettent de trouver la valeur d'un entier classique.

Le $\lambda\mu$-calcul possède "presque" toutes les bonnes propriétés : la confluence, la conservation de type et la normalisation forte. Par contre, on perd dans ce calcul l'unicité de la représentation des données. De plus la présentation du système typé n'est pas très naturelle. J'ai présenté une extension du $\lambda\mu$-calcul appelé $\lambda\mu^{++}$-calcul qui code exactement la déduction naturelle classique du second ordre. Ce calcul a les propriétés suivantes : la conservation de type, la normalisation forte, l'unicité de la représentation des données et la confluence sur les types de données. Ce calcul permet de programmer le ou-parallèle.

J.-L. Krivine et moi même avons défini un système de typage basé sur une logique appelée "mixte". Ce système permet essentiellement de distinguer entre les preuves classiques et les preuves intuitionnistes et dans lequel on peut caractériser à la fois les opérateurs de mise en mémoire et les opérateurs de contrôles.

Nous avons présenté (A. Nour et moi même) une logique propositionnelle mixte contenant des "copies disjointes" des logiques minimale, intuitionniste et classique. Nous avons démontré un résultat de complétude pour cette logique par rapport à une sémantique de type Kripke. Ce système nous a permis de formaliser la notion de "bonne démonstration" par rapport au raisonnement par l'absurde. Nous nous intéressons à étudier le problème d'élimination des coupures dans la version calcul des séquents de cette logique.

4- Systèmes numériques

La classe de fonctions numériques programmables en λ-calcul pur varie selon le codage choisi pour les entiers. H. Barendregt a démontré que pour un cogade d'entiers (un système d'entiers), on peut représenter toutes les fonctions récursives ssi les trois fonctions particulières : le successeur, le prédécesseur et le test à zéro le sont. J'ai démontré que les trois fonctions précédentes sont indépen-

[3] J.-L. Krivine a démontré, avec des méthodes sémantiques, une partie de ces résultats.

dantes (i.e. j'ai construit des systèmes dans lesquels on peut représenter deux de ces fonctions sans la troisième). Il me semble qu'on ne peut pas faire mieux (i.e. il faut au moins trois fonctions unaires pour pouvoir tout représenter). J'ai démontré qu'un système numérique ayant des λ-termes pour le successeur, le prédécesseur et le test à zéro possède des opérateurs de mise en mémoire. E. Tronci a posé la question suivante : est-il vrai qu'un système d'entiers ayant un successeur et un test à zéro (système numérique) possède un prédécesseur ssi il possède un opérateur de mise en mémoire? Cette question est toujours ouverte mais j'en ai apporté une réponse négative pour les systèmes numériques typables dans le système $\mathcal{F}$.

5- Résultats divers

Les méthodes que j'ai développé m'ont permis de redémontrer des résultats connus en λ-calcul. Par exemple, nous avons caractérisé (R. David et moi même) d'une manière syntaxique l'équivalence opérationnelle de deux λ-termes non $\beta\eta$-équivalents.

C. Raffalli et moi même avons trouvé des preuves très simples des théorèmes de complétude des logiques du second ordre classique et intuitiuonniste.

Chapitre 2

Notions de base en λ-calcul pur et typé

2.1 Quelques résultats du λ-calcul pur

On note Λ l'ensemble des termes du λ-calcul. On adopte la notation de J.-L. Krivine pour l'application : étant donnés des λ-termes $t, u, u_1, ..., u_n$, l'application de t à u est notée $(t)u$, et $(...((t)u_1)...)u_n$ est noté $(t)u_1...u_n$.

On note $\mathbf{id} = \lambda x x$, $\mathbf{0} = \lambda x \lambda y y$ et $\mathbf{1} = \lambda x \lambda y x$. Étant donnés deux λ-termes t, u et un entier n, on pose, par définition, $(t)^n u = (t)...(t)u$ (t étant répété n fois) ; en particulier $(t)^0 u = u$. Le λ-terme $\underline{n} = \lambda x \lambda f (f)^n x$ est l'entier n de Church.

Si t et u sont des λ-termes, on note $< t, u >= \lambda x (x)t\ u$.

Si t est un λ-terme, on note $Fv(t)$ l'ensemble de ses variables libres.

On note $\rightarrow_\beta$ (resp. $\rightarrow_\eta$) la β-réduction (resp. la η-réduction) et par $\simeq_\beta$ (resp. $\simeq_{\beta\eta}$) la β-équivalence (resp. la $\beta\eta$-équivalence).

La notation $u \succ v$ signifie que v est obtenu à partir de u par réduction de tête. On note ainsi $n(u, v)$, le nombre de pas effectué pour passer de u à v. On dit qu'un λ-terme est résoluble ssi sa réduction de tête termine.

La réduction de tête possède les propriétés suivantes (voir [20]) :

Théorème 2.1.1 *Si $t \succ t'$, alors pour tout $u_1, ..., u_r \in \Lambda$:*
1) Il existe $v \in \Lambda$ tel que $(t)u_1...u_r \succ v$, $(t')u_1...u_r \succ v$, et $n((t)u_1...u_r, v) = n((t')u_1...u_r, v) + n(t, t')$.
2) $t[u_1/x_1, ..., u_r/x_r] \succ t'[u_1/x_1, ..., u_r/x_r]$, et
$n(t[u_1/x_1, ..., u_r/x_r], t'[u_1/x_1, ..., u_r/x_r]) = n(t, t')$.

La réduction de tête faible $\succ_f$ est la plus petite relation transitive et réflexive telle que $(\lambda xu)vv_1...v_m \succ_f (u[v/x])v_1...v_m$. Elle possède la propriété suivante :

Théorème 2.1.2 *Pour tous λ-termes t, u, v, si $t \succ_f u$ alors $(t)v \succ_f (u)v$.*

2.2 Le système de typage $\mathcal{AF}2$

Pour pouvoir construire un langage de programmation basé sur le λ-calcul typé il faut envisager un système de type qui permet d'exprimer les spécifications exactes des programmes. Le candidat naturel est celui qu'a choisi J.-L. Krivine sous le nom d'Arithmétique Fonctionnelle du second ordre (noté en abrégé $\mathcal{AF}2$) (voir [18]). Au plan des propriétés théoriques, normalisation forte et représentation des fonctions calculables, le système $\mathcal{AF}2$ ne se distingue pas du système $\mathcal{F}$. La seule différence provient de sa capacité à exprimer les spécifications exactes des programmes, ce qui permet d'obtenir un programme calculant une fonction en écrivant une démonstration de sa totalité.

Le système

On considère le calcul des prédicats intuitionniste du second ordre, écrit avec les symboles logiques $\bot$, $\to$ et $\forall$. On note $\neg A = A \to \bot$ et $A_1, ..., A_n \to A$ la formule $A_1 \to (A_2 \to (...(A_n \to A)...))$.

Si X est une variable de relation unaire, t et t' deux termes, alors la formule $\forall X[Xt \to Xt']$ est notée $t = t'$ et dite équation fonctionnelle ou formule équationnelle. Un cas particulier de l'équation $t = t'$ est une formule de la forme : $t[u_1/x_1, ..., u_n/x_n] = t'[u_1/x_1, ..., u_n/x_n]$ ou $t'[u_1/x_1, ..., u_n/x_n] = t[u_1/x_1, ..., u_n/x_n]$, $u_1, ..., u_n$ étant des termes du langage.

Considérons un langage L du second ordre, et un système E d'équations fonctionnelles de L. On décrit un système de λ-calcul typé, appelé Arithmétique Fonctionnelle du second ordre (en abrégé $\mathcal{AF}2$), dont les types sont les formules de L. Étant donnés un λ-terme t, un type A et un contexte $\Gamma = x_1 : A_1, ..., x_n : A_n$, on définit au moyen des règles suivantes la notion "t est typable, à l'aide du système équationnel E, de type A dans le contexte Γ". Cette notion est notée $\Gamma \vdash_{\mathcal{AF}2} t : A$.

(1) $\Gamma \vdash_{\mathcal{AF}2} x_i : A_i \ (1 \leq i \leq n)$

(2) $$\frac{\Gamma, x : A \vdash_{\mathcal{AF}2} t : B}{\Gamma \vdash_{\mathcal{AF}2} \lambda xt : A \to B}$$

(3) $$\frac{\Gamma \vdash_{\mathcal{AF}2} u : A \to B \quad \triangle \vdash_{\mathcal{AF}2} v : A}{\Gamma, \triangle \vdash_{\mathcal{AF}2} (u)v : B}$$

(4) $\dfrac{\Gamma \vdash_{\mathcal{AF}2} t : A}{\Gamma \vdash_{\mathcal{AF}2} t : \forall x A}$, x n'est pas libre dans Γ

(5) $\dfrac{\Gamma \vdash_{\mathcal{AF}2} t : \forall x A}{\Gamma \vdash_{\mathcal{AF}2} t : A[u/x]}$, u est un terme

(6) $\dfrac{\Gamma \vdash_{\mathcal{AF}2} t : A}{\Gamma \vdash_{\mathcal{AF}2} t : \forall X A}$, X n'est pas libre dans Γ

(7) $\dfrac{\Gamma \vdash_{\mathcal{AF}2} t : \forall X A}{\Gamma \vdash_{\mathcal{AF}2} t : A[F/X]}$, F est une formule

(8) $\dfrac{\Gamma \vdash_{\mathcal{AF}2} t : A[u/x]}{\Gamma \vdash_{\mathcal{AF}2} t : A[v/x]}$, $u = v$ est un cas particulier d'une équation de E

Les types du système $\mathcal{F}$ sont les formules construites avec les symboles logiques $\forall$, $\rightarrow$, et les variables X, Y,... de relation 0-aire (variables propositionnelles). Les règles de typage du système $\mathcal{F}$ constituent un sous-système des règles ci-dessus : ce sont les règles de typage (1), (2), (3), et (6), (7) restreintes aux formules propositionelles.

Le système $\mathcal{AF}2$ possède les propriétés suivantes (voir [19]) :

Théorème 2.2.1 *1) Si $\Gamma \vdash_{\mathcal{AF}2} t : A$, et $t \rightarrow_\beta t'$, alors $\Gamma \vdash_{\mathcal{AF}2} t' : A$ (théorème de conservation de type).*
2) Si $\Gamma \vdash_{\mathcal{AF}2} t : A$, alors t est fortement normalisable (théorème de normalisation forte).

Soient L un langage du second ordre et E un système d'équations de L. On définit sur l'ensemble des termes de L une relation d'équivalence notée $\approx_E$ par : $a \approx_E b$ ssi on peut l'obtenir au moyen des règles suivantes :
– Si $a = b$ est un cas particulier d'une équation de E, alors $a \approx_E b$;
– Quels que soient les termes a, b, c de L, on a : $a \approx_E a$; et si $a \approx_E b$ et $b \approx_E c$, alors $a \approx_E c$.
– Si f est un symbole de fonction n-aire de L, et si $a_i \approx_E b_i$ $(1 \leq i \leq n)$, alors $f(a_1, ..., a_n) \approx_E f(b_1, ..., b_n)$.

Le lemme suivant permet de généraliser la règle de typage (8) (voir [12]).

Lemme 2.2.1 *Si $\Gamma \vdash_{\mathcal{AF}2} t : A[u/x]$ et $u \approx_E v$, alors $\Gamma \vdash_{\mathcal{AF}2} t : A[v/x]$.*

Théorème de programmation

C. Böhm et A. Berarducci ont défini dans [3] les types de données algébriques dans le système $\mathcal{F}$. Ensuite J.-L. Krivine a généralisé leur définition au système $\mathcal{AF}2$ (voir [20]). Voilà quelques exemples :

Le type identité est la formule $Id[x] = \forall X\{Xi \to Xx\}$ (i est un symbole de constante). On a le résultat suivant : si t est un λ-terme normal, alors $\vdash_{\mathcal{AF}2} t : Id[i]$ ssi $t = \mathbf{id}$. Le type identité du système $\mathcal{F}$ est la formule $Id = \forall X\{X \to X\}$.

Le type booléen est la formule $B[x] = \forall X\{X1, X0 \to Xx\}$ (0 et 1 sont des symboles de constante). On a le résultat suivant : si t est un λ-terme normal et $b \in \{0,1\}$, alors $\vdash_{\mathcal{AF}2} t : B[b]$ ssi $t = \mathbf{b}$. Le type booléen du système $\mathcal{F}$ est la formule $B = \forall X\{X, X \to X\}$.

Le type des entiers naturels est la formule $N[x] = \forall X\{X0, \forall y(Xy \to Xsy) \to Xx\}$ (s est un symbole de fonction unaire et 0 est un symbole de constante). On a le résultat suivant : si t est un λ-terme normal et $n \in \mathbb{N}$, alors $\vdash_{\mathcal{AF}2} t : N[s^n(0)]$ ssi $t = \underline{n}$. Le type des entiers naturels du système $\mathcal{F}$ est la formule $N = \forall X\{X, (X \to X) \to X\}$.

On peut définir de la même manière les types liste, arbre, produit, somme, ...

Je vais présenter le théorème de programmation seulement sur les entiers.

Un système d'équations E est dit adéquat pour le type $N[x]$ ssi $s(a) \not\approx_E 0$ et si $s(a) \approx_E s(b)$, alors $a \approx_E b$. Dans la suite on suppose que les systèmes d'équations sont tous adéquats pour le type $N[x]$.

Soit f une fonction totale de $\mathbb{N}^k$ dans $\mathbb{N}$. On dit que E définit la fonction f ssi $f(n) = m \Leftrightarrow f(s^n(0)) \approx_E s^m(0)$. Étant donné un λ-terme P_f, on dit que P_f représente la fonction f si, quels que soient $n_1, ..., n_k \in \mathbb{N}$, $(P_f)\underline{n_1}...\underline{n_k} \to_\beta \underline{f(n_1, ..., n_k)}$.

On peut énoncer à présent le théorème de programmation qui est l'une des motivations principales du système $\mathcal{AF}2$ (voir [19]).

Théorème 2.2.2 *Soient f une fonction totale de $\mathbb{N}^k$ dans $\mathbb{N}$ et E un système d'équations adéquat définissant f. Si P_f est un λ-terme tel que $\vdash_{\mathcal{AF}2} P_f : \forall x_1...\forall x_m\{N[x_1], ..., N[x_k] \to N[f(x_1, ..., x_m)]\}$, alors P_f représente f.*

La sémantique

Le principe de la notion d'interprétation est d'associer à chaque type A un ensemble de λ-termes qui contient ceux de type A. Ainsi, on aura à notre disposition un puissant outil sémantique pour prouver les propriétés des termes typés. La puissance de cette notion provient de la variété des interprétations possibles pour le quantificateur du second ordre. Ces différentes sortes d'interprétations sont déterminées par la donnée d'un sous-ensemble R de $P(\Lambda)$ qui est destiné à être le domaine dans lequel sont choisies les interprétations des variables du second ordre.

Étant donnés deux éléments G et G' de $P(\Lambda)$, on définit un élément de $P(\Lambda)$ noté $G \to G'$, en posant : $u \in (G \to G') \Leftrightarrow (u)t \in G'$ quel que soit $t \in G$.

Un sous-ensemble R de $P(\Lambda)$ est dit adéquat si :
– $G, G' \in R \Rightarrow (G \to G') \in R$;
– Pour toute partie σ de R, l'intersection des éléments de σ appartient à R.

Soit L un langage du second ordre. On va définir la notion d'un Λ-modèle pour L : c'est une modification de la notion classique d'un modèle du second ordre, dans lequel l'ensemble des valeurs de vérité n'est pas $\{0, 1\}$ comme d'habitude, mais un sous-ensemble adéquat R de $P(\Lambda)$.

Un Λ-modèle pour le langage L est la donnée de :
– Un ensemble $|M|$, supposé non vide, appelé la base de M ;
– Un sous-ensemble adéquat R de $P(\Lambda)$ (R est appelé l'ensemble des valeurs de vérité de M) ;
– Pour tout symbole de fonction n-aire f de L, une fonction $f_M : |M|^n \to |M|$;
– Pour tout symbole de relation n-aire P de L, une fonction $P_M : |M|^n \to R$.

Une interprétation I est une application de l'ensemble des variables d'individu (resp. de relation n-aire) dans $|M|$ (resp. dans $R^{|M|^n}$).

Soient I une interprétation, x (resp. X) une variable d'individu (resp. de relation n-aire), et a (resp. Φ) un élément de $|M|$ (resp. de $R^{|M|^n}$). On définit une interprétation $J = I[x \leftarrow a]$ (resp. $J = I[X \leftarrow \Phi]$) en posant $J(x) = a$ (resp. $J(X) = \Phi$) et $J(\xi) = I(\xi)$ (resp. $J(\xi') = I(\xi')$) pour toute variable $\xi \neq x$ (resp. $\xi' \neq X$).

Soit I une interprétation. À chaque terme t de L est associée sa valeur $t_{M,I} \in |M|$, définie par induction sur t :
– Si t est une variable x, alors $t_{M,I} = I(x)$;
– Si $t = f(t^1, ..., t^n)$, alors $t_{M,I} = f_M(t^1_{M,I}, ..., t^n_{M,I})$.

La valeur d'une formule A dans M et dans une interprétation I, notée par $|A|_{M,I}$ est un élément de R défini par induction sur A au moyen des règles suivantes :
– Si A est une formule atomique $P(t^1, ..., t^n)$, où P est un symbole (resp. une variable) de relation n-aire, et $t^1, ..., t^n$ sont des termes de L, alors on définit $|A|_{M,I}$ par $P_M(t^1_{M,I}, ..., t^n_{M,I})$ (resp. $I(X)(t^1_{M,I}, ..., t^n_{M,I})$) qui est un élément de R ;
– Si $A = B \to C$, alors $|A|_{M,I} = |B|_{M,I} \to |C|_{M,I}$;
– Si $A = \forall x B$, où x est une variable d'individu, alors $|A|_{M,I} = \bigcap\{|B[x]|_{M,I[x \leftarrow a]}$; $a \in |M|\}$;
– Si $A = \forall X B$, où X est une variable de relation n-aire, alors $|A|_{M,I} = \bigcap\{|B[X]|_{M,I[X \leftarrow \Phi]}$; $\Phi \in R^{|M|^n}\}$.

Il est clair que $|A|_{M,I}$ dépend seulement des valeurs (par I) des variables libres

de A. En particulier, si A est un type clos, $|A|_{M,I}$ ne dépend pas de I, on la note alors $|A|_M$.

Si on se restreint au système $\mathcal{F}$, un Λ-modèle est composé uniquement d'une partie adéquate R de $P(\Lambda)$, et pour toute constante (resp. variable) propositionnelle P, d'un élément $|P|_M$ de R.

On dit qu'un Λ-modèle M satisfait l'équation $u = v$, si pour toute interprétation I, $u_{M,I} = v_{M,I}$. Si E est un ensemble d'équations de L, on dit que M satisfait E, ou que M est un modèle de E, si M satisfait toute équation de E.

Soit S une relation binaire sur Λ. On dit que S est applicable si, quels que soient les λ-termes t, u et v, $t\ S\ u \Rightarrow (t)v\ S\ (u)v$.

Soit S une relation applicable. Une partie G de Λ est dite stable par S (ou S-saturée) si, quels que soient les λ-termes t et u, $[u \in G \text{ et } t\ S\ u] \Rightarrow t \in G$.

On note $P_S(\Lambda)$ l'ensemble des parties S-saturées de Λ. On a alors le lemme suivant (voir [10]) :

Lemme 2.2.2 *$P_S(\Lambda)$ est un sous-ensemble adéquat de $P(\Lambda)$.*

Soient M un Λ-modèle pour L, R l'ensemble des valeurs de vérité de M et S une relation binaire sur Λ applicable. On dit que M est un Λ_S-modèle si : $(\lambda x u)v\ S\ u[v/x]$ et $R \subset P_S(\Lambda)$.

Un Λ_S-modèle est dit standard si $R = P_S(\Lambda)$. Pour tout type clos A, on note par $|A|_S = \bigcap\{|A|_M \ /\ M\ \Lambda_S\text{-modèle de } E\ \}$.

Le théorème suivant (connu sous le nom du lemme d'adéquation) montre que la notion de typage est plus "fort" que celle de réalisabilité. Il permet d'obtenir l'unicité de la représentation des données et le théorème de normalisation forte (voir [19]).

Théorème 2.2.3 *Si $\vdash_{\mathcal{AF}2} t : A$, alors $t \in |A|_S$.*

Chapitre 3

Opérateurs de mise en mémoire

3.1 La définition

Soit F un λ-terme (une fonction) et $\theta_n \simeq_\beta \underline{n}$ (un argument non calculé de la fonction). Durant la réduction gauche de $(F)\theta_n$, θ_n peut être calculé plusieurs fois (le nombre de fois que F l'utilise). On souhaite transformer $(F)\theta_n$ en $(F)\underline{n}$ et que cette transformation dépende uniquement de θ_n (et pas de F). C'est à dire trouver un λ-terme clos T qui vérifie les propriétés suivantes :

- Pour tout λ-terme F, pour tout $n \in \mathbb{N}$, et pour tout $\theta_n \simeq_\beta \underline{n}$, $(T)\theta_n F \succ_f (F)\underline{n}$.

- Le temps de calcul de $(T)\theta_n F \succ_f (F)\underline{n}$ dépend uniquement de θ_n.

Définition temporaire 1 : Un λ-terme clos T est dit opérateur de mise en mémoire pour les entiers de Church ssi pour tout $n \in \mathbb{N}$, et pour tout $\theta_n \simeq_\beta \underline{n}$, $(T)\theta_n f \succ_f (f)\underline{n}$ (où f est une nouvelle variable).

Il est clair qu'un opérateur de mise en mémoire satisfait les propriétés voulues. En effet, comme $(T)\theta_n f \succ_f (f)\underline{n}$, la variable f n'arrive jamais en tête durant la réduction, et on peut donc la remplacer par un λ-terme quelconque. On verra plus loin (théorème 3.1.1) qu'il n'est pas possible de trouver la forme normale de θ_n. On change donc la définition.

Définition temporaire 2 : Un λ-terme T est dit opérateur de mise en mémoire pour les entiers de Church ssi pour tout $n \in \mathbb{N}$, il existe un λ-terme clos $\tau_n \simeq_\beta \underline{n}$, tel que pour tout $\theta_n \simeq_\beta \underline{n}$, $(T)\theta_n f \succ_f (f)\tau_n$ (où f est une nouvelle variable).

Soit $\underline{s} = \lambda n \lambda x \lambda f((n)(f)x)f$ un λ-terme pour le successeur sur les entiers de

Church. Si on prend : $T_1 = \lambda n((n)\delta)G$ où $\delta = \lambda f(f)\underline{0}$ et $G = \lambda x\lambda y(x)\lambda z(y)(\underline{s})z$; $T_2 = \lambda n\lambda f(((n)f)F)\underline{0}$ où $F = \lambda x\lambda y(x)(\underline{s})y$, alors on peut vérifier que pour tout $\theta_n \simeq_\beta \underline{n}$, $(T_i)\theta_n f \succ_f (f)(\underline{s})^n\underline{0}$ ($i = 1$ ou 2) (voir [19] et [38]). D'où T_1 et T_2 sont des oprérateurs de mise en mémoire pour les entiers de Church.

J.-L. Krivine a démontré dans [19], qu'en utilisant une traduction de Gödel de la logique classique dans la logique intuitionniste, on peut trouver un type très simple pour les opérateurs de mise en mémoire. Mais le λ-terme obtenu τ_n peut contenir des variables substituées par des λ-termes $u_1, ..., u_m$ qui dépendent de θ_n. Comme τ_n est β-équivalent à $\underline{n}$, alors, la réduction gauche de $\tau_n[u_1/x_1, ..., u_m/x_m]$ est équivalente à celle de τ_n et les λ-termes $u_1, ..., u_m$ ne sont pas calculés durant la réduction.

Définition finale : Un λ-terme clos T est dit opérateur de mise en mémoire pour les entiers Church ssi pour tout $n \in \mathbb{N}$, il existe un λ-terme $\tau_n \simeq_\beta \underline{n}$, tel que pour tout $\theta_n \simeq_\beta \underline{n}$, il existe une substitution σ, telle que $(T)\theta_n f \succ_f (f)\sigma(\tau_n)$ (où f est une nouvelle variable).

Revenons sur l'exemple donné au début de ce paragraphe. On remplace la réduction gauche de $(F)\theta_n$, par celle de $(T)\theta_n F$. Comme $(T)\theta_n F = \{(T)\theta_n f\}[F/f]$, alors, on commence par réduire $(T)\theta_n f$ à sa forme normale de tête qui est $(f)\sigma(\tau_n)$, et on réduit ensuite $(F)\sigma'(\tau_n)$ ($\sigma' = \gamma \circ \sigma$ où $\gamma(f) = F$ et $\gamma(x) = x$ si $x \neq f$). T a donc mis en mémoire le résultat τ_n avant de le donner, le nombre de fois qu'il faut, pour n'importe quelle fonction.

Les opérateurs de mise en mémoire les plus efficaces pour les entiers de Church donnent comme résultat $(\underline{s})^n\underline{0}$. Une question se pose : *Peut-on trouver des opérateurs de mise en mémoire pour les entiers de Church qui donnent comme résultats les formes normales des entires ?* Ce type d'opérateurs est appelé opérateurs forts de mise en mémoire.

J'ai démontré dans [33] le résultat suivant :

Théorème 3.1.1 *L'ensemble des entiers de Church ne possède pas d'opérateurs forts de mise en mémoire.*

Ceci provient essentiellement de deux faits :

- *L'infinité des entiers* : En effet, on peut démontrer que chaque ensemble fini d'entiers de Church possède un oprérateur fort de mise en mémoire.

- *La représentation des entiers* : En effet, on peut démontrer qu'on ne peut pas créer un entier de Church $\underline{n}$ ($n \geq 1$) durant une réduction de tête dans une application. Si on change la représentation des entiers, on peut trouver un opérateur fort de mise en mémoire. Pour tout $n \in \mathbb{N}$, on définit $\overline{n}$ par induction : $\overline{0} = \lambda x\lambda f x$ et $\overline{n+1} = \lambda x\lambda f(f)\overline{n}$. Ces entiers sont dits les entiers récursifs.

Soit $\overline{s} = \lambda n \lambda x \lambda f(f) n$; il est facile de vérifier que $\overline{s}$ est un λ-terme pour le successeur. Si on prend $T' = \lambda\nu(\nu)\tau\rho\rho$ où $\tau = \lambda f(f)\overline{0}$, $\rho = \lambda y \lambda z(G)(y)z\tau z$, et $G = \lambda x \lambda y(x)\lambda z(y)\lambda f \lambda x(f)z$, alors, pour tout $\theta_n \simeq_\beta \overline{n}$, $(T')\theta_n f \succ_f (f)\overline{n}$. D'où T' est un oprérateur fort de mise en mémoire pour les entiers récursifs (voir [37]).

Les résultats précédents peuvent se généraliser pour les représentations itératives et récursives des données en λ-calcul (voir [33]).

3.2 Le λ-calcul dirigé

Rappelons qu'un λ-terme clos T est un opérateur de mise en mémoire pour les entiers de Church ssi pour tout $n \in \mathbb{N}$, il existe un λ-terme $\tau_n \simeq_\beta \underline{n}$, tel que pour tout $\theta_n \simeq_\beta \underline{n}$, il existe une substitution σ, telle que $(T)\theta_n f \succ_f (f)\sigma(\tau_n)$ (où f est une nouvelle variable). Analysons la réduction $(T)\theta_n f \succ_f (f)\sigma(\tau_n)$, en remplacant chaque λ-terme provenant de θ_n par une nouvelle variable. Ceci va nous aider à bien comprendre la preuve du théorème de J.-L. Krivine (théorème 4.1.1) et justifier l'introduction du λ-calcul dirigé.

Si $\theta_n \simeq_\beta \underline{n}$, alors $\theta_n \succ \lambda x \lambda g(g)t_{n-1}$, $t_{n-k} \succ (g)t_{n-k-1}$ $(1 \leq k \leq n-1)$, $t_0 \succ x$, et $t_k \simeq_\beta (g)^k x$ $(0 \leq k \leq n-1)$. Soit x_n une nouvelle variable qui représente θ_n. $(T)x_n f$ est résoluble et ses variables libres sont x_n et f, donc on a deux possibilités pour sa forme normale de tête : $(f)\delta$ (dans ce cas on s'arrête) ou $(x_n)a_1...a_m$. Supposons qu'on obtient $(x_n)a_1...a_m$. La variable x_n représente θ_n, et $\theta_n \succ \lambda x \lambda g(g)t_{n-1}$, donc $(\theta_n)a_1...a_m$ et $((a_2)t_{n-1}[a_1/x, a_2/g])a_3...a_m$ possèdent la même forme normale de tête. Le λ-terme $t_{n-1}[a_1/x, a_2/g]$ provient de θ_n. Soit x_{n-1,a_1,a_2} une nouvelle variable qui représente $t_{n-1}[a_1/x, a_2/g]$. Le λ-terme $((a_2)x_{n-1,a_1,a_2})a_3...a_m$ est résoluble et ses variables libres sont parmi x_{n-1,a_1,a_2}, x_n, et f, donc on a trois possibilités pour sa forme normale de tête : $(f)\delta$ (dans ce cas on s'arrête) ou $(x_n)b_1...b_r$ ou $(x_{n-1,a_1,a_2})b_1...b_r$. Supposons qu'on obtient $(x_{n-1,a_1,a_2})b_1...b_r$. La variable x_{n-1,a_1,a_2} représente $t_{n-1}[a_1/x, a_2/g]$, et $t_{n-1} \succ (g)t_{n-2}$, donc $(t_{n-1}[a_1/x, a_2/g])b_1...b_r$ et $((a_2)t_{n-2}[a_1/x, a_2/g])b_1...b_r$ possèdent la même forme normale de tête. Le λ-terme $t_{n-2}[a_1/x, a_2/g]$ provient de θ_n. Soit x_{n-2,a_1,a_2} une nouvelle variable qui représente $t_{n-2}[a_1/x, a_2/g]$. Le λ-terme $((a_2)x_{n-2,a_1,a_2})b_1...b_r$ est résoluble et ses variables libres sont parmi x_{n-2,a_1,a_2}, x_{n-1,a_1,a_2}, x_n, et f, donc on a quatre possibilités pour sa forme normale de tête : $(f)\delta$ (dans ce cas on s'arrête) ou $(x_n)c_1...c_s$ ou $(x_{n-1,a_1,a_2})c_1...c_s$ ou $(x_{n-2,a_1,a_2})c_1...c_s$ ... etc ... Supposons qu'on obtient $(x_{0,d_1,d_2})e_1...e_k$ durant la construction. La variable x_{0,d_1,d_2} représente $t_0[d_1/x, d_2/g]$, et $t_0 \succ x$, donc $(t_0[d_1/x, d_2/g])e_1...e_k$ et $(d_1)e_1...e_k$ possèdent la même forme normale de tête ; alors on démarre la construction avec le λ-terme $(d_1)e_1...e_k$.

Le λ-terme $(T)\theta_n f$ est résoluble, et possède $(f)\sigma(\tau)$ comme forme normale de tête, donc cette construction s'arrête toujours sur un $(f)\delta$. On peut prouver que $\delta \simeq_\beta \underline{n}$.

D'après la construction précédente, la réduction $(T)\theta_n f \succ_f (f)\sigma(\tau_n)$ est décomposée de deux parties : une réduction qui ne dépend pas de n et une autre qui dépend de n (et non de θ_n). Si on ajoute des nouvelles règles de réductions pour obtenir ces dernières (par exemple : $(x_n)a_1a_2 \succ_f (a_2)x_{n-1,a_1,a_2}$; $x_{i+1,a_1,a_2} \succ_f (a_2)u_{i,a_1,a_2}$ $(i > 0)$; $x_{0,a_1,a_2} \succ_f a_1$) on obtient une définition équivalente pour les opérateurs de mise en mémoire : un λ-terme clos T est un opérateur de mise en mémoire pour les entiers de Church ssi pour tout $n \in \mathbb{N}$, $(T)x_n f \succ_f (f)\delta_n$ où $\delta_n \simeq_\beta \underline{n}$. J.-L. Krivine a utilisé la condition suffisante de cette équivalence pour démontrer son théorème.

La notion d'opérateur de mise en mémoire peut être généralisée pour chaque ensemble de λ-termes normaux clos.

Soient t un λ-terme normal clos et T un λ-terme clos. On dit que T est un opérateur de mise en mémoire pour t ssi il existe un λ-terme $\tau_t \simeq_\beta t$, tel que pour tout λ-terme $\theta_t \simeq_\beta t$, il existe une substitution σ, telle que $(T)\theta_t f \succ_f (f)\sigma(\tau_t)$ (où f est une nouvelle variable). Soient $\mathcal{D}$ un ensemble de λ-termes normaux clos et T un λ-terme clos. On dit que T est un opérateur de mise en mémoire pour $\mathcal{D}$ ssi il en est un pour chaque t de $\mathcal{D}$.

Le λ-calcul dirigé est une extension du λ-calcul ordinaire dont le but est de suivre la trace d'un λ-terme normal durant une réduction de tête. Il permet de formaliser la définition équivalente pour les opérateurs de mise en mémoire donnée précédement.

Soit $\mathcal{V}$ l'ensemble de variables du λ-calcul pur. L'ensemble de termes du λ-calcul dirigé, noté $\Lambda[]$, est défini de la manière suivante :
- Si $x \in \mathcal{V}$, alors $x \in \Lambda[]$;
- Si $x \in \mathcal{V}$, et $u \in \Lambda[]$, alors $\lambda xu \in \Lambda[]$;
- Si $u, v \in \Lambda[]$, alors $(u)v \in \Lambda[]$;
- Si $t \in \Lambda$ est un λ-terme normal tel que $Fv(t) \subseteq \{x_1, ..., x_n\}$, et $a_1, ..., a_n \in \Lambda[]$, alors $[t] < a_1/x_1, ..., a_n/x_n > \in \Lambda[]$.

Un $\lambda[]$-terme de la forme $[t] < a_1/x_1, ..., a_n/x_n >$ est dit boite dirigée par t. Cette notation représente, intuitivement, le λ-terme t où toutes ses variables libres $x_1, ..., x_n$ sont remplacées par $a_1, ..., a_n$. La substitution $< a_1/x_1, ..., a_n/x_n >$ est notée $< \mathbf{a}/\mathbf{x} >$. Un $\lambda[]$-terme de la forme $(\lambda xu)v$ est appelé β-redex ; $u[v/x]$ est son réduit. Un $\lambda[]$-terme de la forme $[t] < \mathbf{a}/\mathbf{x} >$ est appelé $[]$-redex ; son réduit R est défini par induction sur t :
- Si $t = x_i$ $(1 \leq i \leq n)$, alors $R = a_i$;
- Si $t = x \neq x_i$ $(1 \leq i \leq n)$, alors $R = x$;
- Si $t = \lambda xu$, alors $R = \lambda y[u] < \mathbf{a}/\mathbf{x}, y/x >$ où $y \notin Fv(\mathbf{a})$;
- Si $t = (u)v$, alors $R = ([u] < \mathbf{a}/\mathbf{x} >)[v] < \mathbf{a}/\mathbf{x} >$.

En interprétant la boite $[t] < a_1/x_1, ..., a_n/x_n >$ par $t[[a_1/x_1, ..., a_n/x_n]]$ (le λ-terme t avec une substitution explicite), les nouvelles règles de réduction sont celles nécessaires pour effectuer la substitution. Ce type du λ-calcul a été étudié

par plusieurs personnes dans un but de mieux contrôler le processus de substitution crée par la β-reduction, et donc implémenter le λ-calcul. La différence essentielle entre ce type de calcul et le λ-calcul dirigé est que : le premier produit des substitutions explicites après chaque β-réduction. Le second "exécute" seulement les substitutions données à l'avance. On peut donc considérer le λ-calcul dirigé comme une restriction (l'interdiction de la production des substitutions explicites) d'un calcul avec substitutions explicites dont l'unique but d'étudier la réduction de tête.

Soit $u, v \in \Lambda[]$. La notation $u \succ_{\beta[]} v$ signifie que v est obtenu à partir de u par quelques réductions de tête faible.

On va énoncer, maintenant, le théorème qui donne une définition équivalente pour les opérateurs de mise en mémoire (voir [8]).

Théorème 3.2.1 *Soient t un λ-terme normal clos, et T un λ-terme clos. T est un opérateur de mise en mémoire pour t ssi il existe un λ-terme $\tau_t \simeq_\beta t$, tel que $(T)[t]f \succ_{\beta[]} (f)\tau_t[[t_1] < \mathbf{a_1}/\mathbf{x_1} > /y_1, ..., [t_m] < \mathbf{a_m}/\mathbf{x_m} > /y_m]$.*

Pour prouver la condition nécessaire on associe à chaque $\theta_t \simeq_\beta t$ une substitution spéciale S_θ sur les boites dirigées par des sous-termes de t telle que $S_\theta([t]) = \theta_t$ et satisfait la propriété suivante : si $u \succ_{\beta[]} v$ alors $S_\theta(u) \succ_f S_\theta(v)$. Donc $(T)\theta_t f \succ_f (f)\sigma(\tau_t)$. Pour la condition suffisante on utilise la méthode expliquée au début de ce paragraphe. L'unique difficulté est de prouver que $\tau_t \simeq_\beta t$. Pour cela on utilise l'indépendance de τ_t et θ_t.

Si t est un terme normalisable, alors on note $n(t)$ (resp. $N(t)$) le nombre de pas nécessaire pour trouver sa forme normale de tête (resp. sa forme normale).

Le théorème 3.2.2 permet de trouver quelques propriétés importantes des opérateurs de mise en mémoire (voir [8]).

Théorème 3.2.2 *1) Soient $\mathcal{D}$ un ensemble de λ-termes normaux clos, T et T' deux λ-termes clos. Si T est un opérateur de mise en mémoire pour $\mathcal{D}$, et $T' \simeq_\beta T$, alors T' est aussi un opérateur de mise en mémoire pour $\mathcal{D}$.*
2) L'ensemble d'opérateurs de mise en mémoire pour un ensemble de λ-termes normaux clos est non récursive. Mais l'ensemble d'opérateurs de mise en mémoire pour un ensemble fini de λ-termes normaux clos est récursivement énumérable.
3) Chaque ensemble fini de λ-termes normaux clos ayant des formes $\beta\eta$-normales distinctes possède un opérateur de mise en mémoire.
4) Soient t un λ-terme normal clos, et T un λ-terme clos. Si T est un opérateur de mise en mémoire pour t, alors il existe deux constantes $A_{T,t}$ et $B_{T,t}$, telles que pour chaque $\theta_t \simeq_\beta t$, $n((T)\theta_t f) \leq A_{T,t}.N(\theta_t) + B_{T,t}$.

3.3 Les S-opérateurs de mise en mémoire

Les constantes x_i et $x_{i,a,b,\overline{c}}$ introduites dans le paragraphe précédent représenent intuitivement des λ-termes provenant des entiers de Church non calculés. L'uniformité des entiers de Church permet la description du fonctionnement de ces constantes lorqu'elles arrivent en tête. Cependant, on a une discription plus élémentaires des entiers qui consiste à dire qu'un entier est soit égale à zéro soit égale à un successeur. D'où les définitions suivantes :

Soit $\{X_i\}_{i\geq 0}$ un ensemble de constantes distinctes. On définit un ensemble de termes (noté Λ_X) de la manière suivante :
- Si $x \in \mathcal{V} \bigcup \{X_i\}_{i\geq 0}$, alors $x \in \Lambda_X$;
- Si $x \in \mathcal{V}$, et $u \in \Lambda_X$, alors $\lambda x u \in \Lambda_X$;
- Si $u \in \Lambda_X$, et $v \in \Lambda_X$, alors $(u)v \in \Lambda_X$;
- Si $n \in \mathbb{N}$, et $a, b, \overline{c} \in \Lambda_X$, alors $X_{n,a,b,\overline{c}} \in \Lambda_X$.

$X_{n,a,b,\overline{c}}$ est considérée comme une constante qui ne figure pas dans les termes $a, b, \overline{c}$. Elle représente un entier non calculé de valeur n. Les termes de l'ensemble Λ_X sont appelés aussi λ_X-termes.

Soit $\underline{S}$ un λ-terme pour le successeur. Un λ-terme clos T est dit $\underline{S}$-opérateur de mise en mémoire pour les entiers de Church ssi pour tout $n \geq 0$, il existe un ensemble fini de réductions de tête faible $\{U_i \succ_f V_i\}_{1\leq i\leq r}$ tel que :
1) U_i et V_i sont des λ_X-termes ;
2) $U_1 = (T)X_n f$ et $V_r = (f)\tau_n$ où τ_n est un λ_X-terme β-équivalent à $\underline{n}$;
3) $V_i = (X_n)ab\overline{c}$ ou $V_i = (X_{l,a,b,\overline{c}})uv\overline{w}$ $(0 \leq l \leq n-1)$;
4) Si $V_i = (X_n)ab\overline{c}$, alors $U_{i+1} = (\underline{0})ab\overline{c}$ si $n = 0$ et $U_{i+1} = ((\underline{S})X_{n-1,a,b,\overline{c}})ab\overline{c}$ si $n \neq 0$;
5) Si $V_i = (X_{l,a,b,\overline{c}})uv\overline{w}$ $(0 \leq l \leq n-1)$, alors $U_{i+1} = (\underline{0})uv\overline{w}$ si $l = 0$ et $U_{i+1} = ((\underline{S})X_{l-1,u,v,\overline{w}})uv\overline{w}$ si $l \neq 0$.

Soit $\underline{S_1} = \lambda n\lambda x\lambda f(f)((n)f)x$ et $\underline{S_2} = \lambda n\lambda x\lambda f((n)f)(f)x$. Il est facile de vérifier que $\underline{S_1}$ et $\underline{S_2}$ sont deux successeurs pour les entiers de Church. On peut vérifier aussi que, pour tout $1 \leq i, j \leq 2$, T_i est un $\underline{S_j}$-opérateur de mise en mémoire.

En générale j'ai obtenu le résultat suivant (voir [42]) :

Théorème 3.3.1 *Si T est un opérateur de mise en mémoire pour les entiers de Church, alors, pour tout successeur $\underline{S}$, T est un $\underline{S}$-opérateur de mise en mémoire.*

La réciproque n'est pas toujours vraie. En effet, j'ai obtenu aussi les deux théorèmes suivants (voir [42]) :

Théorème 3.3.2 *T est $\underline{S_1}$-opérateur de mise en mémoire ssi T est un opérateur de mise en mémoire pour les entiers de Church.*

Soit $T_S = \lambda x(x)\ a\ b\ \underline{0}\ \underline{S}$ où $a = \lambda x\lambda y\lambda z(z)x$,
$b = \lambda x\lambda y\lambda z((x)(z)(x)\mathbf{id}\ \mathbf{id}\ \lambda x\underline{0})\lambda x(\underline{S})(z)x$, et $\underline{S}$ un successeur.

Théorème 3.3.3 *T_S est $\underline{S_2}$-opérateur de mise en mémoire mais pas un opérateur de mise en mémoir pour les entiers de Church.*

Chapitre 4

Opérateurs de mise en mémoire typés

4.1 Le théorème de J.-L. Krivine

Si on essaye de typer les opérateurs de mise en mémoire pour les entiers de Church, on trouve naturellement la formule $\forall x\{N[x] \to \neg\neg N[x]\}$ qui ne les caractérise pas (il suffit de prendre $T = \lambda\nu\lambda f(f)\nu$) car il ne force pas l'indépendance de τ_n et θ_n. Pour résoudre ce problème, il faut empêcher l'utilisation du premier $N[x]$ de $\forall x\{N[x] \to \neg\neg N[x]\}$ ainsi que ses sous-types pour avoir le second. On remplace donc le premier $N[x]$ par une formule $N'[x]$ telle que : pour tout $n \in \mathbb{N}$, $\vdash_{\mathcal{AF}2} \underline{n} : N'[s^n(0)]$ et les sous-types "actifs" de $N'[x]$ se terminent pour un symbole qui ne figure pas dans $N[x]$.

Pour chaque formule F de $\mathcal{AF}2$, on note F^g la formule obtenue en mettant $\neg$ devant chaque formule atomique F (F^g est appelée la traduction de Gödel de F). Par exemple : $N^g[x] = \forall X\{\neg X(0), \forall y(\neg X(y) \to \neg X(sy)) \to \neg X(x)\}$. J.-L. Krivine a démontré dans [19] que si une formule F est démontrable en logique classique, alors sa traduction F^g est démontrable en logique intuitionniste.

On peut vérifier que $\vdash_{\mathcal{AF}2} T_i : \forall x\{N^g[x] \to \neg\neg N[x]\}$ ($i = 1$ ou 2). Et, d'une manière générale, J.-L. Krivine a démontré le résultat suivant (voir [19]) :

Théorème 4.1.1 *Si $\vdash_{\mathcal{AF}2} T : \forall x\{N^g[x] \to \neg\neg N[x]\}$, alors T est un opérateur de mise en mémoire pour les entiers de Church.*

J.-L. Krivine a donné une preuve sémantique de ce théorème utilisant le lemme d'adéquation du système $\mathcal{AF}2$ et une bonne interprétation du $\bot$. J'en ai donné dans [31] une preuve purement syntaxique utilisant le λ-calcul dirigé.

Les opérateurs de mise en mémoire que nous avons présentés jusqu'à maintenant donnent tous des λ-termes clos comme résultat. Ce genre d'opérateurs est

appelé opérateurs propres de mise en mémoire. Une question se pose : *Peut on trouver des opérateurs non propres de mise en mémoire typables dans le système $\mathcal{AF}2$?*

J'ai démontré dans [30] que :

Théorème 4.1.2 *Il existe des opérateurs non propres de mise en mémoire pour les entiers de Church de type* $\forall x\{N^g[x] \to \neg\neg N[x]\}$.

Un exemple de tels opérateurs est le suivant : $T = \lambda\nu(\nu)\gamma D$ où
$D = \lambda u\lambda v(u)\lambda w(((\nu)\lambda y(((y)w)u)v)\lambda xx)\lambda g\lambda k\lambda l(l)\lambda n\lambda m(n)((g)n)m$,
$\gamma = \lambda f(((\nu)\lambda x(f)((((x)n)f)\underline{0})\lambda xx)\lambda x\lambda y\lambda zz$.

J'ai caractérisé dans [30] une classe d'opérateurs propres de mise en mémoire pour les entiers de Church par des types du système de typage simple.

Le théorème 4.1.1 se généralise pour tous les types de données courants : booléen, liste, arbre, produit, somme de types de données, ...

4.2 Généralisations

4.2.1 Les types $\forall^+$ et les transformations de Gödel

Si on analyse la preuve syntaxique du théorème 4.1.1, on remarque que les deux propriétés essentielles du type $N[x]$ utilisées dans la preuve sont les suivantes :
– La position du quantificateur du second ordre dans $N[x]$.
– Le fait que les sous-types "actifs" de $N^g[x]$ se terminent pas $\perp$.

Ceci m'a permis de généraliser ce résultat à d'autres types du système $\mathcal{AF}2$.

On définit deux ensembles de types du système $\mathcal{AF}2$ ($\forall^+$ et $\forall^-$) de la manière suivante :
- Si A est un type atomique, alors $A \in \forall^+$, et $A \in \forall^-$.
- Si $A \in \forall^+$, et $B \in \forall^-$, alors, $B \to A \in \forall^+$, et $A \to B \in \forall^-$.
- Si $A \in \forall^+$ (resp. $A \in \forall^-$), alors $\forall xA \in \forall^+$ (resp. $\forall xA \in \forall^-$).
- Si $A \in \forall^+$, alors $\forall XA \in \forall^+$.
- Si $A \in \forall^-$, et X n'est pas libre dans A, alors $\forall XA \in \forall^-$.
Donc, A est $\forall^+$ (resp. $\forall^-$) ssi le quantificateur universel du second ordre est positif (resp. négatif) dans A.

À chaque variable de relation X, on associe un ensemble fini non vide de variables de relations $V_X = \{X_1, ..., X_n\}$ de même arité que X tel que : si $X \neq Y$, alors $V_X \bigcap V_Y = \emptyset$. À chaque variable de relation X d'arité m on associe une formule $F_X[x_1, ..., x_m]$ qui se termine par $\perp$ dont les variables libres sont parmi $x_1, ..., x_m$ et celle de V_X. Pour chaque formule A, on définit la formule A^* de la manière suivante :

- Si $A = \bot$, alors $A^* = A$.
- Si $A = X(t_1, ..., t_m)$, alors $A^* = F_X[t_1/x_1, ..., t_m/x_m]$.
- Si $A = B \to C$, alors $A^* = B^* \to C^*$.
- Si $A = \forall x B$, alors $A^* = \forall x B^*$.
- Si $A = \forall X B$, alors $A^* = \forall X_1 ... \forall X_n B^*$ où $V_X = \{X_1, ..., X_n\}$.

A^* est dite la transformée de Gödel de A.

Soient T un λ-terme clos, et D, E deux types clos du système $\mathcal{AF}2$. On dit que T est un opérateur de mise en mémoire pour le couple de types (D, E) ssi pour tout λ-terme $\vdash_{\mathcal{AF}2} t : D$, il existe deux λ-termes τ_t et τ'_t, tels que $\tau'_t \simeq_\beta \tau_t$, $\vdash_{\mathcal{AF}2} \tau'_t : E$, et pour tout $\theta_t \simeq_\beta t$, il existe une substitution σ, telle que $(T)\theta_t f \succ_f (f)\sigma(\tau_t)$ (où f est une nouvelle variable).

Il est clair qu'un opérateur de mise en mémoire pour les entiers de Church est un opérateur de mise en mémoire pour le couple de types $(N[s^n(0)], N[s^n(0)])$ pour tout $n \in \mathbb{N}$.

J.-L. Krivine et moi même avons démontré dans [22] et [38] le résultat suivant :

Théorème 4.2.1 *Soient D, E deux types clos et $\forall^+$ du système $\mathcal{AF}2$ tels que E ne contient $\bot$. Si $\vdash_{\mathcal{AF}2} T : D^* \to \neg\neg E$, alors T est un opérateur de mise en mémoire pour le couple de types (D, E).*

La preuve syntaxique que j'ai faite repose sur les lemmes 4.2.1 et 4.2.2 ainsi que sur une caractérisation très tecnhique (utilisant le λ-calcul dirigé) des λ-termes de type $D^* \to \neg\neg E$.

Lemme 4.2.1 *Soient $A_1, ..., A_n$ des types qui se terminent par $\bot$, A un type $\forall^+$ qui ne contient pas $\bot$, et t un λ-terme normal. Si $x_1 : A_1, ..., x_n : A_n \vdash_{\mathcal{AF}2} t : A$, alors $\vdash_{\mathcal{AF}2} t : A$.*

Lemme 4.2.2 *Soient A un type $\forall^+$ et t un λ-terme normal. Si $\vdash_{\mathcal{AF}2} t : A$, alors $\vdash_{\mathcal{AF}2} t : A^*$.*

La condition "D, E sont $\forall^+$" est nécessaire pour avoir le résultat précédent ; en effet, soit $D = \forall X\{\forall Y(Y \to X) \to X\}$, $t = \lambda x(x)\lambda y y$, et $T = \lambda\nu(\nu)\lambda x\lambda f(f)\lambda y(y)x$. D n'est pas un type $\forall^+$. Il est facile de vérifier que $\vdash_{\mathcal{AF}2} t : D$, $\vdash_{\mathcal{AF}2} T : D^g \to \neg\neg D$, et T n'est pas un opérateur de mise en mémoire pour le couple (D, D).

4.2.2 Le système $\mathcal{TTR}$

Le système de typage $\mathcal{AF}2$ est satisfaisant du point de vue extensionel ; on peut construire des programmes pour toute fonction dont la preuve de terminaison se fait en arithmétique de Peano du second ordre. Mais du point de vue intensionel la situation est très différente : on ne peut pas toujours obtenir le programme le plus simple (en terme de complexité). Par exemple, on ne peut pas trouver un λ-terme de type $\forall x \forall y\{N[x], N[y] \to N[min(x, y)]\}$ (min est un symbole de

fonction binaire défini par des équations) dans le système $\mathcal{AF}2$ qui calcule le minimum de deux entiers de Church en temps $O(min)$ [1]. Le système $\mathcal{TTR}$ de M. parigot est une extension du système $\mathcal{AF}2$ basée sur des définitions récursives des types, ayant comme objectif de résoudre le problème d'efficacité cité plus haut.

Soit X une variable ou un symbole de relation, et A un type de $\mathcal{AF}2$. On définit les notions "X est positive dans A" et "X est négative dans A" par induction :
- Si X n'apparait pas dans A, alors X est positive et négative dans A.
- Si $A = X(t_1, ..., t_n)$, alors X est positive dans A, et X n'est pas négative dans A.
- Si $A = B \to C$, alors X est positive (resp. négative) dans A ssi X est négative (resp. positive) dans B, et X est positive (resp. négative) dans C.
- Si $A = \forall v B$, et $v \neq X$, alors X est positive (resp. négative) dans A ssi X est positive (resp. négative) dans B.

On ajoute au calcul des prédicats du second ordre un nouveau symbole logique μ, et on étend l'ensemble de formules avec la construction suivante : si A est une formule, C un symbole de prédicat n-aire qui apparait positivement dans A, $x_1, ..., x_n$ des variables du premier ordre, et $t_1, ..., t_n$ des termes, alors $\mu C x_1 ... x_n A < t_1, ..., t_n >$ est une formule appelée le plus petit point fixe de A en C calculé sur les termes $t_1, ..., t_n$.

On étend les notions "X est positive dans un type" et "X est négative dans un type" de la manière suivante : X est positive (resp. négative) dans $\mu C x_1 ... x_n A < t_1, ..., t_n >$ ssi X est positive (resp. négative) dans A. Et on étend la définition de la substitution en supposant que $C, x_1, ..., x_n$ sont liés dans la formule $\mu C x_1 ... x_n A < t_1, ..., t_n >$.

On définit sur les formules une relation binaire $\subseteq$ par : $A \subseteq B$ ssi elle est obtenue par les règles suivantes :

$(ax)\ A \subseteq A$ $\qquad$ $(\to)\ \dfrac{A \subseteq A' \quad B \subseteq B'}{A' \to B \subseteq A \to B'}$

$(\forall i_g)\ \dfrac{A[G/v] \subseteq B}{\forall v A \subseteq B}$ (1) $\qquad$ $(\forall i_d)\ \dfrac{A \subseteq B}{A \subseteq \forall v B}$ (2)

$(e)\ \dfrac{A \subseteq B[v/y]}{A \subseteq B[w/y]}$ (3) $\qquad$ $(tr)\ \dfrac{A \subseteq D \quad D \subseteq B}{A \subseteq B}$

$(\mu_d)\ D[\mu C x_1 ... x_m D < z_1, ..., z_m > / C(z_1, ..., z_m)][t_1/x_1, ..., t_m/x_m] \subseteq \mu C x_1 ... x_m D < t_1, ..., t_m >$

$(\mu'_g)\ \mu C x_1 ... x_m D < t_1, ..., t_m > \subseteq$

[1] R. David a donné dans [7] un λ-terme de type $N, N \to N$ dans le système $\mathcal{F}$ qui calcul le minimum de deux entiers de Church en temps $O(min.Log(min))$. La notion d'opérateur de mise en mémoire joue un rôle important dans sa construction.

$D[\mu C x_1...x_m D < z_1,...,z_m > /C(z_1,...,z_m)][t_1/x_1,...,t_m/x_m]$

$$(\mu_g)\ \frac{D[E/C(x_1,...,x_m)] \subseteq E}{\mu C x_1...x_m D < t_1,...,t_m > \subseteq E[t_1/x_1,...,t_m/x_m]}$$

(1) G est une formule ou un terme.
(2) v n'est pas libre dans A.
(3) $v = w$ est un cas particulier d'une équation.

(μ_d) et (μ'_g) sont les règles de factorisation et de développement d'un point fixe, et (μ_g) exprime le fait que $\mu C x_1...x_m D < t_1,...,t_m >$ est un plus petit point fixe.

Les règles de typage du système $\mathcal{TTR}$ (pour Théorie des Types Récursifs) sont les suivantes :
- Les règles de typage (1),...,(8) du système $\mathcal{AF}2$.
- $(\subseteq)\ \dfrac{\Gamma \vdash_{\mathcal{TTR}} t : A \quad A \subseteq B}{\Gamma \vdash_{\mathcal{TTR}} t : B}$.
- $(Y)\ \dfrac{\Gamma \vdash_{\mathcal{TTR}} t : \forall x_1...\forall x_m[C(x_1,...,x_m) \to E] \to \forall x_1...\forall x_m[D \to E]}{\Gamma \vdash_{\mathcal{TTR}} (Y)t : \forall x_1...\forall x_m[\mu C x_1...x_m D < x_1,...,x_m > \to E]}$

où C n'est pas libre dans E et G, et Y est le point fixe de Turing.

La règle (Y) exprime aussi le fait que $\mu C x_1...x_m D < t_1,...,t_m >$ est un plus petit point fixe.

Le système $\mathcal{TTR}$ possède les propriétés suivantes (voir [52]) :

Théorème 4.2.2 *1) Si $\Gamma \vdash_{\mathcal{TTR}} t : A$, et $t \to_\beta t'$, alors $\Gamma \vdash_{\mathcal{TTR}} t' : A$ (conservation de type).*
2) Si $\Gamma \vdash_{\mathcal{TTR}} t : A$ sans utiliser la règle (Y), alors t est fortement normalisable (normalisation forte).
3) Si $\Gamma \vdash_{\mathcal{TTR}} t : A$, et tous les points fixes de A sont positifs, alors t est normalisable (normalisation faible).

Le type des entiers récursifs est la formule $N^r[x] = \mu C z[\forall X\{X(0), \forall y(C(y) \to X(sy)) \to X(z)\}] < x >$. On peut démontrer que, pour tout $n \in \mathbb{N}$, $\overline{n}$ est le seul λ-terme normal clos de type $N^r[s^n(0)]$.

On étend la définition de l'ensemble $\forall^+$ en supposant que : si $A \in \forall^+$, $x_1,...,x_n$ sont des variables d'individu, $t_1,...,t_n$ dont des termes, C est un symbole de relation n-aire qui est positif dans A, alors $\mu C x_1...x_n A < t_1,...,t_n > \in \forall^+$. On étend aussi la définition de la transformationde Gödel en posant :
$(\mu C x_1...x_n A < t_1,...,t_n >)^* = \mu C x_1...x_n A^* < t_1,...,t_n >$.

Soient T un λ-terme clos, et D, E deux types clos du système $\mathcal{TTR}$. On dit que T est un opérateur de mise en mémoire pour le couple de types (D, E) ssi pour tout λ-terme $\vdash_{\mathcal{TTR}} t : D$, il existe deux λ-termes τ_t et τ'_t, tels que

$\tau'_t \simeq_\beta \tau_t$, $\vdash_{\mathcal{TTR}} \tau'_t : E$, et pour tout $\theta_t \simeq_\beta t$, il existe une substitution σ, telle que $(T)\theta_t f \succ_f (f)\sigma(\tau_t)$ (où f est une nouvelle variable).

Soient $T'_1 = (Y)H$ où $H = \lambda x\lambda y((y)\delta)\lambda z(G)(x)z$, $G = \lambda x\lambda y(x)\lambda z(y)(\overline{s})z$, et $\delta = \lambda f(f)\overline{0}$; $T'_2 = \lambda\nu(\nu)\tau\rho\rho$ où $\tau = \lambda d\lambda f(f)\overline{0}$, et $\rho = \lambda y\lambda z(G)(y)z\tau z$,
alors, pour tout $\theta_n \simeq_\beta \overline{n}$, $(T_i)\theta_n f \succ_f (f)(\overline{s})^n\overline{0}$ (voir [37]).
Donc, pour tout $n \in \mathbb{N}$, T'_1 et T'_2 sont des opérateurs de mise en mémoire pour le couple $(N^r[s^n(0)], N^r[s^n(0)])$.

J'ai obtenu le résultat suivant (voir [37]) :

Théorème 4.2.3 *Soient D, E deux types clos et $\forall^+$ du système $\mathcal{TTR}$ tels que E ne contient $\bot$. Si $\vdash_{\mathcal{TTR}} T : D^* \to \neg\neg E$, alors T est un opérateur de mise en mémoire pour le couple de types (D, E).*

La preuve de ce théorème est très tecnhique.

4.2.3 Le système $\mathcal{AF}2_\bot$

Avec les types proposés dans le paragraphe 4.2.1, on ne peut pas typer le simple opérateur de mise en mémoire suivant : $T = \lambda\nu\lambda f((\nu)(T_i)\nu f)\lambda x x$ ($i = 1$ ou 2). Ceci provient du fait que la forme normale de T contient une occurrence de la variable ν appliquée à deux arguments et une autre occurrence appliquée à trois arguments. Donc si la variable ν est déclarée de type $N^*[x]$, le nombre d'arguments de ν sera fixé une fois pour toute. Pour résoudre ce problème, on remplace $N^*[x]$ dans le type des opérateurs de mise en mémoire par un autre qui ne limite pas le nombre d'arguments de ν et permet de déduire uniquement des formules qui se terminent par $\bot$.

On suppose que pour tout entier n, on a un ensemble de variables du second ordre n-aires spéciales notées : $X_\bot, Y_\bot, Z_\bot$...., et appelées $\bot$-variables. Un type A est dit $\bot$-type ssi A se termine par $\bot$ ou par une $\bot$-variable.

On ajoute au système $\mathcal{AF}2$ les deux règles suivantes :

$(6')$ $\dfrac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X_\bot A}$ (*) $\qquad\qquad$ $(7')$ $\dfrac{\Gamma \vdash t : \forall X_\bot A}{\Gamma \vdash t : A[G/X_\bot]}$ (**)

(*) $X_\bot$ ne figure pas dans Γ et (**) G est un $\bot$-type.

On appelle $\mathcal{AF}2_\bot$ le nouveau système de typage.

À chaque variable de relation X, on associe une $\bot$- variable $X_\bot$. Pour toute formule A du système $\mathcal{AF}2$, on définit la formule $A^\bot$ de la manière suivante :
- Si $A = \bot$, alors $A^\bot = A$;
- Si $A = X(t_1, ..., t_n)$, où X est une variable de relation n-aire, alors $A^\bot = X_\bot(t_1, ..., t_n)$;

- Si $A = B \to C$, alors $A^\perp = B^\perp \to C^\perp$;
- Si $A = \forall x B$, alors $A^\perp = \forall x B^\perp$;
- Ssi $A = \forall X B$, alors $A^\perp = \forall X_\perp B^\perp$.

J'ai obtenu la généralisation suivante (voir [35]) :

Théorème 4.2.4 *Soient D, E deux types clos et $\forall^+$ du système $\mathcal{AF}2$ tels que E ne contient $\perp$. Si $\vdash_{\mathcal{AF}2_\perp} T : D^\perp \to \neg\neg E$, alors T est un opérateur de mise en mémoire pour le couple de types (D, E).*

Je signale enfin que C. Raffalli a obtenu dans [57] un résultat sur les opérateurs de mise en mémoire pour tous les types du système $\mathcal{AF}2$.

Chapitre 5

Types de données

Les résultats des paragraphes 5.1 et 5.2 ont été obtenu avec S. Farkh dans le cadre de sa thèse de doctorat.

5.1 Types de données complets du système $\mathcal{AF}2$

Le lemme d'adéquation de système $\mathcal{AF}2$ stipule que pour une relation S applicable qui vérifie $(\lambda xu)v\ S\ u[v/x]$, $|A|_S$ contient l'ensemble des λ-termes de type A. Nous cherchons des classes de types pour lesquelles nous avons la "réciproque" qui est bien entendu fausse en général. Ces types (où la sémantique des parties S-saturées est complète) sont appelés types S-complets.

On dit qu'un type clos A est $\simeq_{\beta\eta}$-complet si, pour tout λ-terme t, $t \in |A|_{\simeq_{\beta\eta}}$ ssi il existe un λ-terme clos t' tel que $t \simeq_{\beta\eta} t'$ et $\vdash_{\mathcal{AF}2} t' : A$.

Le résultat suivant est dû à Labib-Sami (voir [25]).

Théorème 5.1.1 *Si A est un type $\forall^+$ clos du système $\mathcal{F}$, alors A est $\simeq_{\beta\eta}$-complet.*

Nous avons généralisé ce résultat au système $\mathcal{AF}2$ (voir [12]) :

Théorème 5.1.2 *Si A est un type $\forall^+$ clos du système $\mathcal{AF}2$, alors A est $\simeq_{\beta\eta}$-complet.*

Nous obtenons donc le corollaire suivant :

Corollaire 5.1.1 *Soient A un type $\forall^+$ clos du système $\mathcal{AF}2$, et t un λ-terme. Si $t \in |A|_{\simeq_{\beta\eta}}$, alors t est normalisable et β-équivalent à un λ-terme clos.*

Je vais donner brièvement l'idée de la démonstration du théorème 5.1.2.

Soient $\Omega = \{x_i / i \in \mathbb{N}\}$ une énumération d'un ensemble infini de variables du λ-calcul, et $\{A_i / i \in \mathbb{N}\}$ une énumération des types $\forall^-$ de $\mathcal{AF}2$, où chaque type $\forall^-$ se répète une infinité de fois. On définit alors l'ensemble $\Gamma^- = \{x_i : A_i / i \in \mathbb{N}\}$. Soit u un λ-terme, tel que $Fv(u) \subseteq \Omega$. On définit le contexte Γ^-_u comme étant la restriction de Γ^- aux déclarations de variables de $Fv(u)$. La notation $\Gamma^- \vdash_{\mathcal{AF}_2} u : B$ exprime que $\Gamma^-_u \vdash_{\mathcal{AF}_2} u : B$. On pose $\Gamma^- \vdash^{\beta\eta}_{\mathcal{AF}_2} u : B$ ssi il existe un λ-terme u', tel que $u \simeq_{\beta\eta} u'$ et $\Gamma^- \vdash_{\mathcal{AF}_2} u' : B$.

On définit un $\Lambda_{\simeq_{\beta\eta}}$-modèle $\mathcal{M}$ de la façon suivante :
– L'ensemble de base est $|\mathcal{M}| = \dfrac{M_0}{\approx_E}$ (M_0 l'ensemble de tous les termes), on note $\overline{a}$ la classe d'équivalence d'un élément a de $|\mathcal{M}|$;
– L'ensemble adéquat $R = P_{\simeq_{\beta\eta}}(\Lambda)$;
– L'interprétation de chaque symbole de fonction n-aire f est l'application $f_{\mathcal{M}}$ de $|\mathcal{M}|^n$ dans $|\mathcal{M}|$ définie par $f_{\mathcal{M}}(\overline{a_1}, ..., \overline{a_n}) = \overline{f(a_1, ..., a_n)}$;
– L'interprétation de chaque symbole de relation n-aire P est l'application $P_{\mathcal{M}}$ de $|\mathcal{M}|^n$ dans $P_{\simeq_{\beta\eta}}(\Lambda)$ définie par $P_{\mathcal{M}}(\overline{a_1}, ..., \overline{a_n}) = \{\tau \in \Lambda : \Gamma^- \vdash^{\beta\eta}_{\mathcal{AF}_2} \tau : P(a_1, ..., a_n)\}$.

On définit une interprétation particulière $\mathcal{I}$ sur les variables en posant :
– $\mathcal{I}(x) = \overline{x}$;
– $\mathcal{I}(X) = \Phi_X$, où Φ_X est l'application de $|\mathcal{M}|^n$ dans $P_{\simeq_{\beta\eta}}(\Lambda)$ définie par $\Phi_X(\overline{a_1}, ..., \overline{a_n}) = \{\tau \in \Lambda : \Gamma^- \vdash^{\beta\eta}_{\mathcal{AF}_2} \tau : X(a_1, ..., a_n)\}$.

Le lemme suivant (qui se démontre par induction simultanée sur les types $\forall^+$ et $\forall^-$) permet de déduire le théorème 5.1.2.

Lemme 5.1.1 *Soient S une formule et τ un λ-terme.*
(i) Si S est $\forall^+$, et $\tau \in |S|_{\mathcal{M},\mathcal{I}}$, alors $\Gamma^- \vdash^{\beta\eta}_{\mathcal{AF}_2} \tau : S$.
(ii) Si S est $\forall^-$, et $\Gamma^- \vdash^{\beta\eta}_{\mathcal{AF}_2} \tau : S$, alors $\tau \in |S|_{\mathcal{M},\mathcal{I}}$.

La condition $\forall^+$ est nécessaire pour avoir le théorème 4.1.2. En effet, soient $D = \forall X\{\forall Y(Y \to X) \to X\}$, $t = \lambda x(x)(\delta)\delta$, où $\delta = \lambda x(x)x$. On peut vérifier facilement que $t \in |D|_{\simeq_{\beta\eta}}$, mais t n'est pas normalisable.

Nous avons trouvé un raffinement supplémentaire du théorème 5.1.2. Nous commençons par définir une classe de type notée $\mathcal{B}^+$.

On définit de la façon suivante les types positifs (resp. négatifs), notés en abrégé B^+ (resp. B^-) :
– Une formule atomique est B^+ (resp. B^-) ;
– Si A est B^+ (resp. B^- qui ne commence pas par un quantificateur du premier ordre) et B est B^- (resp. B^+), alors $B \to A$ est B^+ (resp. B^-) ;
– Si A est B^+, et X une variable de relation qui figure dans A, alors $\forall X A$ est B^+ ;

– Si A est B^-, alors $\forall x A$ est B^-.

On définit sur les types de $\mathcal{AF}2$ les deux relations binaires $<'$ et $\sim'$ de la manière suivante :
$\forall x A <' A[u/x]$, si u est un terme du langage ;
$\forall X A <' A[F/X(x_1, ..., x_n)]$, si F est une formule du langage ;
$A \sim' B$ ssi $A = C[u/x]$, $B = C[v/x]$, et $u = v$ est un cas particulier d'une équation.
Soient $\leq$ et $\sim$ les clôtures réflexives et transitives respectives de $<'$ et $\sim'$. On note $A < B$ ssi $A \leq B$ et $A \neq B$.

On définit sur les types de $\mathcal{AF}2$ une relation binaire s de la façon suivante : EsF ssi $F = E[\boldsymbol{t}/\boldsymbol{x}]$, où $\boldsymbol{t}$ (resp. $\boldsymbol{x}$) est une suite finie de termes (resp. de variables du premier ordre). On définit la relation $\sim_s$ comme étant la plus petite relation d'équivalence contenant l'union des relations s et $\sim$.

On dit qu'un type B se détruit strictement avec un type C ssi $B < G$, $G \sim C_n$, et $C \leq C_1 \to (C_2 \to (... \to (C_n \to D)...))$.

On dit qu'un type B^+ (resp. B^-) A satisfait la condition (*) si, lorsque B et C sont deux sous-types négatifs (resp. positifs) de A, modulo $\sim_s$, alors B ne se détruit pas strictement avec C.

On dit qu'un type A est un bon type positif (resp. bon type négatif), noté en abrégé $\mathcal{B}^+$ (resp. $\mathcal{B}^-$) s'il est B^+ (resp. B^-) et satisfait la condition (*).

Théorème 5.1.3 *La classe des types $\mathcal{B}^+$ contient les types de données de J.-L. Krivine.*

Pour trouver une classe intéressante de types $\succ_f$-complets, on est amené à donner la définition suivante :

On dit qu'un type clos A est $\succ_f$-complet si, pour tout λ-terme t, $t \in |A|_{\succ_f}$ ssi il existe un λ-terme clos t' tel que $t \to_\beta t'$ et $\vdash_{\mathcal{AF}2} t' : A$.

Nous avons démontré le théorème suivant (voir [12]) :

Théorème 5.1.4 *Si A est un type $\mathcal{B}^+$ clos du système $\mathcal{AF}2$, alors A est $\succ_f$-complet.*

Nous obtenons donc :

Corollaire 5.1.2 *Soient A un type $\mathcal{B}^+$ clos du système $\mathcal{AF}2$, et t un λ-terme. Si $t \in |A|_{\succ_f}$, alors t est normalisable et se réduit par β-réduction à un terme clos.*

Nous avons obtenu aussi le résultat important suivant (voir [12]) :

Corollaire 5.1.3 *Si A est un type $\mathcal{B}^+$ clos du système $\mathcal{AF}2$, alors $|A|_{\succ_f}$ est stable par β-équivalence (i.e si $t \in |A|_{\succ_f}$ et $t \simeq_\beta t'$, alors $t' \in |A|_{\succ_f}$).*

On ne sait pas si le résultat précédent est valable pour tous les types.

La preuve du théorème 5.1.4 provient essentiellement de deux faits :
– Les types $\mathcal{B}^+$ se conservent durant une η-réduction (théoréme 5.1.5).
– Une généralisation du lemme d'adéqation (théoréme 5.1.6).

Théorème 5.1.5 *Soient A un type $\mathcal{B}^+$ et τ un λ-terme clos.*
Si $\vdash_{\mathcal{AF}2} \tau : A$ et $\tau \rightarrow_{\beta\eta} \tau'$, alors $\vdash_{\mathcal{AF}2} \tau' : A$.

Théorème 5.1.6 *Soient M un $\Lambda_{\succ_f}$-modèle d'un système d'équation E, et I une interprétation. Si $x_1 : B_1, ..., x_n : B_n \vdash_{\mathcal{AF}2} t' : A$, $t \simeq_\beta t'$, et $u_i \in |B_i|_{M,I}$ $(1 \leq i \leq n)$, alors $t[u_1/x_1, ..., u_n/x_n] \in |A|_{M,I}$.*

Nous avons démontré que les conditions qui définissent un type $\mathcal{B}^+$ sont toutes nécessaires pour avoir le théorème 5.1.4.

Il est clair qu'un type A du système $\mathcal{F}$ est $\mathcal{B}^+$ ssi A est $\forall^+$ et propre (c.à.d. pour tout sous-type $\forall XB$ de A la variable X est libre dans B). D'autre part un type est $\succ_f$-complet dans $\mathcal{AF}2$ ssi il est $\succ_f$-complet dans $\mathcal{F}$.

D'où le résultat suivant :

Théorème 5.1.7 *Si A est un type $\forall^+$ clos et propre du système $\mathcal{F}$, alors A est $\succ_f$-complet.*

On a d'autres classes de types $\succ_f$-complets. En effet, si on considére le type $S = \forall X\{\forall Y(Y \rightarrow X) \rightarrow Id\}$, on démontre que $t \in |S|_{\succ_f}$ ssi $t \rightarrow_\beta \lambda x\mathbf{id}$ et $\vdash_{\mathcal{F}} \lambda x\mathbf{id} : S$.

Le point le plus important pour avoir le théorème 5.1.4 est la conservation des types par η-réduction. Pour cette raison, nous allons ajouter des règles de typage au système $\mathcal{AF}2$ pour avoir cette propriété afin de démontrer que tous les types $\forall^+$ sont $\succ_f$-complets.

On définit sur les formules de $\mathcal{AF}2$ une relation binaire notée $\subseteq$ par : $A \subseteq B$ ssi elle est obtenue à partir de règles de démonstrations suivantes :

$A \subseteq A$

$\forall\xi(C \rightarrow D) \subseteq \forall\xi C \rightarrow \forall\xi D$

$$\frac{C' \subseteq C \quad D \subseteq D'}{C \rightarrow D \subseteq C' \rightarrow D'}$$

$$\frac{A \subseteq \forall \xi C}{A \subseteq C[F/\xi]}$$

$$\frac{A \subseteq D}{A \subseteq \forall \xi D}, \ \xi \text{ n'est pas libre dans } A$$

$$\frac{A \subseteq D \quad D \subseteq B}{A \subseteq B}$$

$$\frac{A \subseteq D[u/y]}{A \subseteq D[v/y]}, \ u = v \text{ est un cas particulier d'une équation de } E$$

On décrit un système de λ-calcul typé noté $\mathcal{AF}2_{\subseteq}$ dont les types sont les formules du langage, comme étant une extension du système $\mathcal{AF}2$ auquel on ajoute la règle de typage suivante :

$$(\subseteq) \quad \frac{\Gamma \vdash_{\mathcal{AF}2_{\subseteq}} t : A \quad A \subseteq B}{\Gamma \vdash_{\mathcal{AF}2_{\subseteq}} t : B}$$

Le système de typage $\mathcal{AF}2_{\subseteq}$ possède les propriétés suivantes (voir [10]) :

Théorème 5.1.8 *1) Si $\Gamma \vdash_{\mathcal{AF}2_{\subseteq}} t : A$, et $t \rightarrow_{\beta\eta} t'$, alors $\Gamma \vdash_{\mathcal{AF}2_{\subseteq}} t' : A$ (théorème de conservation de type).*
2) Si $\Gamma \vdash_{\mathcal{AF}2_{\subseteq}} t : A$, alors t est fortement normalisable (théorème de normalisation forte).

Nous avons démontré aussi que (voir [10]) :

Théorème 5.1.9 *Si $\Gamma \vdash_{\mathcal{AF}2_{\subseteq}} t : A$, alors il existe un λ-terme u tel que $u \rightarrow_{\eta} t$ et $\Gamma \vdash_{\mathcal{AF}2} u : A$.*

On dit qu'un type A est $\succ'_f$-complet si, pour tout terme t, $t \in |A|_{\succ_f}$ ssi il existe un λ-terme clos t' tel que $t \rightarrow_{\beta} t'$ et $\vdash_{\mathcal{AF}2_{\subseteq}} t' : A$.

Nous avons démontré le résultat suivant (voir [10]) :

Théorème 5.1.10 *Si A est un type $\forall^+$ clos et propre du système $\mathcal{F}$, alors A est $\succ'_f$-complet.*

5.2 Types de données syntaxiques du système $\mathcal{F}$

Nous avons essayé de trouver une définition syntaxique à un type de données dans le système $\mathcal{F}$. Nous l'avons défini comme étant un type entrée et sortie.

Type entrée : Une machine doit être capable de tester si les entrées sont bien typées, c'est à dire le problème de typage d'un type entrée doit être décidable. On définit donc un type entrée comme étant un type dont toutes ses démonstrations se font dans une restriction décidable (notée $\mathcal{F}_0$) du système $\mathcal{F}$.

Type sortie : Si E, S sont deux types du système $\mathcal{F}$, et t un λ-terme clos tel que $\vdash_{\mathcal{F}} t : E \to S$, alors t peut être vu comme un programme qui à un élément de $\Lambda(E)$ (l'ensemble des λ-termes de type E) associe un élément de $\Lambda(S)$. Pour nous, la sortie doit dépendre de l'entrée. Les seuls programmes (fonctions) qui ne tiennent pas compte de leurs arguments (entrées) sont les fonctions constantes (i.e les λ-termes de la forme λxt où t est un terme clos). En formalisant les types qui vérifient cette propriété, nous avons obtenu la définition suivante : un type sortie est un type clos S tel que si $\vdash \lambda xt : \forall X(X \to S)$ (t est un λ-terme normal), alors x est non libre dans t.

Soit O une constante de type. La définition précédente est équivalente à la suivante :

Un type clos S (ne contenant pas la constante O) est un type sortie ssi pour tout λ-terme normal t, si $\alpha : O \vdash_{\mathcal{F}} t : S$ alors $\alpha \notin Fv(t)$.

On peut démontrer facilement que les types Id, B et N sont des types sorties mais le type $N \to N$ ne l'est pas. Nous avons trouvé une grande classe des types sorties (voir [13]).

Théorème 5.2.1 *Si S est un type $\forall^+$, alors S est un type sortie.*

La réciproque de ce théorème n'est pas en général vraie. En effet il suffit de considérer le type $S = \forall X\{\forall Y(Y \to X) \to Id\}$.

Nous avons obtenu la généralisation suivante (voir [13]) :

Théorème 5.2.2 *Un type clos S est un type sortie ssi pour tout λ-terme normal t et pour tous types A_1,..., A_r qui se terminent par O, si $x_1 : A_1, ..., x_r : A_r \vdash_{\mathcal{F}} t : S$, alors, pour tout $1 \leq i \leq r$, $x_i \notin Fv(t)$.*

Si A et B sont deux types du système $\mathcal{F}$, alors le type $A \wedge B = \forall X\{(A \to (B \to X)) \to X\}$ est dit type produit de A et B, le type $A \vee B = \forall X\{(A \to X) \to ((B \to X) \to X)\}$ est dit type somme disjointe de A et B, et le type $LA = \forall X\{X \to [(A \to (X \to X)) \to X]\}$ est dit type liste d'objet de A.

En utilisant le théorème 5.2.2, nous déduisons (voir [13]) :

Théorème 5.2.3 *Si A et B sont des types sorties, alors $A \wedge B$, $A \vee B$, et LA sont des types sorties.*

J.-L. Krivine a défini les types de données du système $\mathcal{F}$ comme étant les types clos A tels que $|A| \neq \emptyset$ et tout terme $t \in |A|$ se réduit par β-réduction à un terme clos. Il a montré dans [19] que les types Id, B et N sont des types de données.

Nous avons obtenu le résultat suivant (voir [13]).

Théorème 5.2.4 *Si A est un type de données, alors A est un type sortie.*

On définit le système $\mathcal{F}_0$ comme étant le système $\mathcal{F}$ sans la règle d'élimination du quantifiateur. Il est facile de voir que le problème de typabilité d'un λ-terme normal dans le système $\mathcal{F}_0$ est décidable.

Un type clos E du système $\mathcal{F}$ est dit entrée s'il vérifie la propriété suivante : si t est un λ-terme normal tel que $\vdash_{\mathcal{F}} t : E$, alors $\vdash_{\mathcal{F}_0} t : E$.

On peut démontrer facilement que les types Id, B et N sont des types entrées. Nous avons trouvé une grande classe des types entrées (voir [13]).

Théorème 5.2.5 *Si A est un type $\forall^+$, alors A est un type entrée.*

On dit qu'un type clos D du système $\mathcal{F}$ est un type de données syntaxique s'il est à la fois un type entrée et sortie.

D'après les théorèmes 5.2.1 et 5.2.5 on a :

Théorème 5.2.6 *Si D est un type clos $\forall^+$ du système $\mathcal{F}$, alors D est un type de données syntaxique.*

Nous avons démontré le résultat suivant (voir [13]) :

Théorème 5.2.7 *Si E est un type entrée du système $\mathcal{F}$, alors E est un type sortie.*

Donc les types de données syntaxiques sont exactement les types sorties. Ceci nous a permis de déduire le résultat suivant.

Théorème 5.2.8 *Si A et B sont des types entrées, alors $A \wedge B$, $A \vee B$, et LA sont des types entrées.*

Le théorème 5.2.7 nous a conduit à regarder la réciproque. Nous l'avons démontré dans des cas particuliers où nous imposons des restrictions sur la règle d'élimination du quantificateur. On définit le système $\mathcal{F}_F$ comme étant le système $\mathcal{F}$ où on remplace la règle d'élimination du quantificateur par la règle :

$$\frac{\Gamma \vdash_{\mathcal{F}_F} t : \forall X A}{\Gamma \vdash_{\mathcal{F}_F} t : A[G/X]}$$

où $A = \forall \boldsymbol{X_0}(A_1 \to \forall \boldsymbol{X_1}(A_2 \to ... \to \forall \boldsymbol{X_{n-1}}(A_n \to \forall \boldsymbol{X_n} Y)...))$, avec $Y = X$ ou l'un des A_i se termine par X.

Alors nous avons le résultat suivant (voir [13]) :

Théorème 5.2.9 *E est un type sortie dans le système $\mathcal{F}_F$ ssi E est un type entrée.*

On ne sait pas si le résultat précédent est vrai dans le système $\mathcal{F}$.

5.3 Les I-types du système $\mathcal{F}$

L'ensemble des λI-termes (noté ΛI) est défini par induction :
– Si x est une variable, alors $x \in \Lambda I$.
– Si $u, v \in \Lambda I$, alors $(u)v \in \Lambda I$.
– Si $u \in \Lambda I$ et x est libre dans u, alors $\lambda x u \in \Lambda I$.

Un λ-terme que n'est pas un λI-terme est dit λK-terme.

Le λI-calcul possède les propriétés suivantes (voir [2]).

Théorème 5.3.1 *1) Si $t \in \Lambda I$, et $t \rightarrow_{\beta\eta} t'$, alors $t' \in \Lambda I$ et $Fv(t) = Fv(t')$.*
2) Un λI-terme est fortement normalisable ssi il est faiblement normalisable.
3) On peut programmer en λI-calcul toutes les fonctions récursives partielles.

Les types du système $\mathcal{F}$ qui représentent les types de données courants sont tous habités par au moins un λK-terme. Une question se pose : Peut-on représenter les types de données courants par des types du système $\mathcal{F}$ habités uniquement par des λI-termes ?

Un type clos D du système $\mathcal{F}$ est dit I-type ssi si t est un λ-terme clos β-normal tel que $\vdash_{\mathcal{F}} t : D$, alors t est un λI-terme.

J'ai obtenu le résultat suivant (voir [47]) :

Théorème 5.3.2 *Si D est un I-type démontrable du système $\mathcal{F}$, alors $D \in \forall^+$.*

La démonstration de ce résultat se fait en deux étapes :

* *La première étape est très technique* : On démontre qui si la règle d'élimination de $\forall$ est utilisée sur un sous-terme qui ne commence pas par λ dans un typage d'un λI-terme normal clos de type D, alors il existe un λI-terme normal t contenant une variable libre α tel que $\alpha : O \vdash_{\mathcal{F}} t : D$ (O est une constante de type). On déduit alors que le λ-terme $t[\mathbf{0}/\alpha]$ est normal (car α est en postion d'argument) de type D et ce n'est pas un λI-terme.

* *La deuxième étape est plus simple* : On démontre que s'il existe un λI-terme normal clos de type D sans utiliser la règle d'élimination de $\forall$ sur un de ses sous-termes qui ne commence pas par λ, alors D est $\forall^+$.

On déduit donc que les I-types démontrables du système $\mathcal{F}$ sans quasiment ceux du système des types simples.

J'ai donné dans [47] la liste de tous les I-types démontrables contenant au plus deux quantificateurs.

Théorème 5.3.3 *Si D est un I-type démontrable du système $\mathcal{F}$ contenant un seul quantificateur, alors $D = \forall X\{X \rightarrow X\}$.*

Si B, A sont des types, alors, pour tout $n \geq 1$, on note $B^n \rightarrow A$ le type $B, ..., B \rightarrow A$ où B est répété n fois. Si X et Y sont deux variables de types, alors on note les formules $\forall X \forall Y A$ et $\forall Y \forall X A$ par $\forall X, Y A$.

Théorème 5.3.4 *Si D est un I-type démontrable du système $\mathcal{F}$ contenant deux quantificateurs, alors $D = \forall X, Y\{[(Y \rightarrow Y)^n \rightarrow X] \rightarrow X\}$, $D = \forall X\{(\forall Y(Y \rightarrow Y) \rightarrow X) \rightarrow X\}$, $D = \forall X, Y\{Y, (Y^n \rightarrow X) \rightarrow X\}$ ou $D = \forall X, Y\{(Y^n \rightarrow X), Y \rightarrow X\}$.*

Si D est un λ-terme clos, alors on note $\Lambda(D) = \{t \ \beta\eta\text{-normal clos} \ / \vdash_{\mathcal{F}} t : D\}$.

Corollaire 5.3.1 *Si D est un I- type du système $\mathcal{F}$ contenant au plus deux quantificateurs, alors $card(\Lambda(D)) \leq 1$.*

Ceci m'a conduit à conjecturer l'énoncé suivant :

Conjecture : *Si D est un I-type du système $\mathcal{F}$, alors $card(\Lambda(D)) \leq 1$.*

Chapitre 6

Systèmes numériques

6.1 Systèmes numériques en λ-calcul pur

Un système d'entiers est une suite $\mathbf{d} = d_0, d_1, ..., d_n, ...$ de λ-termes clos $\beta\eta$-normaux différents.

Un λ-terme clos S_d est dit successeur pour $\mathbf{d}$ ssi : $(S_d)d_n \simeq_\beta d_{n+1}$ pour tout $n \in \mathbb{N}$. Un λ-terme clos P_d est dit prédécesseur pour $\mathbf{d}$ ssi : $(P_d)d_{n+1} \simeq_\beta d_n$ pour tout $n \in \mathbb{N}$. Un λ-terme clos Z_d est dit test à zéro pour $\mathbf{d}$ ssi : $(Z_d)d_0 \simeq_\beta \mathbf{1}$ et $(Z_d)d_{n+1} \simeq_\beta \mathbf{0}$ pour tout $n \in \mathbb{N}$.

Avec la terminologie de H. Barendregt (voir [2]) un système d'entiers qui possède un successeur et un test à zéro est appelé système numérique et un système numérique qui possède un prédécesseur est appelé système numérique adéquat. Signalons aussi que l'une des différences entre nos définitions et celles de H. Barendregt est le fait d'imposer aux λ-termes d_i d'être normaux et distincts. En effet, ces conditions permettent de trouver la valeur exacte d'une fonction numérique calculée sur des entiers.

Chaque système d'entiers peut être considéré comme un codage des entiers en λ-calcul et donc on peut représenter les fonctions numériques totales de la manière suivante :

Une fonction numérique partielle $\phi : \mathbb{N}^p \to \mathbb{N}$ est dite λ-définissable dans le système d'entiers $\mathbf{d}$ ssi il existe un λ-terme clos F_ϕ tel que pour tout $n_1, ..., n_p \in \mathbb{N}$:
$(F_\phi)d_{n_1}...d_{n_p} \simeq_\beta d_{\phi(n_1,...,n_p)}$ si $\phi(n_1, ..., n_p)$ est défini ;
$(F_\phi)d_{n_1}...d_{n_p}$ est non résoluble si $\phi(n_1, ..., n_p)$ est non défini.

H. Barendregt a démontré dans [2] le résultat suivant :

Théorème 6.1.1 *Un système d'entiers* $\mathbf{d}$ *est un système numérique adéquat ssi toutes les fonctions numériques récursives partielles sont* λ*-définissables dans* $\mathbf{d}$.

On ne trouve pas dans la littérature (notament dans [2]) des exemples des systèmes numériques non adéquats. Ceci m'a amené à démontrer les trois théorèmes suivants (voir [41]) :

Pour tout $n \in \mathbb{N}$, on note $a_n = \lambda x_1 ... \lambda x_n \mathbf{id}$ et $\mathbf{a} = a_0, a_1, ...$

Théorème 6.1.2 **a** *est un système d'entiers qui possède un successeur et un prédécesseur mais ne possède pas un test à zéro.*

Soient $b_0 =< \mathbf{1}, \mathbf{id} >$ et pour tout $n \geq 1$, $b_n =< \mathbf{0}, a_{n-1} >$.

Théorème 6.1.3 **b** *est un système d'entiers qui possède un successeur et un test à zéro mais ne possède pas un prédécesseur.*

Soit **e** une suite de λ-termes normaux clos pour laquelle il n'existe pas de terme clos A tel que : $(A)\mathbf{id} \simeq_\beta e_1$ et $(A) < ... << \mathbf{id}, e_1 >, e_2 >, ..., e_n >) \simeq_\beta e_{n+1}$ pour tout $n \geq 1$. Soit $c_0 = \mathbf{id}$ et pour tout $n \geq 1$, $c_n =< c_{n-1}, e_n >$.

Théorème 6.1.4 **c** *est un système d'entiers qui possède un prédécesseur et un test à zéro mais ne possède pas un successeur.*

Ces trois théorèmes montrent que les fonctions successeur, prédécesseur et test à zéro sont indépendantes. Je conjecture l'énoncé suivant :

Conjecture : *Il n'existe pas de fonctions récursives unaires f, g telles que : pour tout système d'entiers* **d**, *f, g sont λ-définissables ssi toutes les fonctions récursives sont λ-définissables dans* **d**.

Si on autorise les fonctions binaires, on obtient le résultat suivant (voir [41]) :

Soit k la fonction binaire définie par : $k(n,m) = \begin{cases} n+1 & \text{si } m = 0 \\ |n-m| & \text{si } m \neq 0 \end{cases}$.

Théorème 6.1.5 *Pour tout système d'entiers* **d**, *k est λ-définissable ssi toutes les fonctions récursives sont λ-définissables dans* **d**.

6.2 Systèmes numériques typables

Soient **d** un système d'entiers et O_d un λ-terme clos. On dit que O_d est un opérateur de mise en mémoire pour **d** ssi pour tout $n \in \mathbb{N}$, il existe un λ-terme $\tau_n \simeq_\beta d_n$, tel que, pour tout $\theta_n \simeq_\beta d_n$, il existe une substitution σ telle que $(O_d)\theta_n f \succ_f (f)\sigma(\tau_n)$ (où f est une nouvelle variable).

J'ai obtenu le résultat suivant (voir [44]) :

Théorème 6.2.1 *Chaque système numérique adéquat possède un opérateur de mise en mémoire.*

E. Tronci a conjecturé l'énoncé suivant :

Conjecture : *Un système numérique est adéquat s'il possède un opérateur de mise en mémoire.*

J'ai réussi à donner une réponse négative à cette conjecture mais uniquement pour les systèmes numériques typables dans le système $\mathcal{F}$.

Un système numérique typable est une paire $\mathcal{D} =< D, \mathbf{d} >$ où D est un type clos du système $\mathcal{F}$, et $\mathbf{d} = d_0, d_1, ..., d_n, ...$ est une suite de λ-termes $\beta\eta$- normaux clos tels que :
– Si t est un λ-terme normal, alors $\vdash_{\mathcal{F}} t : D$ ssi il existe $i \in \mathbb{N}$ tel que $t = d_i$.
– Il existe des λ-termes clos S_d et Z_d tels que :
$\vdash_{\mathcal{F}} S_d : D \to D$ et $(S_d)d_n \simeq_\beta d_{n+1}$ pour tout $n \in \mathbb{N}$;
$\vdash_{\mathcal{F}} Z_d : D \to B$ et $(Z_d)d_n \simeq_\beta \begin{cases} \mathbf{1} & \text{si } n = 0 \\ \mathbf{0} & \text{si } n \geq 1 \end{cases}$.

Les λ-termes S_d et Z_d sont appelés successeur et test à zéro pour $\mathcal{D}$.

Un système numérique typé $\mathcal{D}$ est dit adéquat ssi il existe un λ-terme clos P_d tel que $\vdash_{\mathcal{F}} P_d : D \to D$ et $(P_d)d_{n+1} \simeq_\beta d_n$ pour tout $n \in \mathbb{N}$. Le λ-terme P_d est appelé prédécesseur pour $\mathcal{D}$.

On a vu que, dans le système de typage $\mathcal{F}$, le type $N^g \to \neg\neg N$ convient pour les opérateurs de mise en mémoire pour le système numérique de Church. D'où la définition suivante :

Soit $\mathcal{D} =< D, \mathbf{d} >$ un système numérique typable tel que pour tout λ-terme clos t, si $\vdash_{\mathcal{F}} t : D$, alors $\vdash_{\mathcal{F}} t : D^g$. Soit O_d un λ-terme clos. On dit que O_d est un opérateur de mise en mémoire pour $\mathcal{D}$ ssi $\vdash_{\mathcal{F}} O_d : D^g \to \neg\neg D$, et pour tout $n \in \mathbb{N}$, il existe un λ-terme $\tau_n \simeq_\beta d_n$ et $\vdash_{\mathcal{F}} \tau_n : D$ tel que, pour tout $\theta_n \simeq_\beta d_n$, il existe une substitution σ telle que $(O_d)\theta_n f \succ (f)\sigma(\tau_n)$ (où f est une nouvelle variable).

Soient $P = \forall X \forall Y\{((X \to Y) \to X) \to X\}$ (la loi de Peirce), $Q = P \to \forall X X$ et $D = (Q \to P) \to N$.

Le lemme suivant signifie qu'on ne peut pas utiliser le type D pour montrer B (voir [44]).

Lemme 6.2.1 *Soit t un λ-terme normal clos. $\vdash_{\mathcal{F}} t : D \to B$ ssi $t = \lambda\alpha\mathbf{0}$ ou $t = \lambda\alpha\mathbf{1}$.*

Soit $E = \forall X\{(B, D \to X), X \to X\}$.

Le lemme suivant caractérise les λ-termes de type E (voir [44]).

Lemme 6.2.2 *Soit t un λ-terme normal clos. $\vdash_{\mathcal{F}} t : E$ ssi $t = \mathbf{0}$) ou il existe $n \in \mathbb{N}$ tel que $t = \lambda x\lambda y(x)\mathbf{b}\lambda\alpha\underline{n}$ où $b \in \{0, 1\}$.*

Soit $\mathbf{e}$ le système d'entiers défini par : $e_0 = \mathbf{0}$, $e_{2n+1} = \lambda x\lambda y(x)\mathbf{0}\lambda\alpha\underline{n}$ $(n \geq 0)$, et $e_{2n+2} = \lambda x\lambda y(x)\mathbf{1}\lambda\alpha\underline{n}$ $(n \geq 0)$.

Les lemmes 6.2.1 et 6.2.2 permettent de démontrer le résultat suivant (voir [44]) :

Théorème 6.2.2 *$\mathcal{E} =< E, \mathbf{e} >$ est un système numérique typable non adéquat qui possède un opérateur de mise en mémoire.*

Chapitre 7

Logiques classiques

7.1 Le $\lambda\mathcal{C}$-calcul

Je vais commencer par présenter le système de typage classique de J.-L. Krivine.

On ajoute une constante $\mathcal{C}$ au λ-calcul pur et on note $\Lambda\mathcal{C}$ l'ensemble des termes obtenus (appelés aussi $\lambda\mathcal{C}$-termes). On considère les règles de réduction de tête faible suivantes (appelées les $\mathcal{C}$-réductions) :
1) $(\lambda xu)tt_1...t_n \rightarrow (u[t/x])t_1...t_n$ pour tout $u, t, t_1, ..., t_n \in \Lambda\mathcal{C}$.
2) $(\mathcal{C})tt_1...t_n \rightarrow (t)\lambda x(x)t_1...t_n$ pour tout $t, t_1, ..., t_n \in \Lambda\mathcal{C}$, x est une λ-variable qui ne parait pas dans $t_1, ..., t_n$.
Pour tous $\lambda\mathcal{C}$-termes t, t', on écrit $t \succ_{\mathcal{C}} t'$ si t' est obtenu à partir de t en appliquant ces règles un nombre fini de fois. On peut définir sur les $\lambda\mathcal{C}$-termes les notions de la β-réduction et de la β-équivalence. Un $\lambda\mathcal{C}$-terme t est dit β-normal ssi t ne contient pas de β-redex. Un $\lambda\mathcal{C}$-terme t est dit $\mathcal{C}$-résoluble ssi sa $\mathcal{C}$-réduction termine.

On ajoute au système de typage $\mathcal{AF}2$ la règle suivante :
(0) $\Gamma \vdash \mathcal{C} : \forall X\{\neg\neg X \rightarrow X\}$.
Cette règle axiomatise la logique classique au dessus de la logique intuitionniste. On appelle $\mathcal{C}2$ le nouveau système de typage. Il est clair que $\Gamma \vdash_{\mathcal{C}2} t : A$ ssi $\Gamma, \mathcal{C} : \forall X\{\neg\neg X \rightarrow X\} \vdash_{\mathcal{AF}2} t : A$.

Le système de typage $\mathcal{C}2$ possède les propriétés suivantes (voir [23]) :

Théorème 7.1.1 *1) Si* $\Gamma \vdash_{\mathcal{C}2} t : A$, *et* $t \rightarrow_\beta t'$, *alors* $\Gamma \vdash_{\mathcal{C}2} t' : A$.
2) Si $\Gamma \vdash_{\mathcal{C}2} t :\bot$, *et* $t \succ_{\mathcal{C}} t'$, *alors* $\Gamma \vdash_{\mathcal{C}2} t' :\bot$.
3) Si A *est un type atomique, et* $\Gamma \vdash_{\mathcal{C}2} t : A$, *alors* t *est* $\mathcal{C}$*-résoluble.*

Dans ce système on perd la propriété de l'unicité de la représentation des données.

Soit $n \in \mathbb{N}$. Un entier classique θ_n est un $\lambda\mathcal{C}$-terme clos tel que $\vdash_{\mathcal{C}2} \theta_n : N[s^n(0)]$. On dit aussi que θ_n est un entier classique de valeur n.

Soit $\theta_1 = \lambda x\lambda f(\mathcal{C})\lambda y(y)(f)(\mathcal{C})\lambda z(y)(f)x$. On peut vérifier facilement que θ_1 est un entier classique de valeur 1 qui ne se comporte pas comme un entier de Church. En effet, si g, a, x_0, x_1 et x_2 sont des variables, alors :
$(\theta_1)agx_0 \succ_{\mathcal{C}} (g)t_1x_0$ où $t_1 = (\mathcal{C})\lambda z(\lambda x(x)x_0)(g)a$;
$(t_1)x_1 \succ_{\mathcal{C}} (g)t_2x_0$ où $t_2 = a$;
$(t_2)x_2 \succ_{\mathcal{C}} (a)x_2$.

D'une manière générale, J.-L. Krivine a démontré dans [23] le théorème suivant :

Théorème 7.1.2 *Soient $n \in \mathbb{N}$, θ_n un entier classique de valeur n, et x, g des variables distinctes.*
- Si $n = 0$, alors, pour toute variable v, on a : $(\theta_n)xgv \succ_{\mathcal{C}} (x)v$.
- Si $n \neq 0$, alors il existe $m \in \mathbb{N}$, et une application $I : \{0, ..., m\} \to \mathbb{N}$, tels que pour toutes variables $v_0, v_1, ..., v_m$, on a :
$(\theta_n)xgv_0 \succ_{\mathcal{C}} (g)t_1v_{r_0}$; $(t_i)v_i \succ_{\mathcal{C}} (g)t_{i+1}v_{r_i}$ $(1 \leq i \leq m-1)$; $(t_m)v_m \succ_{\mathcal{C}} (x)v_{r_m}$
où $I(0) = n$, $I(r_m) = 0$, et $I(i+1) = I(r_i) - 1$ $(0 \leq i \leq m-1)$.

Le théorème précédent permet de trouver la valeur d'un entier classique. Soient θ_n un entier classique de valeur n et v, g, x des variables distinctes. Si $(\theta_n)xgv \succ_{\mathcal{C}} (x)v$, alors $n = 0$. Sinon, il existe $m \in \mathbb{N}$, et une suite $(r_i)_{1 \leq i \leq m}$ où $(0 \leq r_i \leq m)$ et une application $J : \{0, ..., m\} \to \mathbb{N}$ tels que $J(0) = 0$, et $J(i+1) = J(r_i) + 1$ $(0 \leq i \leq m-1)$. D'où $J(r_m) = n$.

J'ai généralisé, par des méthodes syntaxiques, ce dernier résultat.

Soient $\mathcal{V}$ l'ensemble de variable du $\lambda\mathcal{C}$-calcul et $\mathcal{P}$ un ensemble infini de constantes appelées constantes de pile[1]. On définit un ensemble de $\lambda\mathcal{C}$-termes $\Lambda\mathcal{CP}$ par :
- Si $x \in \mathcal{V}$, alors $x \in \Lambda\mathcal{CP}$;
- Si $t \in \Lambda\mathcal{CP}$, et $x \in \mathcal{V}$, alors $\lambda xt \in \Lambda\mathcal{CP}$;
- Si $t \in \Lambda\mathcal{CP}$, et $u \in \Lambda\mathcal{CP} \bigcup \mathcal{P}$, alors $(t)u \in \Lambda\mathcal{CP}$.
Autrement dit, $t \in \Lambda\mathcal{CP}$ ssi les constantes de pile de t ne sont pas en position d'argument.
Soit σ une fonction définie sur $\mathcal{V} \bigcup \mathcal{P}$ telle que :
- Si $x \in \mathcal{V}$, alors $\sigma(x) \in \Lambda\mathcal{CP}$.
- Si $p \in \mathcal{P}$, alors $\sigma(p) = (t_1, ..., t_n)$, $n \geq 0$, $t_i \in \Lambda\mathcal{CP} \bigcup \mathcal{P}$ $(1 \leq i \leq n)$.
On définit $\sigma(t)$ pour tout $t \in \Lambda\mathcal{CP}$ par :
- $\sigma((u)v) = (\sigma(u))\sigma(v)$ si $v \notin \mathcal{P}$.
- $\sigma(\lambda xu) = \lambda x\sigma(u)$.
- $\sigma((t)p) = (t)t_1...t_n$ si $\sigma(p) = (t_1, ..., t_n)$.
σ est appelée $\mathcal{P}$-substitution.
On définit sur l'ensemble $\Lambda\mathcal{CP}$, les règles de réduction de tête faible suivantes :
(1) $(\lambda xu)tt_1...t_n \to (u[t/x])t_1...t_n$ pour tout $u, t \in \Lambda\mathcal{CP}$ et $t_1, ..., t_n \in \Lambda\mathcal{CP} \bigcup \mathcal{P}$;

[1] La notion des constantes de pile a été introduite par J.-L. Krivine.

(2) $(\mathcal{C})tt_1...t_n \rightarrow (t)\lambda x(x)t_1...t_n$ pour tout $t \in \Lambda\mathcal{CP}$ et $t_1, ..., t_n \in \Lambda\mathcal{CP} \bigcup \mathcal{P}$, et x est une λ-variable qui ne parait pas dans $t_1, ..., t_n$.
Pour tout $t, t' \in \Lambda\mathcal{CP}$, on écrit $t \succ_{\mathcal{CP}} t'$, si t' est obtenu à partir de t en appliquant un nombre fini de fois les règles précédentes.

Le lemme suivant montre l'utilité des constantes de pile. Il signifie que durant une $\mathcal{CP}$-réduction, on peut remplacer ces constantes par n'importe quelles suites finies de $\lambda\mathcal{C}$-termes (voir [36]) :

Lemme 7.1.1 *Si $t \succ_{\mathcal{CP}} t'$, alors pour toute $\mathcal{P}$-substitution σ, $\sigma(t) \succ_{\mathcal{CP}} \sigma(t')$.*

J'ai obtenu la caractérisation suivante des entiers classiques (voir [36]) :

Théorème 7.1.3 *Soient $n \in \mathbb{N}$, θ_n un entier classique de valeur n, et x, g des variables distinctes.*
- Si $n = 0$, alors, pour toute constante de pile p, on a : $(\theta_n)xgp \succ_{\mathcal{CP}} (x)p$.
- Si $n \neq 0$, alors il existe $m \in \mathbb{N}$, et une application $I : \{0, ..., m\} \rightarrow \mathbb{N}$, tels que pour toutes constantes de pile $p_0, p_1, ..., p_m$, on a :
$(\theta_n)xgp_0 \succ_{\mathcal{CP}} (g)t_1p_{r_0}$; $(t_i)p_i \succ_{\mathcal{CP}} (g)t_{i+1}p_{r_i}$ $(1 \leq i \leq m-1)$; $(t_m)p_m \succ_{\mathcal{CP}} (x)p_{r_m}$ *où $I(0) = n$, $I(r_m) = 0$, et $I(i+1) = I(r_i) - 1$ $(0 \leq i \leq m-1)$.*

J.-L. Krivine a démontré dans [23] que des opérateurs de mise en mémoire typés dans le système $\mathcal{AF}2$ peuvent être utilisés pour trouver les valeurs des entiers classiques.

Théorème 7.1.4 *Si $\vdash_{\mathcal{AF}2} T : \forall x\{N^g[x] \rightarrow \neg\neg N[x]\}$, alors pour tout $n \in \mathbb{N}$, il existe un $\lambda\mathcal{C}$-terme $\tau_n \simeq_\beta \underline{n}$, tel que pour tout entier classique θ_n de valeur n, il existe une substitution σ, telle que $(T)\theta_n f \succ_{\mathcal{C}} (f)\sigma(\tau_n)$ (où f est une nouvelle variable). En particulier $(T)\theta_n\lambda xx \succ_{\mathcal{C}} \sigma(\tau_n) \rightarrow_\beta \underline{n}$.*

Comme on a vu précédement, les entiers classiques peuvent se comporter dans un calcul d'une manière assez compliquée. Si on analyse les raisons pour lesquelles on a ce genre de phénomème, on s'apercoit que ceci provient essentiellement du fait que l'absurde, qui représente dans ce système le type du calcul achevé, ne contient pas d'information sur la partie de l'entier que l'on calcul. Pour avoir cette dernière propriété, on ajoute un symbole de relation unaire $\perp$ pour l'absurde et pour toute formule A, et pour tout entier n, on note $\neg_n A$ la formule $A \rightarrow \perp(s^n(0))$. On remplace la règle de typage (0) par la règle suivante :
(0′) Si $\Gamma \vdash t : \neg_n\neg_n A$ pour tout $n \in \mathbb{N}$, alors $\Gamma \vdash (\mathcal{C})t : A$.
On écrit $\Gamma \vdash_{\mathcal{CF}2} t : A$ si dans ce système t est de type A dans le context Γ.

La règle (0′) peut être interprétée de la façon suivante : On considére $\neg_n A$ comme étant la formule A est fausse à l'instant n. De ce fait, la règle signifie qu'on peut affirmer qu'une formule est vraie si elle n'est pas fausse à chaque instant. On peut appeler cette logique "la logique persistante".

On garde les règles de réduction de tête faible du $\lambda\mathcal{C}$-calcul.

J'ai démontré dans [34] que dans ce système de typage les entiers classiques se comportent exactement comme les entiers de Church.

Théorème 7.1.5 *Soient* $n \in \mathbb{N}$, *et* $\vdash_{\mathcal{CF}2} \theta_n : N[s^n(0)]$, *alors pour toutes constantes de pile* $p_0, p_1, ..., p_n$ *:* $(\theta_n)xgp_0 \succ_{\mathcal{CP}} (g)t_1p_0$ *;* $(t_i)p_i \succ_{\mathcal{CP}} (g)t_{i+1}p_i$ $(1 \leq i \leq n-1)$ *et* $(t_n)p_n \succ_{\mathcal{CP}} (x)p_n$.

J'ai obtenu dans [32] deux résultats supplémentaires sur le système $\mathcal{C}2$.

Soit $\mathcal{A}$ (pour Abort) le $\lambda\mathcal{C}$-terme $\lambda x(\mathcal{C})\lambda yx$. Il est facile de vérifier que :
– Pour tout $t, t_1, ..., t_n \in \Lambda\mathcal{C}$, $(\mathcal{A})tt_1...t_n \succ_{\mathcal{C}} t$.
– $\vdash_{\mathcal{C}2} \mathcal{A} : \forall X\{\bot \to X\}$.

D'une manière générale j'ai démontré :

Théorème 7.1.6 *Si* $\vdash_{\mathcal{C}2} T : \forall X\{\bot \to X\}$, *alors pour tout* $n \in \mathbb{N}$, *et pour tous* $\lambda\mathcal{C}$*-termes* $t, t_1, ..., t_n$, $(T)tt_1...t_n \succ_{\mathcal{C}} t$.

La constante $\mathcal{C}$ satisfait les relations suivantes : $(\mathcal{C})tt_1...t_n \succ_{\mathcal{C}} (t)U$ et $(U)y \succ_{\mathcal{C}} (y)t_1...t_n$ où y est une nouvelle variable. En générale, j'ai obtenu la caractérisation suivante :

Théorème 7.1.7 *Si* $\vdash_{\mathcal{C}2} T : \forall X\{\neg\neg X \to X\}$, *alors il existe un entier* m, *tel que, pour tout entier* n, *et pour tous* $\lambda\mathcal{C}$*-termes* $t, t_1, ..., t_n$ *:*
$(T)tt_1...t_n \succ_{\mathcal{C}} (t)V_1$, $(V_i)y_i \succ_{\mathcal{C}} (t)V_{i+1}$ $(1 \leq i \leq m-1)$, *et* $(V_m)y_m \succ_{\mathcal{C}} (y_i)t_1...t_n$ *où* $y_1, ..., y_m$ *sont des nouvelles variables.*

7.2 Le $\lambda\mu$-calcul

Je vais commencer par présenter le système de typage classique de M. Parigot.

Le $\lambda\mu$-calcul possède deux alphabets distincts de variables : un ensemble de λ-variables x, y, z,..., et un ensemble de μ-variables α, β, γ,.... Les $\lambda\mu$- termes sont définis par la grammaire suivante : $t := x \mid \lambda xt \mid (t)t \mid \mu\alpha[\beta]t$.

La réduction en $\lambda\mu$-calcul est induite par deux notions différentes de réductions :

Les règles de calcul :
(C_1) $(\lambda xu)v \to u[v/x]$
(C_2) $(\mu\alpha u)v \to \mu\alpha u[v/{}^*\alpha]$
où $u[v/{}^*\alpha]$ est obtenu à partir du u en remplaçant inductivement chaque sous-terme de la forme $[\alpha]w$ par $[\alpha](w)v$.

Les règles de simplification :
(S_1) $[\alpha]\mu\beta u \to u[\alpha/\beta]$
(S_2) $\mu\alpha[\alpha]u \to u$, si α n'a pas d'occurrences libres dans u

(S_3) $\mu\alpha u \to \lambda x \mu\alpha u[x/{}^*\alpha]$, si u contient un sous terme de la forme $[\alpha]\lambda y w$.

M. Parigot a démontré dans [53] que :

Théorème 7.2.1 *En $\lambda\mu$-calcul, la réduction est confluente* [2].

Si u est obtenu à partir de t par une réduction de tête, on note $t \succ_\mu u$. L'équivalence de tête est notée : $u \sim_\mu v$ ssi il existe w tel que $u \succ_\mu w$ et $v \succ_\mu w$.

Les preuves sont écrites dans un système de déduction naturelle avec plusieurs conclusions présentées avec des séquents :
- Les formules à gauche de $\vdash$ sont étiquetées par des λ-variables.
- Les formules à droite de $\vdash$ sont étiquetées par des μ-variables, excepté d'une formule qui est étiquetée par un $\lambda\mu$-terme.
- Des formules distinctes ne possèdent pas la même étiquette.
Soient t un $\lambda\mu$-terme, A un type, $\Gamma = x_1 : A_1, ..., x_n : A_n$, et $\triangle = \alpha_1 : B_1, ..., \alpha_m : B_m$ deux contextes. On définit (par les règles suivantes) la notion "t est de type A dans Γ et $\triangle$", et on écrit $\Gamma \vdash_\mu t : A, \triangle$.
– Les règles du système $\mathcal{AF}2$.
– Si $\Gamma \vdash_\mu t : A, \beta : B, \triangle$, alors : $\Gamma \vdash_\mu \mu\beta[\alpha]t : B, \alpha : A, \triangle$ si $\alpha \neq \beta$ et $\Gamma \vdash_\mu \mu\alpha[\alpha]t : B, \triangle$ si $\alpha = \beta$.

Ce système possède les propriétés suivantes (voir [52] et [54]) :

Théorème 7.2.2 *1) Le type est préservé durant une réduction.*
2) Les $\lambda\mu$-termes typables sont fortement normalisables.

Par contre, on perd dans ce système la propriété de l'unicité de la représentation des données.

Soit $n \in \mathbb{N}$. Un entier classique est un $\lambda\mu$-terme clos θ_n, tel que $\vdash_\mu \theta_n : N[s^n(0)]$ pour un certain entier n. On dit aussi que l'entier classique θ_n est de valeur n.

On va caratériser maintenant les entiers classiques.

Soient x et f deux variables fixes, et $N_{x,f}$ l'ensemble des $\lambda\mu$-termes définis par le grammaire suivant : $u := x \mid \mu\alpha[\beta]x \mid (f)u \mid \mu\alpha[\beta](f)u$.

On définit, pour chaque $u \in N_{x,f}$, l'ensemble $rep(u)$, qui désigne intuitivement l'ensemble des entiers représentés par u :
- $rep(x) = \{0\}$;
- $rep((f)u) = \{n+1 \text{ si } n \in rep(u)\}$;
- $rep(\mu\alpha[\beta]u) = \bigcap rep(v)$ pour chaque sous terme $[\alpha]v$ de $[\beta]u$.

Le théorème suivant caractérise les entiers classiques (voir [54]).

[2] W. Py a donné dans [56] une preuve complète de ce résultat.

Théorème 7.2.3 *Les entiers classiques normaux de valeur n sont exactement les $\lambda\mu$-termes de la forme $\lambda x\lambda f u$ où $u \in N_{x,f}$ sans μ-variables libres et tel que $rep(u) = \{n\}$.*

Soit $\theta = \lambda x\lambda f u$ où
$u = (f)\mu\alpha[\alpha](f)\mu\phi[\alpha](f)\mu\psi[\alpha](f)(f)\mu\beta[\phi](f)\mu\delta[\beta](f)\mu\gamma[\alpha](f)\mu\rho[\beta](f)x$.
On vérifie que $rep(u) = \{4\}$, donc θ est un entier classique de valeur 4. Si on sait à l'avance que θ est un entier classique, alors on n'a pas besoin de chercher $rep(u)$ pour tous les sous-termes de u mais uniquement ceux des sous-termes de θ qui représentent un seul entier.

Je vais présenter maintenant une méthode très simple pour trouver la valeur d'un entier classique.

On définit, pour chaque $u \in N_{x,f}$, l'ensemble $val(u)$, qui désigne intuitivement l'ensemble des valeurs possibles de u :
- $val(x) = \{0\}$;
- $val((f)u) = \{n+1 \text{ si } n \in val(u)\}$;
- $val(\mu\alpha[\beta]u) = \bigcup val(v)$ pour chaque sous terme $[\alpha]v$ de $[\beta]u$.

Soient $u \in N_{x,f}$ sans μ-variables libres et $\alpha_1, ..., \alpha_n$ les μ-variables de u qui vérifient : α_1 est la μ-variable, telle que $[\alpha_1](f)^{i_1}x$ est un sous-terme de u, α_j $2 \leq j \leq n$ est la μ-variable, telle que $[\alpha_j](f)^{i_j}\mu\alpha_{j-1}u_{j-1}$ est un sous-terme de u et $u = (f)^{i_{n+1}}\mu\alpha_n u_n$. Soient $t_0 = x$ et $t_j = \mu\alpha_j u_j$ $1 \leq j \leq n$.

Le lemme suivant fournit un algorithme pour calculer la fonction val (voir [40]).

Lemme 7.2.1 *Pour tout $1 \leq j \leq n+1$ on a :*
1) $val(t_{j-1}) = \{\sum_{1 \leq k \leq j} i_k\}$.
2) Pour chaque sous terme t de u_j, tel que $t \neq (f)^r t_k$ $0 \leq k \leq j-1$, on a $val(t) = \emptyset$.
En particulier $val(u) = \{\sum_{1 \leq k \leq n+1} i_k\}$.

En utilisant le lemme 7.2.1 et le fait que pour tout $u \in N_{x,f}$, $rep(u) \subseteq val(u)$, on déduit le résultat suivant :

Théorème 7.2.4 *Si θ est un entier classique normal de valeur n, alors $\theta = \lambda x\lambda f u$ où $u \in N_{x,f}$ sans μ-variables libres et $val(u) = \{n\}$.*

Reprenons le dernier exemple

$$\theta = \lambda x\lambda f \overbrace{(f)}^{4} \underbrace{\mu\alpha[\alpha](f)\mu\phi[\alpha](f)\mu\psi[\alpha]}_{partie\quad fausse} \overbrace{(f)(f)}^{3} \underbrace{\mu\beta[\phi](f)\mu\delta[\beta](f)\mu\gamma[\alpha](f)\mu\rho[\beta]}_{partie\quad fausse} \overbrace{(f)x}^{1}.$$

- $\alpha_1 = \beta$, et $\alpha_2 = \alpha$;

- $i_1 = 1$, $i_2 = 2$, et $i_3 = 1$.
Donc la valeur de θ est égale à 4.

M. Parigot a démontré dans [53] que des opérateurs de mise en mémoire typés dans le système $\mathcal{AF}2$ peuvent être utilisés pour trouver les valeurs des entiers classqiues.

Théorème 7.2.5 *Si* $\vdash_{\mathcal{AF}2} T : \forall x\{N^g[x] \to \neg\neg N[x]\}$, *alors pour tout* $n \geq 0$, *il existe un* λ*-terme* $\tau_n \simeq_\beta \underline{n}$, *tel que pour tout entier classique normal* θ *de valeur* n, *il existe une substitution* σ, *telle que* $((T)\theta)f \sim_\mu \mu\alpha[\alpha](f)\sigma(\tau_n)$.

J'ai démontré le résultat suivant (voir [40]) :

Théorème 7.2.6 *Si* $u \in N_{x,f}$ *tel que* $val(u) = \{n\}$, *alors* $((T_i)\lambda x\lambda f u)f \succ (f)(\underline{s})^n\underline{0}$ *(i=1 ou 2).*

Le théorème 7.2.5 ne reste pas vrai pour un opérateur de mise en mémoire quelconque. Par exemple :
- Soient $u = \mu\alpha[\alpha](f)\mu\beta[\alpha]x$ et $\theta = \lambda x\lambda f u$. On a $val(u) = \{0\}$ et $rep(u) = \emptyset$.
- Soit $T = \lambda n((n)\delta)\lambda d\lambda g((T_N^i)n)\lambda x(g)(\underline{s})(\underline{p})x$ où
$\underline{p} = \lambda n(((n)\underline{0})\lambda a\lambda c((c)(\underline{s})\lambda c((c)\underline{0})(a)\lambda x\lambda yy)(a)\lambda x\lambda yx)\lambda x\lambda yy)\lambda x\lambda yy$ est un λ-terme pour le "prédécesseur".
On vérifie que pour tout $\theta_n \simeq_\beta \underline{n}$, $((T)\theta_n)f \succ \begin{cases} (f)\underline{0} & \text{si } n = 0 \\ (f)(\underline{s})(\underline{p})(\underline{s})^n\underline{0} & \text{si } n \neq 0 \end{cases}$.
Donc T est un opérateur de mise en mémoire [3] pour les entiers.
- $\vdash_{\mathcal{AF}2} T : \forall x\{N^g[x] \to \neg\neg N[x]\}$ (le typage nécessite l'introduction d'un système d'équations adéquat pour le type des entiers).
- Mais $((T)\theta)f \succ_\mu (f)(\underline{s})(\underline{p})\underline{0}$ et $(\underline{s})(\underline{p})\underline{0} \simeq_\beta \underline{1}$.

7.3 Le $\lambda\mu^{++}$-calcul

Le $\lambda\mu$-calcul code des preuves faites en logique classique et possède "presque" toutes les bonnes propriétés : la confluence, la conseravtion de type et la normalisation forte. Par contre, comme on a vu dans le paragraphe précédent, on perd dans ce calcul l'unicité de la représentation des données. De plus la présentation du système typé n'est pas très naturelle et ceci pour plusieurs raisons : l'autorisation d'avoir plusieurs formules à droite et forcer la succession de μ et [] dans la formation des termes. On va présenter une extention de ce calcul appelé $\lambda\mu^{++}$-calcul qui code exactement la déduction naturelle classique du second ordre. Ce calcul a les propriétés suivantes : la conseravtion de type, la normalisation forte, l'unicité de la représentation des données et la confluence sur les types de données.

[3] Cet opérateur est donné par J.L. Krivine.

Les termes du $\lambda\mu^{++}$-calcul (appelés aussi $\lambda\mu^{++}$-termes) sont définis par la grammaire suivante : $T := x \mid \alpha \mid \lambda x T \mid \mu\alpha T \mid (T)T$.

La réduction en $\lambda\mu^{++}$-calcul est induite par deux notions différentes de réductions :

Les règles du calcul
(C_λ) $(\lambda x U)V \rightharpoonup_\lambda U[V/x]$
(C_μ) $(\mu\alpha U)V \rightharpoonup_\mu \mu\beta U[\lambda y(\beta)(y)V/\alpha]$

Les règles de simplification
(S_1) $(\alpha)U\ V \rightharpoonup_1 (\alpha)U$
(S_2) $\mu\alpha\mu\beta U \rightharpoonup_2 \mu\alpha U[\lambda x x/\beta]$
(S_3) $(\alpha)(\beta)U \rightharpoonup_3 (\beta)U$
(S_4) $(\beta)\mu\alpha U \rightharpoonup_4 U[\lambda y(\beta)y/\alpha]$
(S_5) $\mu\alpha U \rightharpoonup_5 \lambda z\mu\beta U[\lambda y(\beta)(y)z/\alpha]$ (*)
(S_6) $\mu\alpha U[(\alpha)V/y] \rightharpoonup_6 V$ (**)

(*) U contient un sous-terme de la forme $(\alpha)\lambda x V$
(**) y est libre dans U et α n'est pas libre dans V

La règle (S_6) signifie que si $\mu\alpha T$ possède un sous-terme $(\alpha)V$ où V ne contient pas de variables libres qui sont liées dans $\mu\alpha T$, alors on peut rendre V comme résultat. Il est clair que cette règle est très difficile à implémenter mais pour les exemples et les propriétés que je vais présenter, une condition de type : "pas de lieurs actifs entre $\mu\alpha$ et (α)" suffira.

On écrit $T \rightharpoonup T'$ si on peut passer de T à T' en utilisant un nombre fini de fois les règles précédentes.

Il est clair que le $\lambda\mu^{++}$-calcul est non confluent.

Soit $\mathcal{A}$ un ensemble de $\lambda\mu^{++}$-termes normaux. On note $T \rightarrow \mathcal{A}$ ssi :
– Pour tout $U \in \mathcal{A}$, $T \rightharpoonup U$.
– Si $T \rightharpoonup U$ et U est normal, alors $U \in \mathcal{A}$.

Soit $T_{\mathbb{N}} = (Y)F$ où $F = \lambda x\lambda y\mu\alpha(\alpha)(((y)\lambda d(x)(\underline{s})y)\lambda x x)\mu\beta(\alpha)(((y)\lambda d y)\lambda x x)\ \alpha$, Y est le point fixe de turing et $\underline{s}$ un λ-terme pour le successeur sur les entiers de Church. Soit $\underline{N}$ l'ensemble des entiers de Church. Il est facile de vérifier que $(T_{\mathbb{N}})\ \underline{0} \rightarrow \underline{N}$.

Soient t un $\lambda\mu^{++}$-terme, A un type, et $\Gamma = x_1 : A_1, ..., x_n : A_n, \alpha_1 : \neg B_1, ..., \alpha_m : \neg B_m$ un contexte. On définit (par les règles suivantes) la notion "t est de type A dans Γ", et on écrit $\Gamma \vdash_+ t : A$.
– Les règles du système $\mathcal{AF}2$.
– Si $\Gamma, \alpha : \neg B \vdash_+ t :\bot$, alors : $\Gamma \vdash_+ \mu\alpha t : B$

Cette dernière règle axiomatise la logique classique au dessus de la logique intuitionniste.

Dès lors, on peut apporter des explications supplémentaires pour la règle (S_6). Elle signifie que dans une preuve d'une formule on ne peut pas avoir une sous-preuve de la formule elle même. Comme $\mu\alpha U[(\alpha)V/y]$ et V possède le même type, alors la règle autorise un programme de se réduire à un de ses sous-programmes qui fait la même chose.

Le $\lambda\mu^{++}$-calcul typé possède les propriétés suivantes :

Théorème 7.3.1 *1) Si* $\Gamma \vdash_+ U : A$ *et* $U \rightharpoonup V$, *alors* $\Gamma \vdash_+ V : A$.
2) Si $\Gamma \vdash_+ U : A$, *alors* U *est fortement normalisable.*

De plus, dans ce système, on récupère l'unicité de la représentation des données.

Théorème 7.3.2 *Soit* $n \in \mathbb{N}$. *Si* $\vdash_+ T : N[s^n(0)]$, *alors* $T \to \{\underline{n}\}$.

J'ai trouvé aussi un $\lambda\mu^{++}$-terme qui simule le ou-parallèle. Ceci n'est pas possible en $\lambda\mu$-calcul.

Un $\lambda\mu^{++}$-terme clos T est dit un faux booléen ssi :
$T \not\rightharpoonup \lambda xU$ ou
$T \rightharpoonup \lambda xU$ et $(U \not\rightharpoonup \lambda yV$ et $U \not\rightharpoonup (x)VV_1...V_n)$ ou
$T \rightharpoonup \lambda x\lambda yU$ et $(U \not\rightharpoonup \lambda yV$ et $U \not\rightharpoonup (x)WW_1...W_n$ et $U \not\rightharpoonup (y)WW_1...W_n)$.
Intuitivement un faux booléen est donc un terme qui peut donner les premiers éléments d'un booléen avant de boucler.

Soient FB l'ensemble des faux booléens, $B = \{T\,;\, T \to \{\mathbf{0}\}$ ou $T \to \{\mathbf{1}\}\}$, et $\mathcal{B} = B \bigcup FB$.

On dit qu'un $\lambda\mu^{++}$-terme normal clos T simule le ou-parallèle ssi pour tout $U, V \in \mathcal{B}$:
– $(T)UV \to \{\mathbf{0}, \mathbf{1}\}$;
– $(T)UV \rightharpoonup \mathbf{1}$ ssi $U \to \mathbf{1}$ et $V \to \mathbf{1}$;
– $(T)UV \rightharpoonup \mathbf{0}$ ssi $U \to \mathbf{0}$ ou $V \to \mathbf{0}$.

Soit $\widehat{\mathbf{1}} = \lambda p\mathbf{1}$ et $\widehat{\mathbf{0}} = \lambda p\mathbf{0}$.

Soit $\bigvee = \lambda x\lambda y\mu\alpha(\alpha)(((x)\widehat{\mathbf{1}})((y)\widehat{\mathbf{1}})\widehat{\mathbf{0}}))\ \mu\beta(\alpha)(((y)\widehat{\mathbf{1}})((x)\ \widehat{\mathbf{1}})\widehat{\mathbf{0}}))\alpha$.

Soit ou un symbôle de fonction binaire et considérons le système d'équations suivant : $ou(1, x) = 1$, $ou(0, x) = x$, $ou(x, 1) = 1$, et $ou(x, 0) = x$.

Théorème 7.3.3 $\bigvee$ *simule le ou-parallèle et* $\vdash_+ \bigvee : \forall x \forall y\{B[x], B[y] \to B[ou(x, y)]\}$.

Chapitre 8

Logiques mixtes

8.1 Un système de typage basé sur la logique mixte

Le théorème 7.1.4 ne peut être généralisé au système $\mathcal{C}2$. En effet, si $T = \lambda\nu\lambda f(f)(\mathcal{C})(T_i)\nu$ ($i = 1$ ou 2), alors il est facile de vérifier que $\vdash_{\mathcal{C}2} T : \forall x\{N^g[x] \to \neg\neg N[x]\}$ et qu'il n'existe pas de $\lambda\mathcal{C}$-terme $\tau_n \simeq_\beta \underline{n}$ tel que pour tout entier classique θ_n de valeur n, il existe une substitution σ, telle que $(T)\theta_n f \succ_{\mathcal{C}} (f)\sigma(\tau_n)$.
Ceci suggère plusieurs questions :
- Quelle est la relation entre les entiers classiques de valeur n et les λ-termes de type $N^g[s^n(0)]$?
- Pourquoi a-t-on besoin de la logique intuitionniste pour modéliser les opérateurs de mise en mémoire et de la logique classique pour modéliser les opérateurs de contrôle ?

Pour répondre à ces questions nous avons introduit (J.-L. Krivine et moi même) un système de typage basé sur une logique appelée "mixte". Cette logique permet de distinguer entre les preuves intuitionnistes et les preuves classiques et à la fois de typer les opérateurs de mise en mémoire et les opérateurs de contrôle.

Pour chaque entier n, on suppose qu'on a un ensemble dénombrable de variables spéciales d'arité n : X_C, Y_C, Z_C....appelées variables classiques. Un type A est dit classique ssi il se termine par $\perp$ ou par une variable classique. Nous ajoutons au système $\mathcal{AF}2$ les règles suivantes :

$$(0') \quad \Gamma \vdash \mathcal{C} : \forall X_C\{\neg\neg X_C \to X_C\}$$

$$(6'') \quad \frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X_C A}\ (*) \qquad\qquad (7'') \quad \frac{\Gamma \vdash t : \forall X_C A}{\Gamma \vdash t : A[G/X_C]}\ (**)$$

(*) X_C n'est pas libre dans Γ et (**) G est un type classique.
On appelle $\mathcal{M}2$ le nouveau système de typage.

On associe à chaque variable classique X_C, une variable spéciale $X^\bullet$ du système $\mathcal{AF}2$ ayant la même arité que X_C. Pour chaque formule A de $\mathcal{M}2$, on définit la formule A° de $\mathcal{AF}2$ de la manière suivante :
- Si $A = \bot$, alors $A^\circ = A$.
- Si $A = X(t_1, ..., t_n)$, alors $A^\circ = A$.
- Si $A = X_C(t_1, ..., t_n)$, alors $A^\circ = \neg X^\bullet(t_1, ..., t_n)$.
- Si $A = B \to C$, alors $A^\circ = B^\circ \to C^\circ$.
- Si $A = \forall x B$ (resp. $A = \forall X B$), alors $A^\circ = \forall x B^\circ$ (resp. $A^\circ = \forall X B^\circ$).
- Si $A = \forall X_C B$, alors $A^\circ = \forall X^\bullet B^\circ$.

Le résultat suivant signifie qu'on ne peut pas faire une preuve normale non intuitionniste d'une formule $\forall^+$ ne contenant pas de variables classiques (voir [45]).

Théorème 8.1.1 *Soient A un type $\forall^+$ de $\mathcal{AF}2$ et t un $\lambda\mathcal{C}$-terme β-normal. Si $\vdash_{\mathcal{M}2} t : A$, alors t est λ-terme normal, et $\vdash_{\mathcal{AF}2} t : A$.*

À chaque variable X du système $\mathcal{C}2$, on associe une variable classique X_C ayant la même arité que X. Pour chaque formule A de $\mathcal{C}2$, on définit une formule A^C de $\mathcal{M}2$ de la manière suivante :
- Si $A = \bot$, alors $A^C = A$.
- Si $A = X(t_1, ..., t_n)$ où X est une variable de relation, alors $A^C = X_C(t_1, ..., t_n)$.
- Si $A = B \to C$, alors $A^C = B^C \to C^C$.
- Si $A = \forall x B$, alors $A^C = \forall x B^C$.
- Si $A = \forall X B$, alors $A^C = \forall X_C B^C$.

Par exemple $N^C[x] = \forall X_C\{X_C(0), \forall y(X_C(y) \to X_C(s(y))) \to X_C(x)\}$.

Nous avons le résultat suivant (voir [45]).

Théorème 8.1.2 *Soient A un type de $\mathcal{C}2$, et t un $\lambda\mathcal{C}$-terme. $\vdash_{\mathcal{C}2} t : A$ ssi $\vdash_{\mathcal{M}2} t : A^C$.*

Les résultats précédents nous permettent de caractériser les entiers du système $\mathcal{M}2$ (voir [45]).

Théorème 8.1.3 *Soit $n \in \mathbb{N}$. Si $\vdash_{\mathcal{M}2} t : N[s^n(0)]$, alors $t \simeq_\beta \underline{n}$.*

Théorème 8.1.4 *Soit $n \in \mathbb{N}$. Si $\vdash_{\mathcal{M}2} t : N^C[s^n(0)]$, alors t est un entier classique de valeur n.*

Soit T un $\lambda\mathcal{C}$-terme clos. On dit que T est un opérateur de mise en mémoire pour les entiers classiques ssi pour tout $n \in \mathbb{N}$, il existe un $\lambda\mathcal{C}$-terme $\tau_n \simeq_\beta \underline{n}$, tel que pour tout entier classique θ_n de valeur n, il existe une substitution σ, telle que $(T)\theta_n f \succ_C (f)\sigma(\tau_n)$ (où f est une nouvelle variable).

La généralisation du théorème 7.1.4 que nous avons trouvée est la suivante (voir [45]).

Théorème 8.1.5 *Si* $\vdash_{\mathcal{M}2} T : \forall x\{N^C[x] \to \neg\neg N[x]\}$, *alors* T *est un opérateur de mise en mémoire pour les entiers classiques.*

Le théorème précédent signifie que si $\vdash_{\mathcal{M}2} T : \forall x\{N^C[x] \to \neg\neg N[x]\}$, alors T prend un entier classique en argument et donne comme résultat l'entier de Church correspondant.

Nous avons démontré qu'il suffit de prouver le théorème 8.1.5 dans la restriction propositionnelle du système $\mathcal{M}2$ notée $\mathcal{M}$ (voir [45]).

Théorème 8.1.6 *Si* $\vdash_{\mathcal{M}} T : N^C \to \neg\neg N$, *alors pour tout* $n \in \mathbb{N}$, *il existe* $m \in \mathbb{N}$ *et un* $\lambda\mathcal{C}$*-terme* $\tau_m \simeq_\beta \underline{m}$, *tels que pour tout entier classique* θ_n *de valeur* n, *il existe une substitution* σ, *telle que* $(T)\theta_n f \succ_C (f)\sigma(\tau_m)$.

La preuve de ce théorème utilise deux résultats indépendants :
– Le théorème 7.1.3 qui décrit le comportement d'un entier classique.
– Le théorème 8.1.7 qui décrit le comportemant des $\lambda\mathcal{C}$-termes de type $N^C \to \neg\neg N$.

Soient ν et f deux variables distinctes. On note par $x_{n,a,b,\overline{c}}$ (où n est un entier, a, b deux $\lambda\mathcal{C}$-termes, et $\overline{c}$ une suite finis de $\lambda\mathcal{C}$-termes) une variable qui ne figurent pas dans $a, b, \overline{c}$.

Théorème 8.1.7 *Soit* $\vdash_{\mathcal{M}} T : N^C \to \neg\neg N$ *et* $n \in \mathbb{N}$. *Il existe* $m \in \mathbb{N}$ *et une suite finie de* $\mathcal{C}$*-réductions de tête* $\{U_i \succ_C V_i\}_{1 \leq i \leq r}$ *telles que :*
1) $U_1 = (T)\nu f$ *et* $V_r = (f)\tau_m$ *où* $\tau_m \simeq_\beta \underline{m}$ *;*
2) $V_i = (\nu)ab\overline{c}$ *où* $V_i = (x_{l,a,b,\overline{c}})\overline{d}$ $(0 \leq l \leq n-1)$ *;*
3) Si $V_i = (\nu)ab\overline{c}$, *alors* $U_{i+1} = (a)\overline{c}$ *si* $n = 0$ *et* $U_{i+1} = ((b)x_{n-1,a,b,\overline{c}})\overline{c}$ *si* $n \neq 0$
4) Si $V_i = (x_{l,a,b,\overline{c}})\overline{d}$ $(0 \leq l \leq n-1)$, *alors* $U_{i+1} = (a)\overline{d}$ *si* $l = 0$ *et* $U_{i+1} = ((b)x_{l-1,a,b,\overline{d}})\overline{d}$ *si* $l \neq 0$.

Le théorème 8.1.5 se généralise aussi pour tous les types $\forall^+$.

8.2 La logique mixte propositionnelle

Les calculs propositionnels minimal, intuitionniste, et classique (notés (LM), (LI) et (LC)) sont construits l'un au dessus de l'autre en ajoutant des schémas d'axiomes ou des nouvelles règles. La formalisation la plus connue consiste à ajouter la règle de l'absurdité intuitionniste (de l'absurde on déduit toutes les formules) à (LM) pour obtenir (LI), et ajouter la règle de l'absurdité classique (une formule non fausse est vraie) à (LM) ou (LI) pour obtenir (LC). Avec ce type de formalisme on se heurte à de nombreux problèmes :

– Une formule propositionnelle démontrable en (LC) ne contient pas d'information sur le plus petit système dans lequel elle est dérivable. Pour avoir ces informations on pourrait utiliser les algorithmes de décision non efficaces de

(LM) et (LI). D'autant plus, avec ces algorithmes on ne saurait sur quelles formules les raisonnements par l'absurde ont été utiliés.

– Une formule ne contient pas d'informations sur sa "meilleure" dérivation. Par exemple on peut prouver la formule $A = (X \to Y) \vee (Y \to X)$ en faisant un raisonnement par l'absurde classique sur A. Un raisonnement par l'absurde classique sur la variable Y suffit aussi pour la prouver. En effet, si Y est vraie, on aura, dans (LM), $X \to Y$, et si Y est fausse, on aura, dans (LI), $Y \to X$. La deuxième dérivation se raproche plus du raisonnement humain, pour cela on l'appelera "une bonne dérivation" pour A.

– Chacune de ces trois logiques a une sémantique. Pour (LC) c'est les tables de vérité et pour (LI) (resp. (LM)) c'est les modèles de Kripke intuitionnistes (resp. minimaux). Si on observe de près les preuves des théorèmes de complétude on constate une grande ressemblance. Alors, pourquoi, ne pas étudier ces logiques en même temps ? C.à.d. introduire une unique sémantique pour ces logiques et démontrer un seul théorème de complétude.

Nous avons présenté (A. Nour et moi même) un système de déduction naturelle dans lequel des réponses patielles aux questions posées auparavant ont été données.

On suppose qu'on a trois ensembles dénombrables disjoints de variables propositionnelles : $V_m = \{X_m, Y_m, Z_m, ...\}$ l'ensemble de variables minimales ; $V_i = \{X_i, Y_i, Z_i, ...\}$ l'ensemble de variables intuitionnistes ; $V_c = \{X_c, Y_c, Z_c, ...\}$ l'ensemble de variables classiques ; et une constante spéciale $\bot$. Les formules sont construites sur l'ensemble $\mathcal{P} = V_m \bigcup V_i \bigcup V_c \bigcup \{\bot\}$ en utilisant les connecteurs $\wedge$, $\vee$, et $\to$. On note $\neg A$ la formule $A \to \bot$. Pour toute formule A, on note $var(A)$ l'ensemble de ses variables. Une formule classique (resp. intuitionnoste, minimale) est une formule A telle que $var(A) \subseteq V_c$ (resp. $var(A) \subseteq V_i$, $var(A) \subseteq V_m$).

On définit sur ces formules un système de déduction naturelle appelé "logique mixte propositionnelle" et noté (PML) dont les règles sont celles habituelles de la logique minimale plus deux autres règles pour l'$\bot$:

$$(\bot_i)\ \frac{\Gamma \vdash \bot}{\Gamma \vdash A} \qquad\qquad (\bot_c)\ \frac{\Gamma \vdash \neg\neg B}{\Gamma \vdash B}$$

où A est une formule intuitionniste et B est une formule classique.

Les formules suivantes $(X_m \to X_c) \vee (X_c \to X_i)$ et $(X_c \to X_m \vee X_i) \to (X_m \vee (X_c \to X_i))$ sont démontrables.

Je vais définir une sémantique de type Kripke pour (PML).

Un modèle de Kripke généralisé est un triplet $\mathcal{K} = (K, \leq, \Vdash)$, où $(K, \leq)$ est un ensemble non vide partiellement ordonné, et $\Vdash$ est une relation binaire sur $K \times \mathcal{P}$ telle que :
1) Si $\alpha \Vdash X$ et $\beta \geq \alpha$, alors $\beta \Vdash X$.
2) Si $\alpha \Vdash \bot$, alors pour toute variable classique X_c (resp. pour toute variable intuitionniste X_i) $\alpha \Vdash X_c$ (resp. $\alpha \Vdash X_i$).
3) Si $\alpha \Vdash X_c$ et, $\alpha \not\Vdash \bot$, alors, pour tout $\beta \in K$, $\beta \Vdash X_c$.

$\Vdash$ s'étend sur les formules par les clauses suivantes :
– $\alpha \Vdash A \wedge B$ ssi $\alpha \Vdash A$ et $\alpha \Vdash B$.
– $\alpha \Vdash A \vee B$ ssi $\alpha \Vdash A$ ou $\alpha \Vdash B$.
– $\alpha \Vdash A \rightarrow B$ ssi pour tout $\beta \geq \alpha$, si $\beta \Vdash A$, alors $\beta \Vdash B$.

Une formule A est dite valide dans $\mathcal{K} = (K, \leq, \Vdash)$ ssi pour tout $\alpha \in K$, $\alpha \Vdash A$; notation $\mathcal{K} \Vdash A$. Si Γ est un ensemble de formules, on dit que $\Gamma \Vdash A$ ssi pour tout $\mathcal{K}$: si pour tout $B \in \Gamma$, $\mathcal{K} \Vdash B$, alors $\mathcal{K} \Vdash A$.

Nous avons obtenu le théorème de complétude suivant (voir [46]) :

Théorème 8.2.1 $\Gamma \vdash_{pml} A$ *ssi* $\Gamma \Vdash A$.

Le système (PML) possède aussi la propriété des modèles finis et donc il est décidable (voir [46]).

On écrit $\Gamma \vdash_{(c)} A$ si $\Gamma \vdash A$ est dérivable sans utiliser la règle $(\bot_c)$ et on écrit $\Gamma \vdash_{(c,i)} A$ si $\Gamma \vdash A$ est dérivable sans utiliser les règles $(\bot_i)$ et $(\bot_c)$. Le résultat suivant (théorème 8.2.2) montre que pour dériver une formule en logique mixte dans le sous-calcul qui utilise les règles de la logique classique, intuitionniste ou minimale, il est nécessaire que la formule contienne respectivement une variable classique, intuitionniste ou minimale. Ce résultat se démontre facilement si le système (PML) possède la propriété de la sous-formule. Habituellement cette dernière propriété se déduit du théorème d'élimination des coupures qui est difficile à démontrer ici car les formules disjonctives ne peuvent être coder (i.e. $\neg(\neg A \wedge \neg B) \rightarrow A \vee B$ n'est pas dérivable). Pour ces raisons nous avons fait une preuve sémantique (voir [46]) :

Théorème 8.2.2 *1) Soit* $\Gamma \bigcup \{A\}$ *un ensemble de formules sans variables classiques.* $\Gamma \vdash_{pml} A$ *ssi* $\Gamma \vdash_{(c)} A$.
2) Soit $\Gamma \bigcup \{A\}$ *un ensemble de formules sans variables classiques et intuitionnistes.* $\Gamma \vdash_{pml} A$ *ssi* $\Gamma \vdash_{(c,i)} A$.

Nous avons, par contre, réussi à faire une preuve syntaxique du théorème précédent dans un sous système de (PML) noté $(PML^{\vee})$. Ce système est obtenu avec la restriction suivante sur la règle d'élimination de $\vee$:

$$(\vee_e)\ \frac{\Gamma_1 \vdash A \vee B \quad \Gamma_2, A \vdash C \quad \Gamma_3, B \vdash C}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash C}$$

si $A \vee B$ est une formule classique, alors C est aussi une formule classqiue.

On va établir maintenant des relations entres les systèmes (LM), (LI), (LC) et (PML).

Soit $V = \{X, Y, Z, ...\}$ un ensemble dénombrable de variables propositionnelles. On suppose que les ensembles V_m, V_i, et V_c sont obtenus en indexant les variables de V. En utilisant $V \bigcup \{\bot\}$, on définit les systèmes (LM), (LI), et (LC). Une formule sur $V \bigcup \{\bot\}$ est appelée formule ordinaire. On définit sur $V_m \bigcup V_i \bigcup V_c$ une relation binaire $<$ en posant : pour toute $X \in V$, $X_m < X_i < X_c$.

Une décoration est une fonction $d : V \to \mathcal{P}$ telle que $d(X) \in \{X_m, X_i, X_c\}$. Elle s'étend naturellement sur les formules ordinaires. On définit sur les décorations une relation binaire notée aussi $<$ en posant : $d < d'$ ssi pour toute variable $X \in V$, $d(X) \leq d'(X)$ et il existe $X \in V$ telle que $d(X) < d'(X)$.

Soit A une formule ordinaire telle que $\vdash_c A$. Une décoration pour A est une décoration d telle que $\vdash_{pml} d(A)$ et pour toute variable X qui ne parait pas dans A, $d(X) = X_m$. Une décoration minimale pour A est une décoration d pour A telle que si $d' \leq d$ est une décoration pour A, alors $d' = d$.

Nous avons obtenu le résultat suivant (voir [46]) :

Théorème 8.2.3 *Soit A une formule ordinaire telle que $\vdash_c A$. A possède une décoration minimale.*

Une décoration minimale pour une formule A n'est pas unique. En effet soient d et d' les deux décorations suivantes : $d(X) = X_c$, $d(Y) = Y_i$, $d'(X) = X_i$ et $d'(Y) = Y_c$. Il est facile de vérifier que d et d' sont des décorations minimales non comparables pour la formule $(X \to Y) \vee (Y \to X)$.

Pour décorer les dérivations, on modifie légèrement le système (PML) en remplacant la règle $(\bot_i)$ par :

$$(\bot_i)\ \frac{\Gamma \vdash \bot}{\Gamma \vdash A}$$

si $var(A) \bigcap V_m = \emptyset$.
Il est clair que le système obtenu est équivalent à (PML).

Une décoration pour une dérivation $\mathcal{D}$ de (LC) est une décoration d telle que pour toute variavle X qui ne parait pas dans $\mathcal{D}$, $d(X) = X_m$ et l'image de $\mathcal{D}$ par d est une dérivation de (PML). Une décoration minimale pour $\mathcal{D}$ est une décoration d pour $\mathcal{D}$ telle que : si $d' \leq d$ est décoration pour $\mathcal{D}$, alors $d' = d$.

Nous avons obtenu le résultat suivant (voir [46]) :

Théorème 8.2.4 *Soit $\mathcal{D}$ une dérivation de (LC). $\mathcal{D}$ possède une unique décoration minimale (notée $d_{\mathcal{D}}$).*

Soit A une formule ordinaire telle que $\vdash_c A$. Une bonne dérivation pour A est une dérivation $\mathcal{D}$ de A en (LC) telle que $d_{\mathcal{D}}$ est une décoration minimale pour A. Intuitivement, une bonne dérivation pour une formule est une dérivation dans laquelle le "minimum" possible du raisonnement par l'absurde est fait.

Nous avons démontré dans [46] le résultat suivant :

Théorème 8.2.5 *Chaque formule ordinaire démontrable en (LC) possède une bonne dèrivation.*

Nous avons proposé une version à la calcul des séquents de (PML). Nous nous interréssons à démontrer un théorème d'élimination des coupures pour ce système.

Chapitre 9

Résultats divers

9.1 Équivalence opérationnelle

Deux λ-termes clos M et N sont opérationnellement équivalent ($M \simeq_{oper} N$) ssi pour tout contexet C : $C[M]$ est résoluble ssi $C[N]$ l'est.

Soit $J = (Y)G$ où Y est l'opérateur de point fixe de Turing et $G = \lambda x\lambda y\lambda z(y)(x)z$. J est donc un η-expansé infini de **id**. Son arbre de Böhm est en fait $\lambda x\lambda x_1(x)\lambda x_2(x_1)\lambda x_3(x_2)\lambda x_4(x_3)....$

Le théorème suivant est bien connu (voir [2] et [18]).

Théorème 9.1.1 $J \simeq_{oper}$ **id**.

La preuve connue de ce résultat est purement sémantique : deux λ-termes sont opérationnellement équivalent ssi ils ont la même interprétation dans le modèle D_∞.

Nous avons donné (R. David et moi même) une preuve élémentaire et purement syntaxique de ce résultat. La preuve analyse finement les réductions de C[**id**] et $C[J]$ en distinguant les "vrais" β -redex de ceux qui proviennent des η-expansions. L'idée de la démonstration est la suivante : nous prouvons que, si nous simulons les réductions où **id** (resp. J) sont en position de tête, C[**id**] et $C[J]$ se réduisent, par réduction de tête de la même façon. Pour ceci nous ajoutons une constante H (qui représente à la fois **id** et J). Nous définissons sur ces termes la **id** (resp. J) réduction de tête, qui correspondent aux cases où $H =$ **id** (resp. J). Pour prouver que les réductions sont équivalents, nous montrons que les termes obtenus en éliminant la constante H sont égaux.

Cette démonstration peut être généralisée pour prouver l'équivalence operationnelle de deux λ-termes ayant le même arbre de Böhm à η-expansion infinie près. Pour cela on a besoin du λ-calcul dirigé (voir [8]).

9.2 Complétudes de la logique du second ordre

La preuve du théorème de complétude de la logique classique du second ordre se déduit de celui de la logique classique du premier ordre (pour une certaine théorie) moyennant un codage des formules. L'idée principale de ce codage est d'associer à chaque variable du second ordre X d'arité n, une variable du premier ordre x et de coder la formule $X(x_1, ..., x_n)$ par $Ap_n(x, x_1, ..., x_n)$ où Ap_n est un symbole de relation d'arité $n+1$. Pour montrer que ce codage transforme les dérivations faites en logique du second ordre en dérivations faites en logique du premier ordre, on ajoute habituellement une liste d'axiomes qui empêche de prouver des formules du premier ordre qui ne sont pas des images des formules du second ordre. Ceci rend les preuves lourdes et difficiles à formaliser proprement. C. Raffalli et moi même avons proposé une solution très simple à ce problème [1]. Il s'agit de construire un codage inverse (de la logique du premier ordre dans la logique du second ordre) telle que l'image successive par les deux cogades d'une formule du second ordre est la formule elle même. Cette méthode permet aussi de déduire le théorème de complétude de la logique intuitionniste du second ordre à partir de celui de la logique du premier ordre.

Soit $\mathcal{L}_2$ le langage de la logique du second ordre contenant :
(i) les symboles logiques : $\bot$,$\rightarrow$, $\wedge$, $\vee$, $\forall$ et $\exists$
(ii) un ensemble dénombrable $\mathcal{V}$ de variables d'individu : $x_0, x_1, x_2, ...$;
(iii) pour chaque $i \geq 0$, un ensemble dénombrable $\mathcal{V}_i$ de variables de relation d'arité i : $X_0^i, X_1^i, X_2^i, ...$

Soit $\mathcal{L}_1$ le langage de la logique du premier ordre contenant :
(i) les mêmes symboles logiques ;
(ii) l'ensemble dénombrable $\mathcal{V}$ de variables d'individu ;
(iii) pour chaque $i \geq 0$, un symbole de relation Ap_i d'arité $i+1$.

On note $A \leftrightarrow B$ la formule $(A \rightarrow B) \wedge (B \rightarrow A)$.

On considère, comme système de démonstration, la déduction naturelle et on écrit $\Gamma \vdash_n A$ où $n \in \{1, 2\}$ (pour logique du premier ou du second ordre).

On fixe, pour chaque $i \geq 0$, une bijection ϕ_i entre $\mathcal{V}_i$ et $\mathcal{V}$.

Soit A une formule de la logique du second ordre. On définit par induction sur A son code A^* dans la logique du premier ordre :
– $\bot^* = \bot$
– $(X^n(x_1, ..., x_n))^* = Ap_n(\phi_n(X^n), x_1, ..., x_n)$
– $(A \diamond B)^* = A^* \diamond B^*$; $\diamond \in \{\rightarrow, \wedge, \vee\}$
– $(QxA)^* = Qy(A[y/x])^*$ où $y \notin Fv(A^*)$; $Q \in \{\forall, \exists\}$
– $(QX^nA)^* = Q\phi_n(Y^n)(A[Y^n/X^n])^*$ où $\phi_n(Y^n) \notin Fv(A^*)$; $Q \in \{\forall, \exists\}$

[1] L'article est en cours de rédaction.

La fonction $*$ est évidement non surjective. Par exemple, on ne peut pas trouver une formule A telle que $A^* = \forall x Ap_1(x,x)$.

Le Schéma de compréhension du second ordre (SC_2) est l'ensemble de toutes les formules du second ordre suivantes : Pour chaque formule G telle que $Fv(G) = \{x_1,...,x_n,\zeta_1,...,\zeta_m\}$,
$\forall\zeta_1...\forall\zeta_m[\exists X^n\forall x_1...\forall x_n(G \leftrightarrow X^n(x_1,...,x_n))]$ où $X \notin Fv(G)$.

Il est facile de démontrer que (SC_2) est équivalent à la règle d'élimination du quantificateur universel du second ordre.

Le Schéma de compréhension du premier ordre (SC_1) défini par $(SC_1) = \{G^*\,;\ G \in (SC_2)\}$.

Le résultat suivant montre que la fonction $*$ permet aussi de transformer les preuves faites en logique du second ordre en des preuves faites en logique du premier ordre.

Théorème 9.2.1 *Soient $\Gamma \bigcup \{A\}$ un ensemble de formules du second ordre. Si $\Gamma \vdash_2 A$, alors $\Gamma^*, (SC_1) \vdash_1 A^*$.*

Soit A une formule du premier ordre et x une variable. On définit par induction sur A son code $\overline{A}$ dans la logique du second ordre et une suite ordonnée de variable $\mathcal{V}_A^x$.
– $\overline{\bot} = \bot$ et $\mathcal{V}_\bot^x = \emptyset$
– $\overline{Ap_n(y,y_1,...,y_n)} = \phi_n^{-1}(y)(y_1,...,y_n)$ et
$\mathcal{V}_A^x = \emptyset$ si $y \neq x$ et pour tout $i \in \{1,...,n\}$, $y_i \neq x$
$\mathcal{V}_A^x = < x >$ si $y \neq x$ et il existe $i \in \{1,...,n\}$, $y_i = x$
$\mathcal{V}_A^x = < \phi_n^{-1}(x) >$ si $y = x$ et pour tout $i \in \{1,...,n\}$, $y_i \neq x$
$\mathcal{V}_A^x = < \phi_n^{-1}(x), x >$ si $y = x$ et il existe $i \in \{1,...,n\}$, $y_i = x$
– $\overline{A \diamond B} = \overline{A} \diamond \overline{B}$ et $\mathcal{V}_{A\diamond B}^x = \mathcal{V}_A^x \bigcup (\mathcal{V}_B^x - \mathcal{V}_A^x)\,;\ \diamond \in \{\rightarrow, \wedge, \vee\}$
– $\overline{QyA} = Q\zeta_1...Q\zeta_n\overline{A}$ où $\mathcal{V}_A^y = \{\zeta_1,...,\zeta_n\}$ et $\mathcal{V}_{\forall yA}^x = \mathcal{V}_A^x$ $(y \neq x)\,;\ Q \in \{\forall, \exists\}$

On a le résultat important suivant :

Lemme 9.2.1 *Si A une formule du second ordre, alors $\vdash_2 \overline{A^*} \leftrightarrow A$.*

Le résultat suivant montre que la fonction — permet aussi de transformer les preuves faites en logique du premier ordre en des preuves faites en logique du second ordre.

Théorème 9.2.2 *Soient $\Gamma \bigcup \{A\}$ un ensemble de formules du premier ordre. Si $\Gamma \vdash_1 A$, alors $\overline{\Gamma} \vdash_2 \overline{A}$.*

Une interprétation de $\mathcal{L}_1$ est un couple $\mathcal{M} = (M, (Ap_n^M)_n)$ où M est un ensemble non vide et pour tout $n \geq 0$, $Ap_n^M \subseteq M^{n+1}$. Une $\mathcal{M}$-substitution σ est

une fonction de l'ensemble $\mathcal{V}$ dans M.

Une interprétation de $\mathcal{L}_2$ est un couple $\mathcal{M} = (M, (M_n)_n)$ où M est un ensemble non vide et $M_n \subseteq \mathcal{P}(M^n)$. Pour $n = 0$, on suppose que $M_0 = \mathcal{P}(M^0) = \{0, 1\}$. Une $\mathcal{M}$-substitution σ est une fonction sur les ensembles $\mathcal{V}$ et $(\mathcal{V}_i)_i$ telle que $\sigma(x) \in M$ et $\sigma(X^n) \in M_n$.

Soit $\mathcal{M}_1 = (M, (Ap_n^M)_n)$ une interprétation de $\mathcal{L}_1$. On définit une interprétation $\overline{\mathcal{M}_1} = (M, (M_n)_n)$ de $\mathcal{L}_2$ où $M_n = (M_n^a)_{a \in M}$, $M_0^a = \{1\}$ si $a \in Ap_0^M$, $M_0^a = \{0\}$ si $a \notin Ap_0^M$, et $M_n^a = \{(a_1, ..., a_n) \in M^n ; (a, a_1, ..., a_n) \in Ap_n^M\}$ $(n \geq 1)$.

Soit F une formule du second ordre telle que $Fv(F) = \{x_1, ..., x_n, X_1^{k_1}, ..., X_m^{k_m}\}$ et supposons que $\phi_{k_j}(X_j^{k_j}) = y_j$ $(1 \leq j \leq m)$. Si σ est une $\mathcal{M}_1$-substitution telle que $\sigma(x_i) = a_i$ $(1 \leq i \leq n)$ et $\sigma(y_j) = b_j$ $(1 \leq j \leq m)$, alors on note $\overline{\sigma}$ une $\overline{\mathcal{M}_1}$-substitution telle que $\overline{\sigma}(x_i) = a_i$ $(1 \leq i \leq n)$ et $\overline{\sigma}(X_j^{k_j}) = M_{k_j}^{b_j}$ $(1 \leq j \leq m)$,

Théorème 9.2.3 *$\overline{\mathcal{M}_1} \models F[\overline{\sigma}]$ ssi $\mathcal{M}_1 \models F^*[\sigma]$.*

Donc on obtient le résultat suivant :

Corollaire 9.2.1 *$\mathcal{M}_1 \models (SC_1)$ ssi $\overline{\mathcal{M}_1} \models (SC_2)$.*

D'où le théorème du complétude de la logique classique du second ordre.

Théorème 9.2.4 *Soit $\Gamma \bigcup \{A\}$ un ensemble de formules du second ordre. $\Gamma \vdash_2 A$ ssi pour toute interprétation du second ordre $\mathcal{M}$ telle que $\mathcal{M} \models (SC_2)$ et pour toute $\mathcal{M}$-substitution σ : $\mathcal{M} \models \Gamma[\sigma]$ implique $\mathcal{M} \models A[\sigma]$.*

Bibliographie

[1] M. Abali, L. Cardelli, P.L. Curien and J.L. Levy. *Explicit Substitutions*. technical report 1176, INRIA, 1990.

[2] H. Barendregt. *The lambda calculus : Its Syntax and Semantics*. North Holland, 1984.

[3] C. Böhm and A. Berarducci. *Automatic synthesis of typed Λ-programs on term algebra. Theoretical Computer Science* **39**, 135-154, 1985.

[4] A. Chagrov and M. Zakhryaschev. *Modal Logic* Clarendon Press, Oxford, 1997.

[5] P.L. Curien. *The λρ-calculi : an abstract framework for closures*. Technical report, LIENS - Ecole Normale Supérieure, 1988.

[6] V. Danos et L. Regnier. Notes sur la mise en mémoire. *Manuscript*, 1992.

[7] R. David. The Inf function in system F. *Theorical Computer Science* **135**, 423-431, 1994.

[8] R. David and K. Nour. Storage operators and directed lambda-calculus. *Journal of Symbolic Logic* **60-4**, 1054-1086, 1995.

[9] R. David and K. Nour. A syntactical proof of the operational equivalence of two λ-terms. *Theoretical Computer Science* **180**, 371-375, 1997.

[10] S. Farkh. Types de données en logique du second ordre. *Thèse de doctorat, Université de Savoie*, 1998.

[11] S. Farkh et K. Nour. Un résultat de complétude pour les types $\forall^+$ du système F. *CRAS. Paris* **326, Série I**, 275-279, 1998.

[12] S. Farkh et K. Nour. Résultats de complétude pour des classes de types du système AF2. *Informatique Théorique et Application* **31-6**, 513-537, 1998.

[13] S. Farkh et K. Nour. Les types de données systaxiques du système F. *Soumis à Informatique théorique et applications*, 1999.

[14] S. Farkh et K. Nour. Types complets dans une extension du système AF2. *Soumis à Informatique théorique et applications*, 2000.

[15] M. Felleisein. *The Calculi of $\lambda_v - CS$ conversion : a syntactic theory of control and state in imperative higher order programming*. Ph. D. dissertation, Indiana University, 1987.

[16] P. Giannini and S. Ronchi. Characterization of typing in polymorphic type discipline. *LICS, Edinboug*, 61-70, 1988.

[17] J.-Y. Girard, Y. Lafont, P. Taylor. Proofs and types. *Cambridge University Press*, 1986.

[18] J. Hyland. A syntactic characterization of the equality in some models of the lambda calculus. *J. London Math. Soc. (2)*, **12**, 361-370, 1976.

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

[20] J.-L. Krivine. Opérateurs de mise en mémoire et traduction de Gödel. *Archive for Mathematical Logic* **30**, 241-267, 1990.

[21] J.-L. Krivine. Lambda-calcul, évaluation paresseuse et mise en mémoire. *Theoretical Informatics and Applications* **25-1**, 67-84, 1991.

[22] J.-L. Krivine. Mise en mémoire (preuve générale). *Manuscript*, 1993.

[23] J.-L. Krivine. Classical logic, storage operators and 2nd order lambda-calculus. *Annals of Pure and Applied Logic* **68**, 53-78, 1994.

[24] J.-L. Krivine. A general storage theorem for integers in call-by-name λ-calculus. *Theoretical Computer Science* **129**, 79-94, 1994.

[25] R. Labib-Sami. Typer avec (ou sans) types auxilières *Manuscript*, 1986.

[26] D. Leivant. Reasonning about functional programs and complexity classes associated with type disciplines. *In 24th Annual Symposium on Foundations of Computer Science* **44**, 460-469, 1984.

[27] D. Leivant. Typing and computation properties of lambda expressions. *Theoretical Computer Science* **44**, 51-68, 1986.

[28] J.-C. Mitchell. Polymorphic type inference and containment. *Inf. Comput.* **76 2/3** Feb./Mar., 211-249, 1988.

[29] K. Nour. Opérateurs de mise en mémoire en lambda calcul pur et typé. *Thèse de doctorat, université de savoie*, 1993.

[30] K. Nour. Opérateurs propres de mise en mémoire. *CRAS. Paris* **317, Série I**, 1-6, 1993.

[31] K. Nour. Preuve syntaxique d'un théorème de J.L. Krivine sur les opérateurs de mise en mémoire. *CRAS. Paris* **318, Série I**, 201-204, 1994.

[32] K. Nour. Quelques résultats sur le λC-calcul. *CRAS. Paris***320, Série I**, 259-262, 1995.

[33] K. Nour. Strong storage operators and data types. *Archive for Mathematical Logic* **34**, 65-78, 1995.

[34] K. Nour. Entiers intuitionnistes et entiers classiques en λC-calcul. *Informatique Théorique et Application*, **29-4**, 293-313, 1995.

[35] K. Nour. A general type for storage operators. *Mathematical Logic Quarterly* **41**, 505-514, 1995.

[36] K. Nour. Caractérisation opérationnelle des enties classiques en λC-calcul. *CRAS. Paris* **320, Série I**, 1431-1434, 1995.

[37] K. Nour. Storage operators and $\forall$-positive types of system TTR. *Mathematical Logic Quarterly* **42**, 349-368, 1996.

[38] K. Nour. Opérateurs de mise en mémoire et types $\forall$-positifs. *Informatique Théorique et Application* **30-3**, 261-293, 1996.

[39] K. Nour. An example of a non adequate numeral system. *CRAS. Paris* **323, Série I**, 439-442, 1996.

[40] K. Nour. La valeur d'un entier classique en $\lambda\mu$-calcul. *Archive for Mathematical Logic* **36**, 461-473, 1997.

[41] K. Nour. A conjecture on numeral systems. *Notre Dame Journal of Formal Logic*, **38**, 270-275, 1997.

[42] K. Nour. S-storage operators. *Mathematical Logic Quarterly* **44**, 99-108, 1998.

[43] K. Nour. On storage operators. *Proceedings of a congress "25 Years of Constructive Type Theory" Held in Venice, 19-21 Octobre 1995* 173-190, Oxford University Press, 1998.

[44] K. Nour. Une réponse négative à la conjecture de E.Tronci pour les systèmes numériques typés. *Informatique Théorique et Application* **31-6**, 539-558, 1998.

[45] K. Nour. Mixed logic and storage operators. *A paraitre dans Archive for Mathematical Logic, 2000.*

[46] K. Nour and A. Nour. Propositional mixed logic : its syntax and semantics. *Soumis à Notre Dame Journal of Formal Logic*, 1998.

[47] K. Nour. Les I-types du système F. *Soumis à Informatique Théorique et Application*, 1999.

[48] K. Nour. A non-deterministic classical logic (the $\lambda\mu^{++}$-calculus). *Submitted to Theoretical Computer Science*, 2000.

[49] K. Nour and C. Raffalli. Simple proof of the completeness theorem for second order classical and intuitionictic logic. *En préparation*, 2000.

[50] M. Parigot. Programming with proofs : a second order type theory. *ESOP'88, Lecture Notes in Computer Science* **300**, 145-159, 1988.

[51] M. Parigot. On representation of data in lambda calculus. *To appear in LNCS.*

[52] M. Parigot. Recursive programming with proofs. *Theoretical Computer Science* **94** 335-356, 1992.

[53] M. Parigot. $\lambda\mu$-calculus : an algorithm interpretation of classical natural deduction. *Lecture Notes in Artificial Intelligence, Springer Verlag* **624**, 190-201, 1992

[54] M. Parigot. Classical proofs as programs. *Lectures Notes in Computer Science, Springer Verlag* **713**, 263-276, 1992

[55] M. Parigot. Strong normalization for second order classical natural deduction. *Proceedings of the eighth annual IEEE symposium on logic in computer science* 39-46, 1993.

[56] W. Py. Confluence en $\lambda\mu$-calcul. *Thèse de doctorat, Université de Savoie*, 1998.

[57] C. Raffalli. A semantical storage operator theorem for all types. *Annals of pure and applied logic* **91**, 17-31, 1998.

[58] D. Van Dalen. Intuitionistic logic. *Handbook of Philosophical Logic, Vol III : Alternatives to Classical Logic, D. Gabbay and F. Guenthner (ed.), Chapitre III. 4, D. Reidel Pub. Co.*, 225-339, 1986.

[59] D. Van Dalen. *Logic and structure* Spriger-Verlag Berlin Heidelberg, 1994.